VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZDOKONALENIE PRAVDEPODOBNOSTNÝCH METÓD PRE LÁMANIE HESIEL

ENHANCEMENT OF PROBABILISTIC METHODS FOR PASSWORD CRACKING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FILIP LIŠTIAK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK HRANICKÝ,**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Lištiak Filip, Bc.**
Program: Informační technologie Obor: Matematické metody v informačních technologiích
Název: **Zdokonalení pravděpodobnostních metod pro lámání hesel**
Enhancement of Probabilistic Methods for Password Cracking
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s možnostmi využití pravděpodobnostních bezkontextových gramatik pro účely lámání hesel a s existujícím řešením PCFG Cracker (M. Weir).
2. Navrhněte zdokonalení současných přístupů, které umožní zredukovat množství generovaných hesel při zachování přijatelné míry úspěšnosti nebo zvýší úspěšnost existujících metod.
3. Navržené zdokonalení implementujte formou rozšíření nástroje PCFG Cracker, nebo jako vlastní nástroj.
4. S použitím vhodně zvolených trénovacích a testovacích sad hesel srovnejte zdokonalené řešení s původním řešením.
5. Zhodnoťte dosažené výsledky.

Literatura:

- WEIR, M., AGGARVAL, S., DE MEDEIROS, B., GLODEK, B. Password Cracking Using Probabilistic Context-Free Grammars. In: *30th IEEE Symposium on Security and Privacy*. Berkeley (CA): IEEE 2009. s. 391-405. ISBN 978-0-7695-3633-0.
- S. Houshmand, S. Aggarwal and R. Flood, "Next Gen PCFG Password Cracking," in *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, s. 1776-1791, Aug. 2015. ISSN 1556-6013.
- MA, J., YANG, W., LUO, M., LI, N. A Study of Probabilistic Password Models. In: *IEEE Symposium on Security and Privacy (SP)*. San Jose (CA): IEEE 2014. s. 689-704. ISSN 1081-6011.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Hranický Radek, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 30. října 2018

Abstrakt

Táto práca sa zaoberá lámaním hesiel pomocou pravdepodobnostných bezkontextových gramatík, konkrétne nástrojom PCFG Cracker. Cieľom práce je návrh a implementácia zdokonalení tohto nástroja, ktoré zredukujú veľkosť výstupných slovníkov pri zachovaní prijateľnej úspešnosti. Práca taktiež rieši kritické miesta, ktoré spomaľujú celkový beh programu. Ďalším cieľom práce je analýza a implementácia cielených útočných slovníkov, ktoré zvýšia rozsah a úspešnosť vygenerovaných hesiel.

Abstract

This thesis describes passwords cracking using probabilistic context-free grammars, specifically PCFG Cracker tool. The aim of the thesis is to design and implement enhancements to this tool, which reduce the size of output dictionaries while maintaining acceptable success rate. This work also solves critical parts in the tool that slow down the overall duration of the program. Another goal of the thesis is to analyze and implement targeted attack dictionaries that increase the scope and success rate of generated passwords.

Kľúčové slová

pcfg, lámanie hesiel, pravdepodobnosť, heslo, bezpečnosť, gramatika, vylepšenie, zdokonalenie, slovník.

Keywords

pcfg, password, cracking, probability, security, grammar, enhancement, improvement, dictionary.

Citácia

LIŠTIAK, Filip. *Zdokonalenie pravdepodobnostných metód pre lámanie hesiel*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický,

Zdokonalenie pravdepodobnostných metód pre lámanie hesiel

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Radka Hranického. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Filip Lištiak
20. mája 2019

Poďakovanie

Rád by som poďakoval svojmu vedúcemu Ing. Radkovi Hranickému za vedenie a poskytnuté vecné pripomienky, ktoré mi pomohli pri spracovaní tejto práce. Taktiež ďakujem Bc. Dávidovi Mikušovi za spoluprácu pri experimentovaní s nástrojom PCFG Cracker.

Obsah

Kapitola 1

Úvod

Heslá sú dôležité pre bezpečnosť v mnohých rôznych doménach ako sú sociálne siete, e-maily, šifrovanie citlivých dát alebo bankovníctvo. Ľahko zapamätateľné heslá vytvorené ľuďmi sú tak kľúčovým prvkom v bezpečnosti týchto systémov. Prevažná väčšina ľudí pri vytváraní hesiel dodržiava bežné vzory – od kapitalizácie prvého písmena po vloženie čísla na konci hesla [1]. Tieto vzory používa viacero techník na lámanie hesiel [14]. Jedna z nich využíva na predpovedanie hesla práve pravdepodobnostné bezkontextové gramatiky. V tomto prístupe je systém vytrénovaný na množine uniknutých hesiel a zostaví sa pravdepodobnostná bezkontextová gramatika. Gramatika sa potom používa na vytvorenie odhadov hesiel podľa pravdepodobnosti.

Dôležitým pokrokom v lámaní hesiel pomocou pravdepodobnostných bezkontextových gramatík je práca, ktorú navrhol M. Weir a kol. [21]. V tejto práci bol vytvorený nástroj PCFG Cracker, ktorý dosiahol vynikajúcu úspešnosť a stal sa základným kameňom pre pravdepodobnostné lámanie hesiel. Nástroj má niekoľko slabých stránok, ktoré ho spomaľujú alebo uberajú výslednú hodnotu. Medzi najväčšie problémy patrí veľkosť vygenerovaných slovníkov. Tie môžu dosiahnuť až niekoľko tisíc miliónov hesiel, čo nie je možné vyskúšať ani pri extrémne dlhých procesoch lámania hesiel. Ďalší problém je pomalé generovanie hesiel. Používateľ nevie, ako dlho bude program bežať a koľko hesiel sa zo vstupnej gramatiky dokáže vygenerovať.

Cieľom tejto práce je zdokonalenie pravdepodobnostných metód v nástroji PCFG Cracker tak, aby sa odstránili spomenuté problémy. Práca obsahuje časovú analýzu programu a označenie kritických častí, ktoré spôsobujú pomalé generovanie hesiel. Práca taktiež rieši otázku kedy ukončiť generovanie hesiel a aká je optimálna veľkosť vygenerovaného slovníka pri zachovaní prijateľnej úspešnosti. Ďalším cieľom je použitie útočných slovníkov ako vstup pre generovanie hesiel.

Kapitola 2 popisuje základný prehľad lámania hesiel, typy útokov a generátorov hesiel a existujúce nástroje na lámanie hesiel. V 3. kapitole je detailne vysvetlený princíp a algoritmus nástroja PCFG Cracker, jeho súčasné vylepšenia a problémy. Kapitola 4 sa zameriava na návrh zdokonalení tohto nástroja, medzi ktoré patrí zredukovanie výstupných slovníkov a použitie útočných slovníkov. Ich implementácia je opísaná v kapitole 5, ktorá vysvetľuje výpočet celkového počtu hesiel a vznik programu PCFG Mower. Posledná kapitola 6 zobrazuje experimenty a výsledky implementovaných techník.

Kapitola 2

Prehľad lámania hesiel

Lámanie hesiel (angl. password cracking) označuje rozličné metódy na odhalenie počítačových hesiel. Lámanie hesla je založené na systematickom výbere hesiel, pričom pre každé vybraté heslo sa overuje jeho správnosť. Proces lámania hesla nakoniec končí uhádnutím hesla alebo vyčerpaním sady predpokladaných hesiel, t.j. heslo sa nepodarilo prelomiť. Algoritmus alebo nástroj na výber hesiel sa označuje ako generátor hesiel. Generátory sa rozdeľujú podľa typu útoku, ktorým chceme heslo prelomiť [10]:

- **Útok hrubou silou** - v definovanej dĺžke sa generuje každá možná permutácia znakov na danej abecede, viac v sekcii 2.2.
- **Slovníkový útok** - generátor číta heslá z vopred definovaného slovníka reprezentovaného textovým súborom, databázou atď. Viac v sekcii 2.3.
- **Útok založený na pravidlách** - generátor používa sofistikované metódy na generovanie hesiel: masky, vopred definované pravidlá, regulárne výrazy, pravdepodobnostné gramatiky [21], Markovove reťazce [6] atď. Viac v sekcii 2.4.

V spojitosti s lámaním hesiel je dôležité mať na pamäti, že existujú dva typy útokov na prelomenie hesla: online a offline [20].

Online útok

Pri online útoku útočník navrhuje systému predpokladané heslá v snahe získať prístup. Dôležitá vlastnosť online útoku je, že zariadenie alebo systém, do ktorého chceme získať prístup je v prevádzke a bezpečnostné opatrenia sú stále aktívne (napr. obmedzenie počtu nesprávnych prihlásení). Častým príkladom online útoku je snaha prihlásiť sa do účtu niekoho iného, či už na webovej stránke alebo na počítači. Ďalšie príklady online útokov sú napríklad prelomenie elektronického zámku, uhádnutie PIN kódu do bankomatu alebo získanie prístupu do uzamknutého mobilného telefónu.

Offline útok

Pri offline útoku získal útočník priamy prístup k zahešovaným heslám alebo zašifrovaným súborom. Dôvodom, prečo sa to nazýva offline útok, je ten, že bezpečnostný systém týchto súborov bol prekonaný a preto útočník nie je obmedzený počtom odhadov, ktoré môže vykonať. Offline útoky sa často objavujú po tom, ako útočník úspešne prenikne do počítača alebo na webovú stránku a snaží sa určiť heslá užívateľov. Prelomenie týchto hesiel môže

poskytnúť útočníkovi prístup na iné stránky alebo zdroje. Offline útoky sú tiež používané vo forenznom prostredí, kde orgány činné v trestnom konaní získajú pevný disk, ale musia si poradiť so zašifrovanými súbormi. Na odšifrovanie týchto súborov sa použije offline útok.

2.1 Existujúce nástroje na lámanie hesiel

Väčšina nástrojov na lámanie hesiel funguje podľa rovnakých zásad. Po prvé, program odhadne (predpokladá) heslo a potom skontroluje, či je toto heslo správne. Iné nástroje využívajú tzv. dúhové tabuľky (angl. rainbow tables, kap. 2.5), ktoré obsahujú veľké množstvo pred-vypočítaných hešov hesiel. Na hádanie hesla väčšina nástrojov používa zoznam slovníkov (napr. slovník anglických slov alebo uniknuté heslá z rôznych databáz). Ďalšie nástroje aplikujú na každé odhadnuté heslo zoznam pravidiel, ktoré s ním manipulujú a vykonávajú drobné úpravy (S sa nahradí za $) [4].

Okrem spôsobu generovania hesiel je dôležité brať v úvahu, ako rýchlo dokáže nástroj heslá generovať a aký hardvér daný nástroj potrebuje. Množstvo nástrojov používa pre svoje výpočty procesor, FPGA alebo grafické karty, ktoré sa vyznačujú vyššou rýchlosťou [17]. Medzi najznámejšie nekomerčné nástroje, ktoré bežia len na jednom zariadení (angl. single-machine) patrí John the Ripper, Cain & Abel a Hashcat[1]. Existujú aj distribuované nástroje na lámanie hesiel:

- komerčné: HashStack[2], Elcomsoft[3], Passware[4],
- nekomerčné: Hashtopolis[5], Fitcrack[6].

John the Ripper

John the Ripper [7] je jedným z najstarších udržiavaných programov na lámanie hesiel. Pôvodne založený na programe Crack, jeho hlavným cieľom boli heše založené na unixovej funkcii Crypt(3). Nástroj bol rozšírený, aby zvládal väčšinu zahešovaných hesiel používaných pri prihlasovaní do počítača alebo na webových stránkach. Nezahŕňa však podporu pre prelomenie zašifrovaných súborov [20]. Veľká výhoda tohto nástroja je, že je voľne šíriteľný a spadá pod licenciu GNU GPL[7]. John the Ripper je veľmi obľúbený v oblasti penetračného testovania.

John the Ripper využíva 2 hlavné typy útokov na prelomenie hesiel, a to slovníkový útok a útok hrubou silou. Nástroj podporuje mnoho operačných systémov, najznámejšie z nich sú Unix, OS windows, DOS, Mac OS a OpenVMS. V nasledujúcom zozname sú predstavené niektoré zaujímavosti, ktoré John the Ripper ponúka [7]:

- Typy hešov hesiel sa detekujú automaticky.
- John the Ripper dokáže prelomiť zašifrované heslá založené na rozdielnych hešoch, medzi ktoré patria: DES, MD5, Blowfish, Kerberos AFS.
- Nástroj môže byť prispôsobený užívateľom.

[1] https://hashcat.net/
[2] https://github.com/clawpack/pyclaw/wiki/HashStack
[3] https://www.elcomsoft.com/
[4] https://www.passware.com/
[5] https://github.com/s3inlc/hashtopolis
[6] https://fitcrack.fit.vutbr.cz/
[7] GNU GPL poskytuje slobodu zdieľať a meniť všetky verzie programu.

- Slovníkový útok ponúka rôzne obmieňacie pravidlá (angl. mangling rules), prvých 10 je znázornených v tabuľke 2.1.

#	Pravidlo
1	Skús slová v pôvodnom stave.
2	Každé alfanumerické slovo preveď na malé písmená.
3	Kapitalizuj každé alfanumerické slovo.
4	Každé alfanumerické slovo preveď na množné číslo a malé písmená.
5	Každé alfanumerické slovo preveď na malé písmená a pridaj na koniec „1“.
6	Kapitalizuj každé alfanumerické slovo a pridaj na koniec „1“.
7	Zduplikuj krátke slová (fred → fredfred).
8	Každé slovo preveď na malé písmená a otoč poradie písmen.
9	Pred každé slovo pridaj „1“.
10	Každé slovo preveď na veľké písmená.

Tabuľka 2.1: Prvých 10 predvolených obmieňacích pravidiel v nástroji John the Ripper.

Cain & Abel

Cain & Abel [13] je veľmi rozšírený program na lámanie hesiel s extrémne veľkou fanúšikovskou základňou. Dôvodom popularity je to, že beží na počítačoch so systémom Windows, je zadarmo, má jednoduché grafické rozhranie a obsahuje mnoho ďalších integrovaných nástrojov. Jedna z populárnych funkcií je tzv. network sniffer, ktorý automaticky uloží všetky heslá a heše hesiel, ktoré vidí v sieti. Ak nástroj zachytí heš hesla, automaticky sa ho pokúsi prelomiť. Táto vlastnosť sa stáva ešte silnejšou vzhľadom k tomu, že Cain & Abel má k dispozícii útok „ARP[8] poisoning“. Pri tomto útoku útočník pošle falošné ARP pakety na počítače v lokálnej sieti s cieľom presmerovať celý svoj sieťový tok k útočníkovi. Nástroj taktiež poskytuje útok „man in the middle“ proti zabezpečenej https prevádzke. Spojením všetkých spomínaných funkcií sa z tohto nástroja stal nielen program na lámanie hesiel, ale aj veľmi efektívny nástroj na získavanie hesiel a hešov z lokálnej siete.

Napriek týmto pridaným funkciám má Cain & Abel pomerne obmedzené lámanie hesiel. Aj keď má zabudovanú podporu pre vytváranie dúhových tabuliek (viac v kapitole 2.5) a schopnosť odosielať heslá do online vyhľadávacích databáz, má veľmi limitovanú prácu s pravidlami, ktoré obmieňajú slová (angl. word mangling). Jediné pravidlá, ktoré nástroj podporuje, je manipulácia s kapitalizáciou písmen, výmeny písmen s číslicami a pripojenie číslic na koniec odhadovaného hesla. To znamená, že Cain & Abel nemá schopnosť prelomiť akékoľvek silné heslo. Navyše metódy útoku hrubej sily podporujú iba frekvenčnú analýzu znakov, čo je tiež veľmi obmedzujúce. Stručne povedané, Cain & Abel je veľmi dobre navrhnutý program, ale je užitočný len na lámanie slabých hesiel.

2.2 Útok hrubou silou

Útoky hrubou silou (angl. brute-force attacks) su veľmi časté a obľúbené vo väčšine nástrojov na lámanie hesiel. Generátor hesiel pre tento typ útoku vytvára všetky možné permutácie nad danou abecedou. S výhodou je využívaný na sofistikovanejšie heslá, ktoré nie je

[8] ARP - Protokol rozlišovania adries (angl. address resolution protocol), ktorý primárne slúži na prekladanie IP adresy na MAC adresu.

možné prelomiť slovníkovým útokom. S narastajúcou dĺžkou hesla exponenciálne narastá čas potrebný na jeho generovanie. Nie je teda možné prelomiť heslo nad určitú dĺžku týmto typom útoku v prijateľnom čase. To znamená, že v prípade lámania dlhšieho hesla sa útočník snaží zredukovať vstupnú abecedu. Niekoľko príkladov zredukovaných vstupných znakov z populárneho nástroja na lámanie hesiel [18] je možné vidieť v tabuľke 2.2.

Názov sady	Znaky
numeric	0123456789
loweralpha	abcdefghijklmnopqrstuvwxyz
loweralpha-numeric	abcdefghijklmnopqrstuvwxyz0123456789
mixalpha	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
mixalpha-numeric	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
mix-all-space	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~‘[]{}|\:;“<>,.?/

Tabuľka 2.2: Príklady vstupnej abecedy pre brute-force útoky [20].

Generátor je však jednoducho paralelizovateľný a veľmi dobre vertikálne aj horizontálne škálovateľný. Teda zväčšovanie výkonnosti je možné jednak v rámci jedného počítača pridávaním výpočetných jednotiek, ale aj pridaním viacerých uzlov do výpočetného klastra [6].

2.3 Slovníkový útok

V slovníkovom útoku (angl. dictionary attack) používa útočník súbory slov, o ktorých sa domnieva, že by mohli tvoriť cielené heslo. Slovníky teda môžu obsahovať uniknuté heslá z rôznych databáz, najpravdepodobnejšie cielené slová alebo všetky slová anglického jazyka. Útočník sa potom snaží obmieňať tieto slová podľa určitých pravidiel, ako napríklad kapitalizácia prvého písmena v slove, pridanie dvoch čísel na koniec slova alebo zmena znaku „a“ na „@“. Existujú 3 možné scenáre, kedy slovníkový útok zlyhá:

1. Cieľové heslo bolo vytvorené metódou, ktorá je odolná voči slovníkovému útoku (náhodne zvolené znaky).
2. Základné slovo, z ktorého bolo heslo vytvorené sa nenachádza v slovníku.
3. Útočník neaplikoval správne obmieňacie pravidlá.

Dôležitá vec je, že úspech slovníkového útoku nezáleží len na výbere slovníkov, ale aj na pravidlách, ktoré sa na slová aplikujú. Počet odhadnutých hesiel, ktoré útočník môže vytvoriť je konečný. Z toho vyplýva, že čím väčší je vstupný slovník tým menej pravidiel je možné na slová použiť, pretože každé obmieňacie pravidlo môže často vytvoriť niekoľko nových hesiel pre každé slovo v slovníku. Podobne, keď chce útočník použiť agresívne pravidlá, ktoré generujú niekoľko tisíc hesiel z jedného slova, tak musí zvoliť veľmi malý cielený slovník len s tými najpravdepodobnejšími slovami. Tieto dve techniky (veľký slovník s jednoduchými pravidlami, malý slovník s komplikovanými pravidlami) sa nemusia navzájom vylučovať. Práve naopak, útočník často vykoná niekoľko pokusov o prelomenie hesla, prvý krát pomocou malého a cieleného slovníka a ak tento pokus zlyhá, použije väčší slovník. Ak zlyhajú všetky pokusy slovníkového útoku, útočník môže stále vyskúšať brute-force pred tým, ako sa vzdá [20].

2.4 Útoky založené na pravidlách

Metódy tohto typu útoku využívajú pravidlá založené na pravdepodobnosti a štatistikách a môžu vyžadovať sadu vzorov, ktoré sú vytvorené buď manuálne alebo automaticky analýzou existujúcich textov a súborov hesiel. Tieto techniky zahŕňajú Markovove reťazce [14], frekvenčnú analýzu znakov, hľadanie najčastejších dvojíc a trojíc znakov, pozície znakov atď. Cieľom je vytvoriť také heslá, ktoré majú vysokú pravdepodobnosť, že si ich užívateľ zvolí [10]. Avšak proti náhodne vygenerovaným heslám neponúka táto technika žiadnu výhodu [15]. Tento prístup využíva nástroj PCFG Cracker, ktorému sa venuje celá kapitola 3.

2.5 Dúhové tabuľky

Dúhové tabuľky (angl. rainbow tables) sú databáze predvypočítaných hešov najčastejších hesiel. Používanie dúhových tabuliek značne zredukuje čas potrebný na výpočet hešov. Ak je však pri hešovaní hesla pridaná soľ[9], dúhové tabuľky už nebudú platné a museli by sa prepočítať. Hlavnou výhodou dúhových tabuliek je orientácia na čas a rýchlosť – všetky heše sú vypočítané len jeden krát. Heše je potom možné aplikovať na viac hesiel používajúcich rovnaké alebo podobné princípy výpočtu. Toto je však kompenzované obrovským objemom dát. Vzhľadom na to, že dúhové tabuľky obsahujú niekoľko miliónov hesiel a dĺžka hešu je napr. 64 znakov (SHA-256 [8]), dostávame sa na desiatky gigabajtov dát [11].

[9]Kryptografická soľ je niekoľko náhodných bitov, ktoré slúžia ako doplňujúci vstup pre hešovaciu funkciu a zmení tým jej výsledok.

Kapitola 3

Pravdepodobnostné gramatiky na lámanie hesiel

Pravdepodobnostné gramatiky (angl. probabilistic context-free grammar) budú v tejto práci označované skratkou **PCFG**. Bezkontextové gramatiky sa už dlho používajú na štúdium prirodzených jazykov [16], kde slúžia na generovanie reťazcov s konkrétnou štruktúrou. Ako bude ukázané v nasledujúcej časti kapitoly, bezkontextové gramatiky poskytujú silnú konštrukciu pre hádanie hesiel. Toho využili Matt Weir a kol. a vytvorili nástroj na lámanie hesiel PCFG Cracker[1], ktorého jadro tvorí práve pravdepodobnostná bezkontextová gramatika [21]. V tejto kapitole bude detailne popísaný algoritmus nástroja PCFG Cracker, existujúce vylepšenia a problémy súčasného riešenia, na ktoré sa táto práca zameriava.

3.1 Úvod do PCFG

Bezkontextová gramatika

Bezkontextová gramatika sa označuje ako G = (**V**, **Σ**, **S**, **P**), kde: **V** je je konečná množina premenných (alebo neterminálov), **Σ** je konečná množina terminálov, **S** je počiatočná premenná a **P** je konečná množina pravidiel tvaru:

$$\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta} \tag{3.1}$$

kde $\boldsymbol{\alpha}$ je premenná a $\boldsymbol{\beta}$ je reťazec pozostávajúci z premenných alebo terminálov. Jazykom gramatiky je súbor reťazcov, ktoré pozostávajú z terminálov a sú odvoditeľné z počiatočného symbolu [23].

Pravdepodobnostná bezkontextová gramatika

Pravdepodobnostná bezkontextová gramatika obsahuje pravdepodobnosti priradené ku každému pravidlu tak, že pre každú premennú je suma pravdepodobností pravidiel rovná 1. Ako bude podrobnejšie popísané v kapitole 3.3, sada pravidiel a súvisiacich pravdepodobností je automaticky odvodená z trénovacích súborov reálnych užívateľských hesiel. Tieto pravidlá sa pokúšajú zachytiť spôsob, akým ľudia vytvárajú svoje heslá. Spolu s tým sa získava aj pravdepodobnosť použitých obmieňacích pravidiel.

[1] https://github.com/lakiw/pcfg_cracker

3.2 Použitie gramatiky

Pre jednoduchosť tejto časti bude znázornená gramatika, ktorá používa počiatočný symbol a premenné $\mathbf{L_n}$, $\mathbf{D_n}$ a $\mathbf{S_n}$ pre zadané hodnoty n. Tieto premenné označujú znaky abecedy (**L**, alpha), číslice (**D**, digits) a špeciálne znaky (**S**, special). Premenná n označuje dĺžku reťazca, ktorý sa má za premennú dosadiť. Premenná $\mathbf{D_2}$ sa teda prepíše ako dvojciferné číslo. Aby sa vyhlo duplikovaným heslám, dve rovnaké premenné nesmú byť nikdy vedľa seba, t.j. $\mathbf{D_1D_1}$ by sa nikdy nemalo vyskytnúť v gramatike, no $\mathbf{D_1L_2D_1}$ je správne.

Reťazec odvodený od počiatočného symbolu sa nazýva vetná forma (môže obsahovať premenné a terminály). Pravdepodobnosť vetnej formy je jednoducho výsledkom pravdepodobností pravidiel použitých pri jej odvodení. Gramatika sa najprv odvodí z tréningovej sady hesiel. Príklad takejto gramatiky (spolu so vstupným slovníkom obsahujúcim dve slová) je uvedený v tabuľke 3.1. Touto gramatikou je možné odvodiť napríklad reťazec „cat4!“ s pravdepodobnosťou 0,04875:

$$S \rightarrow L_3D_1S_1 \rightarrow L_34S_1 \rightarrow L_34! \rightarrow cat4! \tag{3.2}$$

$\boldsymbol{\alpha}$		$\boldsymbol{\beta}$	**Pravdepodobnosť**
S	$\rightarrow$	$D_1L_3S_2$	0,75
S	$\rightarrow$	$L_3D_1S_1$	0,25
D_1	$\rightarrow$	4	0,60
D_1	$\rightarrow$	5	0,20
D_1	$\rightarrow$	6	0,20
S_1	$\rightarrow$	!	0,65
S_1	$\rightarrow$	%	0,30
S_1	$\rightarrow$	#	0,05
S_2	$\rightarrow$	$$	0,70
S_2	$\rightarrow$	**	0,30
L_3	$\rightarrow$	cat	0,50
L_3	$\rightarrow$	hat	0,50

Tabuľka 3.1: Príklad pravdepodobnostnej bezkontextovej gramatiky [21].

Tabuľka 3.2 zobrazuje štruktúry v gramatike. Štruktúry $D_1L_3S_2$ a $L_3D_1S_1$ sa nazývajú základné štruktúry (angl. base structures) a sú špecifické tým, že sa používajú v pravidlách z počiatočného symbolu. Vetné formy obsahujúce iba pseudoterminály alebo terminály sa označujú ako pre-terminálne štruktúry (ďalej budú označované ako pre-terminály). Tieto štruktúry sú veľmi dôležité, pretože pravdepodobnosť spojená s pre-terminálnou štruktúrou sa už nemení s akýmikoľvek ďalšími pravidlami a má teda rovnakú pravdepodobnosť ako výsledný terminálny reťazec. Pri generovaní hesiel podľa pravdepodobnosti to umožňuje pracovať s pre-terminálnou štruktúrou namiesto vypĺňania všetkých premenných terminálnymi

Štruktúra	**Príklad**
Základná (base)	$L_3D_1S_1$
Pre-terminálna (pre-terminal)	$L_34!$
Terminálna (terminal)	cat4!

Tabuľka 3.2: Základné štruktúry v gramatike [21].

hodnotami. Pretože jedna pre-terminálna štruktúra môže často generovať tisíce jedinečných terminálnych reťazcov, je tento prístup nevyhnutný pre efektívny beh algoritmu. Premenná L_3 označuje množinu slov o dĺžke 3 písmená (v tomto prípade „cat“ a „hat“) a koncové heslá (cat4!, hat4!) majú tiež pravdepodobnosť 0,04875 [21].

3.3 Trénovanie

Trénovanie je fáza, v ktorej sa vytvára skutočná pravdepodobnostná bezkontextová gramatika. Prebieha typicky na množine uniknutých alebo zverejnených hesiel. Po vytvorení výslednej gramatiky je možné ju použiť na následné lámania hesiel. Výhoda tohto prístupu spočíva v tom, že sa gramatika po vygenerovaní môže distribuovať bez toho, aby bol užívateľom poskytnutý pôvodný zoznam hesiel, na ktorom bola gramatika vytvorená.

Prvý krok k vytvoreniu gramatiky je získanie vhodných vstupných dát. Niektoré uniknuté súbory hesiel sú voľne dostupné na portále GitHub[2] a budú slúžiť ako príklad pre túto kapitolu, konkrétne súbor *rockyou-75.txt*, ktorý obsahuje 59188 hesiel. Druhý krok je spočítanie pravdepodobností jednotlivých čísel, znakov a špeciálnych symbolov, ktoré sa nachádzajú v trénovacích dátach. Získa sa počet všetkých reťazcov o dĺžke n a ku každému reťazcu sa priradí pravdepodobnosť podľa vzorca:

$$P(x) = \frac{X_n}{K_n}, \tag{3.3}$$

kde X_n je počet výskytov reťazca x a X_k je počet všetkých reťazcov daného typu (znaky, čísla, špeciálne symboly) o dĺžke n [21]. Reťazce spolu so svojími pravdepodobnosťami sú následne uložené do textových súborov, ktoré sú rozdelené podľa typu reťazca a jeho dĺžky. Napríklad všetky dvojciferné čísla D_2 budú uložené v súbore *digits2.txt*. V tabuľke 3.3 sa nachádza 10 najpravdepodobnejších dvojciferných čísel v trénovacom súbore *rockyou-75*.

Číslo	**Počet výskytov**	**Pravdepodobnosť (%)**
12	784	11,2
13	511	7,3
11	378	5,4
01	343	4,9
22	315	4,5
07	287	4,1
21	280	4,0
14	280	4,0
23	266	3,8
10	264	3,8

Tabuľka 3.3: Top 10 dvojciferných čísel.

Ďalším krokom je automatické odvodenie všetkých základných štruktúr spolu s ich pravdepodobnosťami. Pravdepodobnosť priradená každej základnej štruktúre sa vypočíta ako:

$$P(x) = \frac{X_n}{K_{total}}, \tag{3.4}$$

[2]https://github.com/danielmiessler/SecLists/tree/master/Passwords

kde X_n je počet výskytov konkrétnej základnej štruktúry a K_{total} je celkový počet všetkých spracovaných základných štruktúr, t.j. počet všetkých hesiel v trénovacom súbore [21]. Príklad základných štruktúr je znázornený v tabuľke 3.4, kde sa z tréningového súboru *rockyou-75* zobrazuje 10 najčastejších základných štruktúr. Väčšina používateľov si zvolí svoje heslo ako obyčajné šesť až sedem znakové slovo a potom zaňho pripojí jednu alebo dve číslice [1, 2]. Toto sa môže zmeniť v závislosti od trénovacieho súboru, požiadavkov na vytvorenie hesla alebo dôležitosti chráneného materiálu. Preto je možné vytvoriť niekoľko rôznych gramatík založených na rôznych tréningových súboroch a pri lámaní hesla vybrať takú gramatiku, ktorá sa najviac zhoduje s profilom cieľa.

Základná štruktúra	**Počet výskytov**	**Pravdepodobnosť (%)**
L_6	10264	17,3
D_6	9376	15,8
L_7	6806	11,4
L_8	5555	9,3
L_5	3004	5,0
L_6D_1	2949	4,9
L_5D_1	2400	4,0
L_6D_2	1903	3,2
L_5D_2	1891	3,1
L_9	1798	3,0

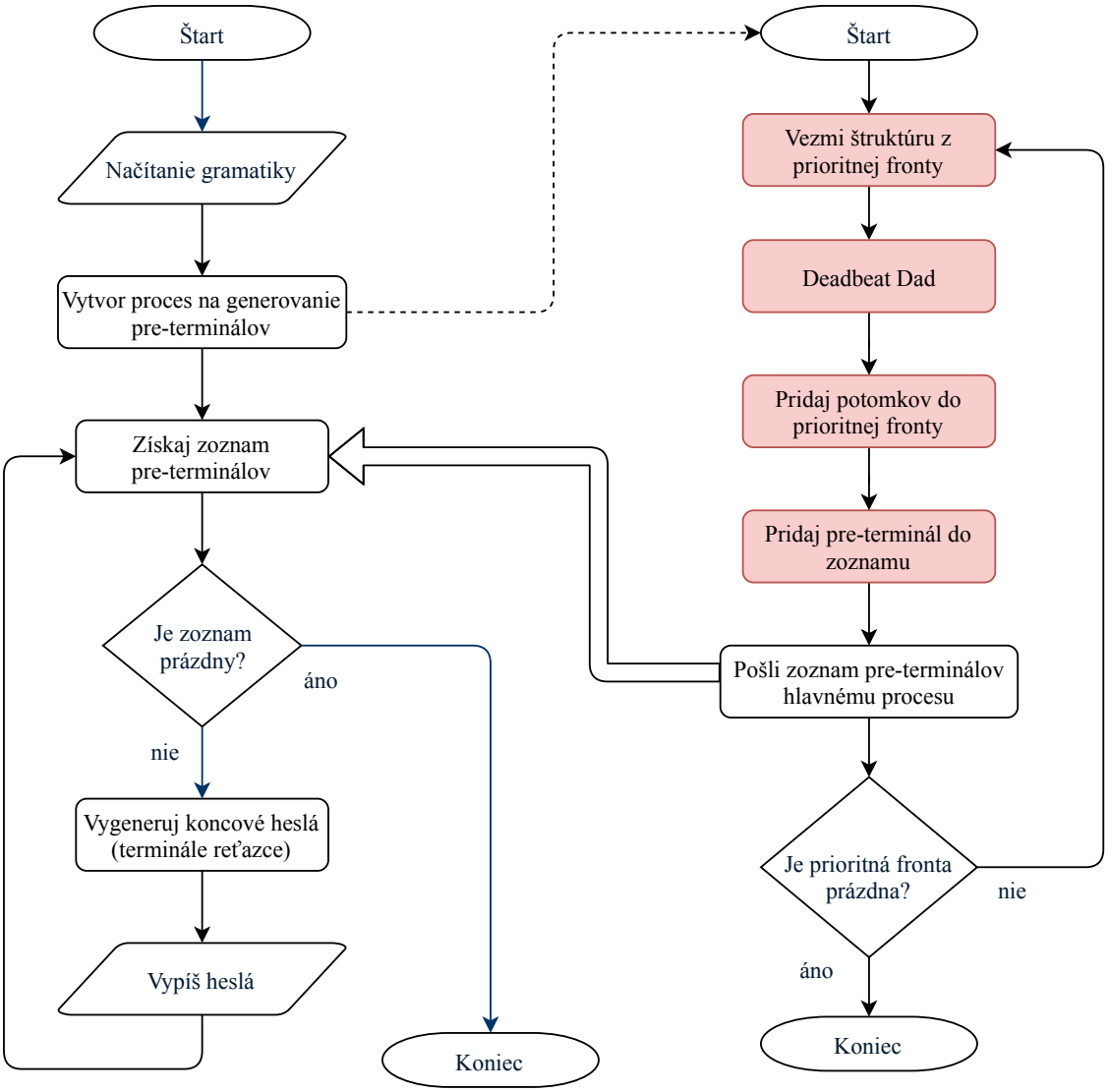
Tabuľka 3.4: Top 10 základných štruktúr.

3.4 Prehľad algoritmu nástroja PCFG Cracker

Použitie pravdepodobnostnej bezkontextovej gramatiky je len prvý krok. Aby bola gramatika užitočná pri lámaní hesiel tak s ňou musí pracovať rýchly a efektívny algoritmus, ktorý berie túto gramatiku ako vstup a generuje heslá podľa pravdepodobnosti.

Vytvorenie najpravdepodobnejšieho hesla je triviálna úloha. Stačí vyplniť všetky základné štruktúry (viď tabuľka 3.2) znakmi s najväčšou pravdepodobnosťou a potom zvoliť najpravdepodobnejšiu pre-terminálnu štruktúru. Napríklad pre gramatiku v tabuľke 3.1 je najpravdepodobnejšia pre-terminálna štruktúra $4L_3\$\$$. Pre ďalšie heslá je však nutné zvoliť sofistikovanejší prístup.

Ako je znázornené na obrázku 3.1, algoritmus začína načítaním natrénovanej gramatiky do internej štruktúry. Následne sa vytvorí vedľajší proces, ktorý slúži na generovanie pre-terminálov. Súčasné riešenie používa ako hlavnú dátovú štruktúru štandardnú prioritnú frontu, kde položka na začiatku fronty obsahuje najpravdepodobnejšiu pre-terminálnu štruktúru. Farebne sú zvýraznené 4 kroky, ktoré tvoria jadro celého programu a sú zodpovedné za výber ďalšieho najpravdepodobnejšieho pre-terminálu a generovanie jeho potomkov. Na generovanie potomkov podľa pravdepodobnosti slúži algoritmus *Deadbeat Dad*, ktorý je detailne opísaný v kapitole 3.5. Vygenerovaní potomkovia sa pridajú do prioritnej fronty a rodičovský pre-terminál sa vloží do zoznamu spracovaných pre-terminálov, ktorý sa pravidelne posiela späť hlavnému procesu. Ten doplní do pre-terminálu znaky a vygeneruje tak koncové heslo. Podľa veľkosti trénovacích dát je možné z jedného pre-terminálu vygenerovať až tisíce hesiel. Gramatika v tabuľke 3.1 teda vygeneruje z pre-terminálu $4L_3\$\$$ heslá 4cat$$ a 4hat$$.



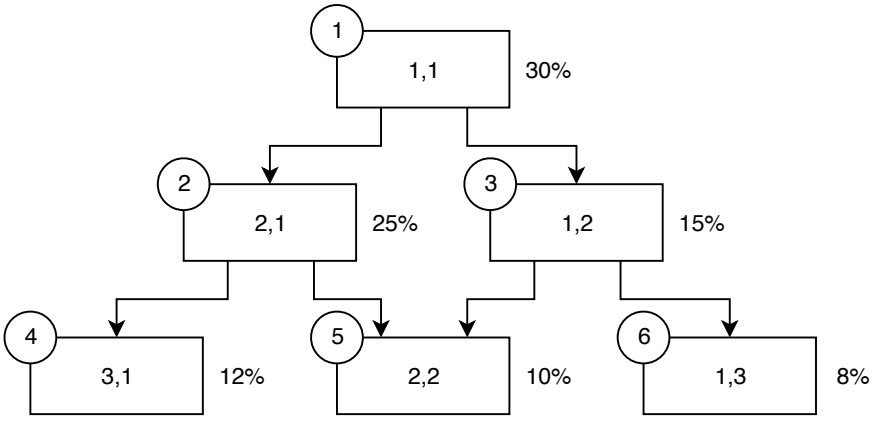
Obr. 3.1: Schéma nástroja PCFG Cracker.

3.5 Deadbeat Dad

Vetné formy v každej pravdepodobnostnej bezkontextovej gramatike vytvárajú stromovú štruktúru, kde počiatočný symbol označuje koreň stromu a terminálne štruktúry označujú listy. Každá hrana značí prepisovacie pravidlo, ktoré transformuje rodičovský uzol na potomka. Pri generovaní hesiel bezkontextovou gramatikou je veľmi dôležité, aby každý terminálny reťazec, ktorý existuje v gramatike, bolo možné vygenerovať presne jednou sadou pravidiel. Podstata algoritmu *Deadbeat Dad* je zaručiť, že pre každý uzol v derivačnom strome existuje jeden konkrétny rodič, ktorý môže tento uzol vygenerovať. Zároveň sa algoritmus snaží držať v prioritnej fronte čo najmenej uzlov a tým znižuje pamäťovú náročnosť programu [20].

Kľúčom na zníženie využitia pamäte je zdržať pridávanie vygenerovaných potomkov do prioritnej fronty na tak dlho, ako je možné. *Deadbeat Dad* tento problém rieši tak, že niektoré

rodičovské uzly sa vzdajú svojich vygenerovaných potomkov a nepridajú ich do prioritnej fronty. Jednoduchý príklad derivačného stromu je možné vidieť na obrázku 3.2. Vpravo od každého uzla je označená pravdepodobnosť jeho výslednej hodnoty. Pravdepodobnosť potomka bude vždy menšia ako pravdepodobnosť jeho rodičovského uzla. Keďže prvky sa do prioritnej fronty vkladajú podľa pravdepodobnosti, tak je nutné, aby sa vždy spracovali všetci potencionálni rodičia konkrétneho uzla a až potom samotný uzol. Potomok by mal byť teda vložený do prioritnej fronty rodičom s najnižšou pravdepodobnosťou [20].



Obr. 3.2: Príklad viacerých rodičov pre uzol č. 5.

Strom na obrázku 3.2 sa bude teda spracovávať následovne:

1. Spracuje sa uzol #1 a vygenerujú sa dvaja jeho potomkovia (#2 a #3) a vložia sa do prioritnej fronty.
2. Spracuje sa uzol #2 a vygenerujú sa jeho potomkovia (#4 a #5), no do prioritnej fronty sa vloží len uzol #4, pretože uzol #5 má ešte druhého rodiča, ktorý zatiaľ nebol spracovaný.
3. Spracuje sa uzol #3, vygenerujú sa jeho potomkovia (#5, #6) a vložia sa do prioritnej fronty.

3.6 Existujúce vylepšenia nástroja PCFG Cracker

Kapitalizácia

Toto rozšírenie pridáva pravidlá kapitalizácie alfa reťazcov do pravdepodobnostnej gramatiky. Je to nevyhnutné na prelomenie silných hesiel, v ktorých ľudia používajú veľké a malé písmená. Pri trénovaní sa získa maska všetkých alfa reťazcov reprezentujúca veľké (**U**, angl. uppercase) a malé (**N**, angl. not uppercase) písmená. Napríklad heslo „PassWord“ bude mať masku $U_1N_3U_1N_4$. Rovnako ako pri ostatných pravidlách sa ku každej maske vypočíta jej pravdepodobnosť [20].

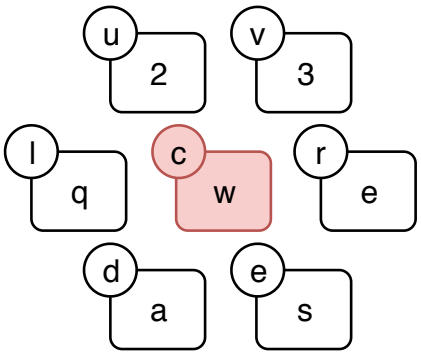
Tabuľka 3.5 zobrazuje 5 najpravdepodobnejších masiek v trénovacom súbore *MySpace*. Výsledky z tabuľky jasne ukazujú, že pokiaľ pri vytváraní hesla nie je povinné aspoň jedno veľké písmeno, tak ich používatelia len veľmi zriedkavo zahrnú do svojho hesla.

Maska	Počet výskytov	Pravdepodobnosť (%)
N_6	7080	93,2
U_1N_5	241	3,1
U_6	222	2,9
N_3U_3	8	0,1
$U_1N_4U_1$	6	0,008

Tabuľka 3.5: Pravdepodobnosť prvých 5 pravidiel kapitalizácie [20].

Klávesové vzory

Klávesové vzory sú fyzické vzory na klávesnici, ktoré sa ľahko pamätajú kvôli svojmu tvaru. Niekedy sa môže zdať, že klávesové vzory pravdepodobne povedú k silným heslám, pretože vytvárajú zdanlivo náhodné reťazce a je možné si ich ľahko zapamätať. V rámci nástroja PCFG Cracker boli implementované metódy, ktoré dokážu extrahovať klávesové vzory a použiť ich pri generovaní hesiel.



Obr. 3.3: Označenie susediacich kláves pre klávesové vzory.

Klávesové vzory sú modelované ako sekvencia susediacich znakov začínajúca od určitej klávesy. Susediace znaky sú definované ako dve klávesy, ktoré ležia fyzicky vedľa seba alebo dva tie isté znaky. Algoritmus nájde vždy najdlhšiu možnú sekvenciu takýchto znakov. Každý klávesový vzor má minimálnu dĺžku 3 znaky. Označenie susediacich kláves pre jednoduchý zápis vzorov je znázornený na obrázku 3.3: **c** označuje aktuálnu klávesu, **u** označuje ľavú hornú, **v** označuje pravú hornú, **l** označuje ľavú, **r** označuje pravú, **d** označuje ľavú dolnú a **e** označuje pravú dolnú. Klávesový vzor **qw34** bude teda charakterizovaný ako **rvr** [9].

Klávesové vzory sa v gramatike označujú novou premennou **K**. Aby gramatika zostala aj po zavedení klávesových vzorov jednoznačná, nasledujúce dva body vysvetľujú kedy je alebo nie je daný klávesový vzor označený **K** namiesto pôvodného označenia **L**, **D**, **S**:

1. Ak je reťazec zložený čisto len z číslic alebo len zo špeciálnych znakov, je označený ako D alebo S.

2. Klávesové vzory, ktoré nespĺňajú prvé pravidlo a majú minimálnu dĺžku 3 znaky sú označené ako K.

Napríklad heslo qwerty7800 bude mať základnú štruktúru K_8D_2 namiesto L_6D_4. Ďalšie príklady základných štruktúr zobrazuje tabuľka 3.6.

Heslo	Základná štruktúra	Štruktúra s klávesovými vzormi
asdf	L4	K_4
q1q1	$L_1D_1L_1D_1$	K_4
ASD1234QW	$L_3D_4L_2$	$K_3D_4L_2$
$%^&	S_4	S_4
qas12saq	$L_3D_2L_3$	$K_3D_2K_3$
q1!2	$L_1D_1S_1D_1$	K_4

Tabuľka 3.6: Základné štruktúry s použitím klávesových vzorov [9].

Vyhladzovanie pravdepodobnosti

Vyhladzovanie pravdepodobnosti (alebo Laplacove vyhladzovanie) zaisťuje, že každý symbol, ktorý nebol nájdený v trénovacom súbore bude mať priradenú nízku pravdepodobnosť. Táto metóda zahŕňa len čísla, špeciálne znaky a klávesové vzory po určitú maximálnu dĺžku.

Ako príklad sa uvažujú hodnoty v C rôznych kategóriách, N_i ako počet prvkov v kategórii i, N je celkový počet prvkov vo všetkých kategóriách. Laplacove vyhladzovanie priradí prvku v kategórii i pravdepodobnosť [9]:

$$Prob(i) = (N_i + \alpha)/(N + C * \alpha) \tag{3.5}$$

V tomto vzorci sa premenná α pohybuje medzi 0 a 1, kde 0 reprezentuje žiadne, prípadne vypnuté vyhladzovanie a 1 reprezentuje veľký stupeň vyhladzovania. Pre $\alpha = 1$, ak sa v trénovacom súbore nachádza 500 dvojciferných čísel ale číslo 33 sa v ňom nevyskytuje, bude mu priradená pravdepodobnosť: $Prob(33) \approx 0,0017$.

3.7 Problémy súčasného riešenia

Táto sekcia popisuje problémy súčasného riešenia nástroja PCFG Cracker. Najväčší z nich je veľkosť výstupných súborov, ktoré dokáže nástroj vygenerovať. Ďalší problém vzniká tým, že ako implementačný jazyk bol zvolený *python*, ktorý nepatrí zrovna medzi najrýchlejšie jazyky.

Veľkosť vygenerovaných slovníkov

PCFG Cracker dokáže vygenerovať tisíce reťazcov z jedného pre-terminálu. Výstupné súbory hesiel tak môžu dosiahnuť aj niekoľko terabajtov. V tabuľke 3.7 je zobrazený počet vygenerovaných reťazcov z rôznych pre-terminálov. Na natrénovanie tejto gramatiky bol použitý súbor *rockyou-75*. Počet vygenerovaných reťazcov ovplyvňujú 2 faktory:

1. veľkosť trénovacieho súboru,
2. dĺžka základnej štruktúry.

Čím väčší je trénovací súbor, tým gramatika získa viac alfa reťazcov, ktoré sa dopĺňajú do pre-terminálov. Použitý trénovací súbor obsahuje až 9582 6-znakových reťazcov a preto pre-terminály, ktoré obsahujú L_6 budú generovať najviac reťazcov.

Dĺžka základnej štruktúry je veľmi dôležitý faktor pre výkon programu a počet vygenerovaných terminálov. Ak štruktúra obsahuje $L_6D_1L_6$, algoritmus vytvorí permutáciu všetkých možných hesiel po dosadení L_6. S každým ďalším prvkom v základnej štruktúre

Pre-terminál	Počet reťazcov	Príklad reťazcu
L_7520	228 244	partner520
L_612L_6	1 950 172	philly12treble
$L_71L_612L_5$	42 952 336	partner1treble12angel
L_584265	60 192	wyatt84265
$L_3.L_3$	29 929	cky.cky
L_53200	115 605	uraqt3200

Tabuľka 3.7: Počet vygenerovaných reťazcov z pre-terminálu.

sa môže počet vygenerovaných reťazcov z tejto základnej štruktúry exponenciálne zvýšiť. Preto pre-terminál $L_71L_612L_5$ vygeneruje až 42 952 336 hesiel.

Trénovací súbor		Vygenerovaný slovník	
Názov	**Počet hesiel**	**Veľkosť**	**Počet hesiel**
UserPassCombo-Jay	700	111 MB	9 M
cirt-default-passwords	1000	5 GB	218 M
darkweb2017-top10000	10000	36 GB	1470 M
myspace	37000	>100 GB	>3000 M
rockyou-75	59000	>500 GB	>13000 M

Tabuľka 3.8: Veľkosť vygenerovaných slovníkov.

Finálne vygenerované slovníky obsahujú obrovské množstvo hesiel a zaberajú niekoľko gigabajtov. Od určitej veľkosti trénovacieho súboru je nemožné aby program vygeneroval úplne všetky reťazce a dobehol v rozumnom čase. Tabuľka 3.8 zobrazuje veľkosti vygenerovaných slovníkov v závislosti na trénovacom súbore.

Pomalé generovanie hesiel

Pomalé generovanie hesiel zapríčiňuje hlavne výber pomalého skriptovacieho jazyka *python* a neefektívne algoritmické konštrukcie, pri ktorých má sám autor poznámku, no doteraz neboli upravené. Obrázok 3.4 zobrazuje časový profil programu PCFG Cracker. Najkritickejšie funkcie sú jasne zvýraznené a menia sa v závislosti od použitej gramatiky, kde môžu nastať 2 hraničné situácie:

1. Gramatika obsahuje veľmi veľa zložitých a dlhých základných štruktúr.
2. Gramatika obsahuje len jednoduché (1-2 prvkové) základné štruktúry.

V prípade prvého bodu sa najkritickejšie miesto stane algoritmus *Deadbeat Dad*. V nástroji PCFG Cracker je každá základná štruktúra spracovaná práve týmto algoritmom [20]. So spracovaním dlhých základných štruktúr má obrovské časové problémy a program sa tak niekoľkonásobne spomalí. Podobnú situáciu znázorňuje práve obrázok 3.4, kde z celkového času 25 sekúnd strávi program takmer polovicu vo funkcii *Deadbeat Dad*. Časové problémy vytvára jeho hlavná časť `DDIsMyParent`, ktorá generuje všetkých možných rodičov všetkých uzlov, ktoré vkladá do prioritnej fronty. Ak je uzol tvorený z dlhej štruktúry, hľadanie všetkých jeho rodičov je výpočetne a časovo veľmi náročné. Funkcia `DDISMyParent` prebieha v iteráciách, ktorých počet sa zvyšuje s každým symbolom v základnej štruktúre.

Rozhodujúci faktor je počet rozdielnych pravdepodobností, ktoré obsahujú prepisovacie pravidlá aplikovateľné na symboly v základnej štruktúre. Ak majú všetky pravidlá pre daný symbol rovnakú pravdepodobnosť, počet iterácií sa nezvýši. Čím viac rozdielnych pravdepodobností, tým rapídnejšie rastie počet iterácií po pridaní ďalšieho symbolu do základnej štruktúry.

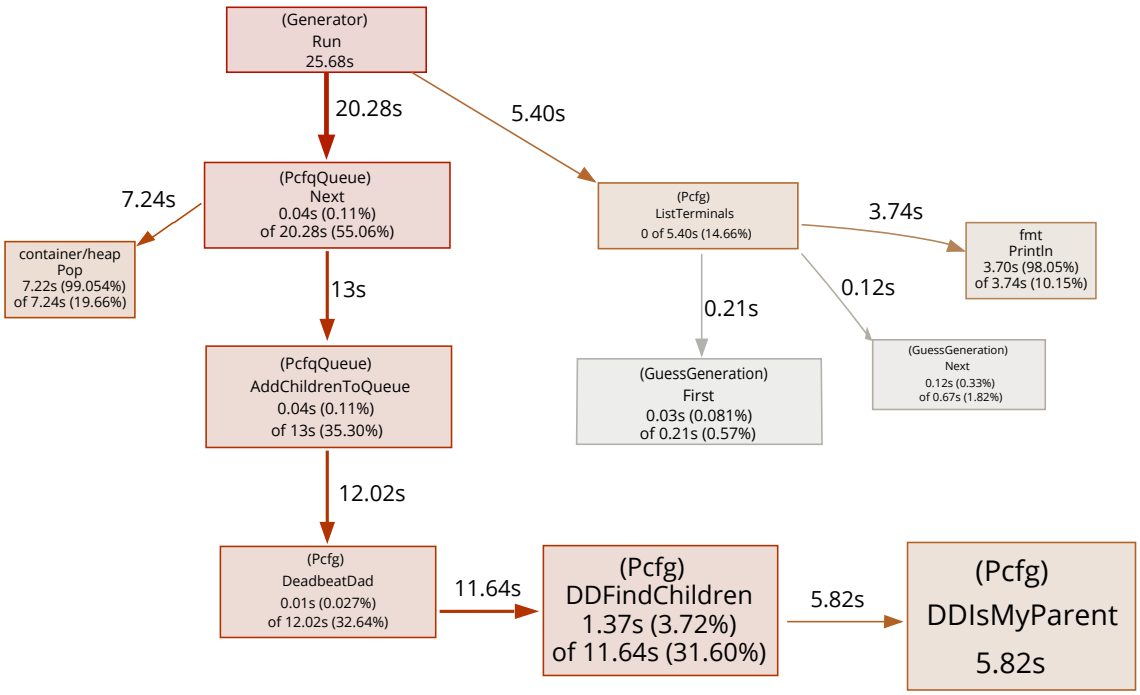
Tabuľka 3.9 zobrazuje počet iterácií funkcie `DDIsMyParent` v závislosti na dĺžke základnej štruktúry. Prepisovacie pravidlá pre symbol D_3 majú rovnakú pravdepodobnosť, takže nemá vplyv na počet iterácií. Pravidlá pre symbol L_1 majú 26 až 29 rozdielnych pravdepodobností, ktoré sa označujú ako pravdepodobnostné skupiny (L_1^p). Pravidlo kapitalizácie pre L_1 je len jedno. V tabuľke je možné vidieť, že počet iterácií stúpa takmer exponenciálne každým pridaným symbolom L_1 do základnej štruktúry.

Základná štruktúra	$\mathbf{L_1^p = 26}$	$\mathbf{L_1^p = 27}$	$\mathbf{L_1^p = 28}$	$\mathbf{L_1^p = 29}$
L_1	103	107	111	115
L_1D_3	103	107	111	115
$L_1D_3L_1$	15811	17067	18371	19723
$L_1D_3L_1D_3$	15811	17067	18371	19723
$L_1D_3L_1D_3L1$	1506286	1688528	1884906	2095948
$L_1D_3L_1D_3L_1D_3$	1506286	1688528	1884906	2095948
$L_1D_3L_1D_3L_1D_3L_1$	120939106	140790314	162990446	187717930

Tabuľka 3.9: Počet iterácií funkcie `DDIsMyParent`.

V gramatike, ktorá bola vytvorená z uniknutých slovníkov hesiel, je rozličnosť pravdepodobností zvyčajne vysoká, obzvlášť pre krátke reťazce. Pri dlhých základných štruktúrach môže funkcia `DDIsMyParent` vykonať niekoľko miliónov iterácií, čo podstatne spomalí generovanie hesiel.

V druhom prípade už nemá *Deadbeat Dad* žiadne spomalenie a program generuje heslá obrovskou rýchlosťou. Jediné kritické miesto sa stáva výpis hesiel a samotný disk. Samotné heslá sa generujú tak rýchlo, že až 90% času z celkového behu programu zaberá výpis hesla a čakanie na disk.



Obr. 3.4: Časový profil programu PCFG Cracker.

Kapitola 4

Návrh zdokonalenia nástroja PCFG Cracker

Táto kapitola popisuje možné návrhy zdokonalenia nástroja PCFG Cracker. Ich cieľom je zredukovať veľkosť výstupných slovníkov a čas potrebný na ich vygenerovanie. Taktiež bude predstavená metóda, ktorá využíva útočné slovníky na zvýšenie úspešnosti vygenerovaných hesiel. Výskum a vývoj ďalších zdokonalení bude prebiehať počas celej tejto práce a preto sa výsledná implementácia môže od pôvodného návrhu líšiť. V nasledujúcej sekcii sú opísané experimenty s týmto nástrojom a z nich vyvodené spôsoby, ako zredukovať problémy nástroja PCFG Cracker vysvetlené v kapitole 3.7. Účel týchto experimentov je potvrdiť alebo vyvrátiť prvotné nápady, ako by sa dali zredukovať vygenerované slovníky. Ďalší zámer experimentov je analyzovať a pochopiť fungovanie programu s rôznymi gramatikami, či už s jednoduchými alebo zložitými.

Aktuálne existujú dve oficiálne verzie tohto nástroja. Prvá verzia je omnoho staršia, archivovaná a napísaná v jazyku *C++*[1]. Táto verzia obsahuje len základné metódy gramatiky bez vylepšení a optimalizácií. Druhá verzia je novšia, udržiavaná, napísaná v jazyku *python*. Obsahuje mnoho ďalších vylepšení z rôznych výskumov [9, 22], no aj od samotného autora. Najväčší rozdiel medzi týmito dvoma verziami je spôsob vytvárania terminálnych reťazcov z pre-terminálov. Zatiaľ čo staršia verzia používa slovníky ako vstupné hodnoty pre alfa reťazce, nová verzia má k dispozícii len tie reťazce, ktoré sa vyskytovali v trénovacom súbore. Tieto detaily sú dôležité pre pochopenie nasledujúcich sekcií.

4.1 PCFG filter

Filtrovanie podľa prahu

Filtrovanie natrénovanej gramatiky bol prvý nápad ako zredukovať veľkosť vygenerovaných slovníkov a zachovať približne rovnakú úspešnosť. Prvotné experimenty boli založené na filtrovaní pravidiel podľa prahu pravdepodobnosti. Pre rôzne typy pravidiel boli zvolené rôzne prahy, t.j. alfa reťazce a čísla mali iný prah ako základné štruktúry. Tento prístup priniesol veľmi zlé výsledky, ktoré budú vysvetlené na zjednodušenom príklade: V gramatike existuje súbor *alpha6.txt*, ktorý obsahuje všetky natrénované 6-znakové slová spolu s ich pravdepodobnosťami. Jednoduchý príklad súboru *alpha6.txt* je v tabuľke 4.1. Cieľom tohto prístupu je odstrániť zo súboru reťazce, ktoré majú nižšiu pravdepodobnosť ako zadaný

[1] https://github.com/lakiw/pcfg_cracker/tree/master/Archived_Work/traditional_pcfg_cracker

#	Pravdepodobnosť	Počet výskytov
1-3	>5%	>25
4-10	3-5%	15-25
10-50	1-2%	5-10
50-500	0,006%	1

Tabuľka 4.1: Príklad súboru alpha6.txt v gramatike.

prah. Podobné rozloženie pravdepodobnosti je takmer v každej gramatike. V tomto súbore je dokopy 500 reťazcov a z nich obrovská časť (90%) má rovnakú pravdepodobnosť, pretože sa v trénovacom súbore vyskytovali iba raz. Ak je prah rovný 0,05%, odstráni sa celá skupina reťazcov, ktoré majú rovnakú pravdepodobnosť 0,006%. Z pôvodného súboru, ktorý obsahoval 500 reťazcov ostane 50, čo je veľmi málo. Nie je možné zvoliť taký prah aby sa zachoval prijatelný počet reťazcov. Tento prístup teda skončil neúspechom.

Pokročilé filtrovanie

Druhý spôsob filtrovania pravidiel je sofistikovanejší a dosahuje veľmi dobré výsledky. Prvá zmena je, že sa filtrujú len základné štruktúry a pravidlá kapitalizácie. Druhý rozdiel je v spôsobe orezávania pravdepodobnosti v danom súbore gramatiky. Každý súbor obsahuje pravidlá a pravdepodobnosti, ktoré dokopy dávajú súčet 100%. Tento prístup sa snaží orezať najmenej pravdepodobné pravidlá tak, aby súčet pravdepodobností zvyšných pravidiel spadal nad prijatelnú hranicu, napr. 90%.

Tréning	**Názov** **Počet hesiel**	Darkweb2017-10000 10000			
Gramatika	**Názov** **Počet ZŠ**[2] **Celková pravdep. ZŠ** **Počet pravidiel kapitalizácie**	Darkweb 307 100% 83	Darkweb1 294 / 307 99,5% 83	Darkweb2 81 / 307 96% 83	Darkweb3 81 / 307 96% 39/83
Vygenerovaný slovník	**Čas** **Veľkosť** **Počet hesiel**	10m 12s 1 GB 100 M	4m 5s 1 GB 100 M	41s 921 MB 100 M	4,6s 94 MB 10,6 M
Lámanie hesiel	**probable-v2** **myspace**	47,4% 26,5%	47,3% 26,7%	47% 25,4%	46,9% 24,5%

Tabuľka 4.2: Úspešnosť rôznych upravených gramatík pomocou filtrovania pravidiel.

Tabuľka 4.2 zobrazuje 4 rôzne gramatiky vytvorené z rovnakého trénovacieho súboru. Pri každej gramatike sú uvedené dôležité parametre, ktoré sa menia filtrovaním. Po spustení nástroja PCFG Cracker s danými gramatikami sú dôležité 3 metriky: trvanie programu, veľkosť a počet vygenerovaných hesiel. Posledná časť tabuľky znázorňuje úspešnosť pri lámaní hesiel v slovníkoch *probable-v2* a *myspace*. Každá z týchto gramatík má vyfiltrované pravidlá do inej miery a každá znázorňuje istý časový pokrok:

Darkweb je pôvodná gramatika vygenerovaná zo vstupného súboru *Darkweb2017-10000* bez žiadnych zásahov. Generovanie hesiel z tejto gramatiky bolo prerušené po dosiahnutí 1 GB výsledných hesiel. Program bežal dokopy 10 minút a 12 sekúnd.

Darkweb1 sa líši od pôvodnej gramatiky tým, že má odstránených 13 dlhých základných štruktúr, ktoré časovo veľmi vyťažovali celý program. Generovanie hesiel bolo taktiež

[2]Základné štruktúry

prerušené po dosiahnutí 1 GB výstupných dát, no tentokrát to programu trvalo len 4 minúty a 5 sekúnd, čo je takmer o 60% väčšia rýchlosť ako pri pôvodnej gramatike. Úspešnosť pri lámaní iného slovníka zostáva takmer rovnaká.

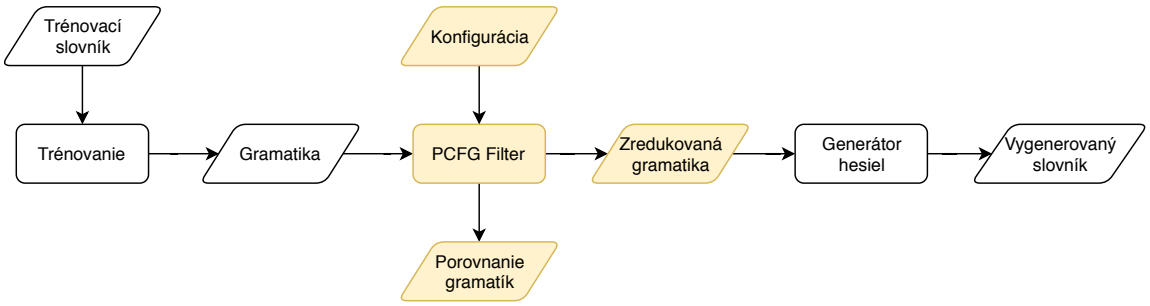
Darkweb2 má značne odfiltrované základné štruktúry (zostalo len 81 z pôvodných 307) a celková pravdepodobnosť týchto štruktúr je 96%, čo je perfektný výsledok. Generovanie hesiel dobehlo za 41 sekúnd a vygenerovaný slovník obsahoval 921 MB, t.j. približne 100 miliónov hesiel. Úspešnosť pri lámaní slovníka *myspace* je o 1% horšia ako pôvodná gramatika.

Darkweb3 je gramatika, ktorá ma spolu so základnými štruktúrami odstránené aj pravidlá kapitalizácie, ktoré mali menšiu pravdepodobnosť ako 1%. Celkovo tak zostalo 39 z pôvodných 83 pravidiel. Generovanie hesiel bežalo len 4,6 sekundy, čo je obrovský časový pokrok a výsledný slovník zaberíá len 94 MB. Úspešnosť oproti pôvodnej gramatike je taktiež veľmi dobrá, v slovníku *probable-v2* sa podarilo prelomiť len o 0,5% hesiel menej a v *myspace* o 2% menej.

Týmto prístupom je teda možné vytvoriť upravenú gramatiku, ktorá vygeneruje niekoľkonásobne menšie slovníky pri zachovaní veľmi dobrej úspešnosti. S takouto upravenou gramatikou má program veľkú šancu, že vygeneruje všetky možné heslá, ktoré sa z danej gramatiky dajú vygenerovať a dobehne v prijateľnom čase.

4.2 Použitie filtru v nástroji PCFG Cracker

Pokročilý PCFG filter bude implementovaný ako voliteľný modul v nástroji PCFG Cracker. Filter bude na vstupe očakávať základnú konfiguráciu, ako napr. celková pravdepodobnosť základných štruktúr, prah pravdepodobnosti pre pravidlá kapitalizácie alebo maximálna povolená dĺžka základných štruktúr. Ďalšia funkcia filtru bude odhadnutie počtu hesiel, ktorý je program schopný vygenerovať z danej gramatiky a s tým spojené aj odhadované trvanie programu. Filter bude taktiež ponúkať rôzne štatistiky a porovnania oproti pôvodnej gramatike, podobne ako v tabuľke 4.2: počet odstránených základných štruktúr a ďalších pravidiel, odhadované zrýchlenie a porovnanie veľkosti výsledného slovníku.



Obr. 4.1: Schéma nástroja PCFG Cracker po pridaní modulu PCFG Filter.

PCFG filter bude zaradený medzi trénovanie a samotné generovanie hesiel podľa obrázku 4.1. Gramatika teda musí byť vytvorená dopredu z nejakého trénovacieho súboru. Po aplikovaní filtru na vytvorenú gramatiku sa používateľ môže rozhodnúť či je predpokladané zrýchlenie dostatočné a následne nad vyfiltrovanou gramatikou spustí generátor.

Akékoľvek odstránenie pravidiel bez úpravy hodnôt pravdepodobnosti má za následok matematicky nesprávnu gramatiku, kde celková pravdepodobnosť pravidiel môže byť nižšia ako jedna. Pre praktické použitie s generátorom hesiel to nevadí. Cieľom filtrovania je vytvoriť kompaktný výstupný slovník a zabezpečiť, aby generovanie hesiel skončilo v prijateľnom čase. Zredukovaná gramatika bude vždy úplne spracovaná, t.j. generátor úspešne dobehne sám bez zásahu užívateľa. To zabezpečí, že aj paralelný beh generátora vygeneruje vždy rovnaké heslá. Najsilnejšou motiváciou pre filtrovanie gramatiky je však potenciálne masívna úspora času procesora. Stanovenie limitu pred začiatkom behu generátora zabráni algoritmu *Deadbeat Dad* vykonávať mnoho zbytočných derivačných krokov na stromoch, ktoré takmer nikdy nevytvoria terminálne heslá kvôli nízkej pravdepodobnosti.

4.3 Útočné slovníky

Ako bolo spomenuté v úvode tejto kapitoly, staršia verzia nástroja PCFG Cracker využívala slovníky ako zdroj alfa reťazcov, no nová verzia túto funkciu nemá. Je to teda ďalšia možnosť, ako zdokonaliť tento nástroj. Ak má byť toto vylepšenie úspešné, je potrebné preskúmať aký je najlepší spôsob výberu útočných slovníkov. Veľkosť a obsah útočných slovníkov môže ovplyvniť pravdepodobnosť, generované heslá, poradie v akom sú generované a aj výslednú efektivitu lámania hesiel. Útočné slovníky, ktoré sa používajú pri lámaní hesiel, sú zvyčajne zoznam bežných hesiel, ktoré už niekedy boli prelomené alebo zoznam anglických slov (alebo českých/slovenských, záleží od cieľa), ktoré boli experimentálne preukázané ako účinné [22].

Cieľom tohto vylepšenia je teda pridať túto funkcionalitu do novej verzie PCFG Cracker a preskúmať, akú účinnosť majú rôzne slovníky s narastajúcim počtom vygenerovaných hesiel. Týmto spôsobom môže program vygenerovať tak veľa hesiel, že aj pri extrémne dlhom procese lámania hesiel nebude možné vyskúšať všetky možnosti. V staršej verzii programu sa všetky slová s rovnakou dĺžkou použijú naraz pri dosadzovaní do pre-terminálu, pretože všetky slová rovnakej dĺžky z útočného slovníku majú rovnakú pravdepodobnosť. Tento spôsob je nutné zmeniť a pridať do programu schopnosť používať viacero útočných slovníkov naraz. Každému slovníku bude možné priradiť inú pravdepodobnosť a tým pádom získame množiny slov s rôznymi pravdepodobnosťami. To programu umožní mať pomerne veľký slovník s nižšími pravdepodobnosťami a menší zoznam bežných hesiel s vyššou pravdepodobnosťou, ktorý bude slúžiť ako sekundárny slovník.

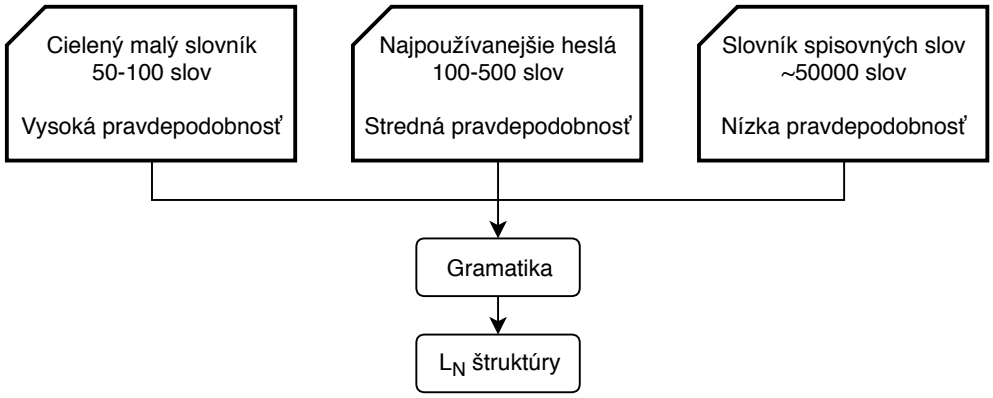
Použitie útočných slovníkov v nástroji PCFG Cracker

Útočné slovníky budú rozdelené minimálne do troch kategórií:

Malé cielené slovníky, ktoré budú obsahovať veľmi málo cielených slov a bude im priradená vysoká pravdepodobnosť. Tento typ slovníku bude vytvorený analýzou cieľa a získaním čo najviac informácií, ktoré by mohli pomôcť pri hádaní hesla – národnosť, mená rodinných príslušníkov, dôležité dátumy a čísla, predošlé zistené heslá atď. Priradení vysoká pravdepodobnosť zaručí, že sa tieto slová použijú pri generovaní hesiel medzi prvými.

Najpoužívanejšie heslá celosvetovo, ktoré sú dostupné na rôznych webových stránkach[3]. Tieto slová budú niesť strednú pravdepodobnosť a budú sa používať priebežne počas celého behu programu.

[3] https://www.passwordrandom.com/most-popular-passwords



Obr. 4.2: Schéma použitia útočných slovníkov pri generovaní hesiel.

Slovník spisovných slov – obrovský slovník anglických / slovenských slov s nízkou pravdepodobnosťou, ktoré program použije až na konci behu pri extrémne dlhých pokusoch o prelomenie hesla.

Tieto slovníky budú spojené s gramatikou počas trénovania. Slová budú rozdelené podľa dĺžky do jednotlivých súborov a bude im priradená pravdepodobnosť podľa typu útočného slovníka, v ktorom sa nachádzajú. Slová sa budú pri generovaní dosadzovať do L_N, D_N a S_N štruktúr.

Kapitola 5

PCFG Mower

V tejto kapitole bude popísaná implementácia programu PCFG Mower, ktorý bol navrhnutý v kapitole 4. Tento program v sebe zahŕňa všetky navrhované vylepšenia – analýza gramatík, vypočítanie celkového počtu hesiel, ktoré je gramatika schopná vygenerovať, orezanie najmenej pravdepodobných pravidiel (PCFG Filter), aplikovanie útočných slovníkov a vytvorenie novej gramatiky s požadovaným počtom hesiel. Ako implementačný jazyk bol zvolený *python*, pre jednoduchšie začlenenie do nástroja PCFG Cracker.

5.1 Konfigurácia a parametre programu

PCFG Mower má povinný len 1 parameter, a to vstupnú gramatiku. Ostatné parametre majú prednastavené hodnoty a sú určené na experimentovanie s gramatikou. Program obsahuje takúto konfiguráciu:

Vstupná gramatika – cesta ku gramatike vytvorenej pomocou programu *pcfg_trainer*.

Výstupná gramatika – názov a cesta výstupnej gramatiky. V prípade, že tento parameter nie je zadaný, program vypíše štatistiky vstupnej gramatiky a ukončí sa.

BS – filtrovací krok pre základné štruktúry (base step).

CS – filtrovací krok pre pravidlá kapitalizácie (capitalization step), viac v 5.4.

Limit – maximálny počet hesiel, ktoré môže vygenerovať výstupná gramatika.

Konfigurácia útočných slovníkov – súbor, ktorý obsahuje názvy útočných slovníkov spolu s ich prioritami.

Tichý mód – nevypisuje žiadny text na štandardný výstup. Poskytuje lepšiu integráciu s inými skriptami.

PCFG Mower je napísaný tak, aby sa dal jednoducho začleniť do iných programov, napríklad do spomínaného nástroja PCFG Cracker. Príklad spustenia programu z príkazovej riadky:

```
python3 pcfg_mower.py -i Myspace                                    (1)

python3 pcfg_mower.py -i Myspace -o Myspace_2 -l 500000             (2)

python3 pcfg_mower.py -i Myspace -o Myspace_3 -a attack_dict_config (3)
```

Vo výpise 5.1 je možnné vidieť príklad výstupu programu PCFG Mower pri spustení príkazu č. 3:

```
limit: 500000000
bs: 0.001
cs: 0.001

Myspace
 Guess count: 12200842908452136311513
 Grammar prob: 1,00
 Grammar size: 1574
 Capitalization cs: 0,001
 Capitalization rules: 179
 Capitalization files: 20
 Capitalization prob: 19,99
 Alpha size: 22587

APPENDED
  best110.txt: 90
  best1050.txt: 928

DUPLICITIES
  best110.txt: 63
  best1050.txt: 465

IGNORED
  best110.txt: 20
  best1050.txt: 121

Myspace_3
 Guess count: 340157791
 Grammar prob: 0,85159
 Grammar size: 78
 Capitalization cs: 0,129
 Capitalization rules: 24
 Capitalization files: 20
 Capitalization prob: 18,65
 Alpha size: 23077
```

Výpis 5.1: Príklad výstupu programu PCFG Mower.

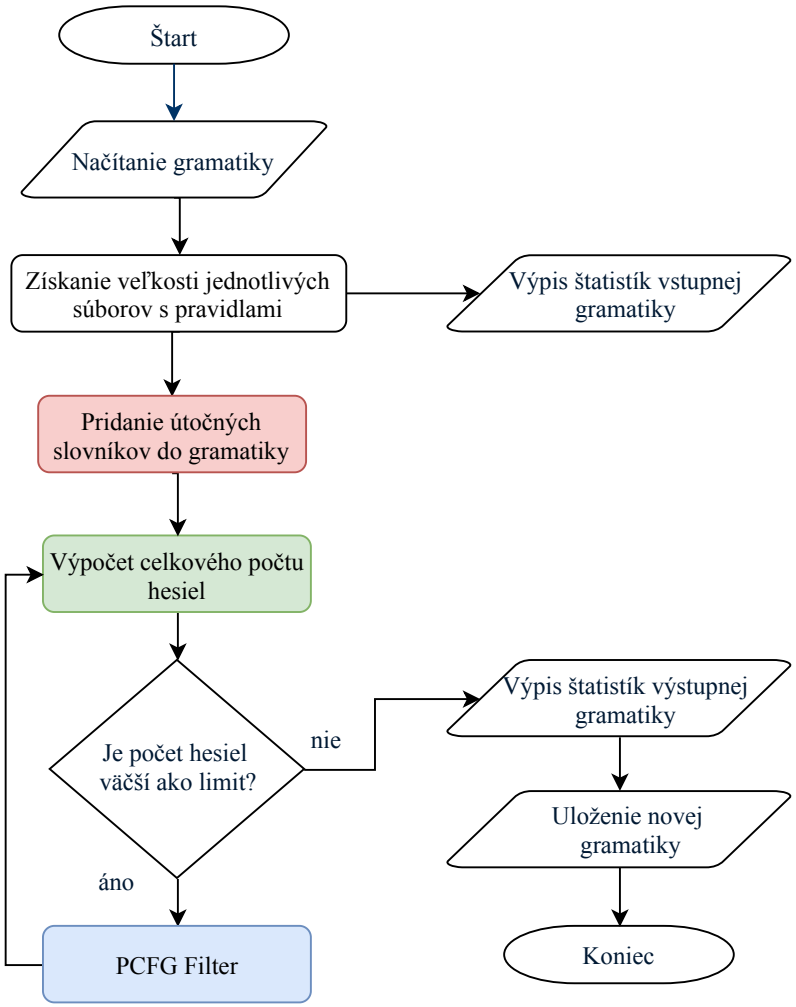
Program ako prvé vypíše svoju základnú konfiguráciu pre filtrovanie gramatiky, a to limit počtu hesiel, filtrovací krok pre základné štruktúry a filtrovací krok pre pravidlá kapitalizácie. Následne vypíše dôležité štatistiky vstupnej gramatiky, ktoré sú kľúčové pri analýze a vyhodnocovaní výsledkov. Tieto vlastnosti gramatiky budú podrobnejšie opísané v kapitole 6. Nasleduje štatistika útočných slovníkov, ktorá obsahuje 3 časti: počet pridaných slov do gramatiky, počet slov, ktoré už v gramatike existovali ale bola im priradená väčšia pravdepodobnosť a počet slov, ktoré sa z útočných slovníkov nepridali do gramatiky. Viac o útočných slovníkoch je možné vidieť v sekcii 5.5. Na konci je vypísaná štatistika novej vyfiltrovanej gramatiky, ktorá je uložená v adresári špecifikovaným parametrom „-o“.

5.2 Popis algoritmu

Algoritmus programu PCFG Mower je zobrazený na obrázku 5.1. Prvý krok je načítanie vstupnej gramatiky do pamäte. Pre lepšiu orientáciu a pochopenie sú použité veľmi podobné štruktúry ako v nástroji PCFG Cracker. Ako typ štruktúr je použité dvojrozmerné asociatívne pole, ktoré umožňuje použiť akýkoľvek typ indexu namiesto číselného. Prvý index označuje typ pravidiel (základné štruktúry, alfa reťazce, čísla, ...) a druhý obsahuje názov súboru daného typu (1.txt, 2.txt). Druhý index sa dá interpretovať tiež ako dĺžka reťazcov, ktoré súbor obsahuje. Hodnoty tejto štruktúry su reprezentované ako pole dvojíc *(reťazec, pravdepodobnosť)*. Príklad takejto štruktúry je v nasledujúcom výpise:

```
ruleset["Grammar"]["Grammar.txt] = [(L8D2, 0,213), (S1L5D1, 0,114)]
ruleset["Alpha"]["8.txt] = [(password, 0,428), (sunshine, 0,028)]
ruleset["Capitalization"]["8.txt] = [(N8, 0,945), (U1N7, 0,055)]
ruleset["Digits"]["2.txt] = [(11, 0,487), (12, 0,249)]
```

Výpis 5.2: Príklad štruktúry gramatiky v programe PCFG Mower.



Obr. 5.1: Diagram programu PCFG Mower.

Ďalším krokom programu je analýza vstupnej gramatiky. Veľkú časť tohto kroku tvorí výpočet veľkostí jednotlivých súborov s pravidlami, ktorá je daná počtom riadkov (počtom pravidiel). Vytvorí sa tým ďalšia štruktúra `sizes`, ktorá namiesto dvojíc *(reťazec, pravdepodobnosť)* obsahuje veľkosť daného súboru. Táto štruktúra bude následne použitá na výpočet celkového počtu hesiel, ktoré dokáže gramatika vygenerovať. Po naplnení týchto štruktúr má program k dispozícii všetky informácie o gramatike a vypíše niektoré jej vlastnosti zobrazené vo výpise 5.1.

Pridanie útočných slovníkov do gramatiky vyznačené červenou farbou je voliteľná funkcionalita programu podrobne opísaná v sekcii 5.5. Podstatou tohto kroku je, že si používateľ môže definovať vlastné súbory hesiel a priradiť im prioritu. Podľa priority sa heslá z útočných slovníkov pridajú do gramatiky, čím sa zvýši počet alfa reťazcov. Cieľom tejto funkcionality je zvýšiť úspešnosť pri lámaní hesiel.

Výpočet celkového počtu hesiel označené zelenou farbou je najdôležitejšia metrika gramatiky v programe PCFG Mower. Celkový počet hesiel je priamo úmerný času generovania týchto hesiel. Keďže hlavným cieľom programu je zredukovať čas a počet vygenerovaných hesiel tak sa táto metrika stáva hlavnou podmienkou pri filtrovaní gramatiky. Výpočet je detailne opísaný v sekcii 5.3.

PCFG Filter je implementovaný podľa návrhu v sekcii 4.1. Jeho hlavnou úlohou je systematicky podľa zadaných krokov (base step, capitalization step) odfiltrovať tie najmenej pravdepodobné pravidlá. Algoritmus je podrobne opísaný v sekcii 5.4. Spolu s výpočtom celkového počtu hesiel tvorí PCFG Filter hlavný cyklus orezávania gramatiky. Tento cyklus sa opakuje dovtedy, kým sa gramatiku podarí orezať na požadovaný počet hesiel.

Posledný krok programu je analýza a výpis štatistík zredukovanej gramatiky. Štatistiky slúžia na porovnanie pôvodnej a zredukovanej gramatiky a lepšiu orientáciu v experimentoch. V štatistikách je možné nájsť aj veľkosť filtrovacích krokov, t.j. do akej miery musela byť gramatika orezaná aby sa dosiahol požadovaný počet hesiel. Následne sa nová gramatika uloží na disk. Ukladanie prebieha tak, že pravidlá, ktoré neboli upravené sa len jednoducho skopírujú z pôvodnej gramatiky. Pravidlá, do ktorých zasiahol PCFG Filter sa vytvoria nanovo z interných štruktúr programu.

5.3 Výpočet celkového počtu hesiel

Výpočet celkového počtu hesiel z gramatiky nie je dostupný v nástroji PCFG Cracker, no je nevyhnutný pre PCFG Mower. Algoritmus 1 zobrazuje princíp tohto výpočtu. Na rýchly a efektívny výpočet sú potrebné 2 údaje: zoznam základných štruktúr a informácie o veľkosti jednotlivých súborov s pravidlami. Všetky tieto údaje sú pripravené v štruktúrach `ruleset` a `sizes`, ktoré algoritmus vyžaduje ako vstupné parametre. Premenná `cnt_all` predstavuje počet hesiel zo všetkých základných štruktúr a premenná `cnt_base` udáva počet hesiel z jednej základnej štruktúry, ktorá sa práve spracováva. Algoritmus začína postupnou iteráciou cez všetky základné štruktúry, ktoré sa uložia do premennej `base_struct`. Každá základná štruktúra má tvar dvojice *(štruktúra, pravdepodobnosť)*. Pravdepodobnosť je pre tento výpočet nepodstatná, pretože nijak neovplyvňuje celkový počet hesiel. Ako príklad bude slúžiť základná štruktúra $\mathbf{S_1L_8D_2}$. Túto štruktúru je nutné spracovať a rozdeliť na typ reťazca (S, L, D) a jeho dĺžku (1, 8, 2) pomocou funkcie `split_alpha_and_digit`. Rozdelená základná štruktúra sa uloží do poľa `base_arr`: [**'S', '1', 'L', '8', 'D', '2'**].

Na piatom riadku nasleduje ďalší cyklus, ktorý prechádza cez vytvorené pole `base_arr` a v každej iterácii získa typ reťazca na pozícii `base_arr[index * 2]` a jeho príslušnú dĺžku `base_arr[index * 2 + 1]`. V prvej iterácii teda algoritmus získa typ **S** a dĺžku **1**.

Na riadku 8 algoritmus zisťuje, koľko reťazcov obsahujú jednotlivé časti základnej štruktúry. V prvej iterácii to bude konkrétne `get_file_size(S,1)`. Táto funkcia vykonáva jednoduchý prevod typu reťazca na jeho plný tvar (S → Special, L → Alpha, ...) a dĺžky na názov súboru (1 → 1.txt, 8 → 8.txt, ...). Následne v štruktúre `sizes` vyhľadá veľkosť tohto súboru: `sizes[Special][1.txt]`. Výsledok sa uloží do premennej `file_size`.

Na získanie počtu hesiel z jednej základnej štruktúry je nutné vynásobiť veľkosti všetkých jej zložiek. V prípade, že základná štruktúra obsahuje $\mathbf{L_n}$ reťazec, tak sa veľkosť príslušného alfa súboru musí vynásobiť počtom pravidiel kapitalizácie ($\mathbf{C}$) pre danú dĺžku $\mathbf{n}$. Posledný krok algoritmu je sčítanie počtu hesiel jednotlivých základných štruktúr.

Nech *size(N)* je počet terminálnych štruktúr, ktoré sa dajú vytvoriť použitím prepisovacích pravidiel na neterminál *N*. Pre základnú štruktúru $B = N_1N_2...N_n$ je celkový počet hesiel daný vzorcom:

$$cnt\_base(B) = \prod_{i=1}^{n} size(N_n). \tag{5.1}$$

To znamená, že počet hesiel zo základnej štruktúry $S_1L_8D_2$ bude následovný:

$$cnt\_base(S_1L_8D_2) = size(S_1) * (size(L_8) * size(C_8)) * size(D_2) \tag{5.2}$$

Pre gramatiku G je celkový počet hesiel rovný súčtu $cnt\_base(B)$ pre každú základnú štruktúru $B \in G$:

$$cnt\_all(G) = \sum_{B \in G} cnt\_base(B). \tag{5.3}$$

Algoritmus 1 Výpočet celkového počtu hesiel.

Vstup: *ruleset*, *sizes*
Výstup: *cnt_all*

```
1:  cnt_all ← 0
2:  for all base_struct ∈ ruleset[Grammar][Grammar.txt] do
3:      cnt_base ← 1
4:      base_arr ← split_alpha_and_digit(base_struct)
5:      for index ← 0 to len(base_arr)/2 do
6:          value ← base_arr[index * 2]
7:          size ← base_arr[index * 2 + 1]
8:          file_size ← get_file_size(value, size)
9:          cnt_base ← cnt_base * file_size
10:     end for
11:     cnt_all ← cnt_all + cnt_base
12: end for
```

5.4 PCFG filter

Filtrovanie pravidiel v gramatike je rozdelené na tri časti: odstránenie dlhých základných štruktúr, filtrovanie základných štruktúr podľa prahu a filtrovanie pravidiel kapitalizácie. Na každú časť je použitá iná technika a taktiež majú aj rozdielny prah, t.j. mieru, do akej sa majú dané pravidlá vymazať. Tieto časti na sebe nie sú nijak závislé a ich algoritmy sú opísané v nasledujúcich sekciách.

Dlhé základné štruktúry

V rámci prvotných experimentov a časového profilu nástroja PCFG Cracker bolo v sekcii 3.7 dokázané, že dlhé základné štruktúry majú obrovský vplyv na rýchlosť generovania hesiel. Heslá, z ktorých sa natrénujú takéto dlhé základné štruktúry sú vo väčšine prípadov náhodné vygenerované heslá, nie vytvorené človekom. Odstránenie dlhých základných štruktúr niekoľkonásobne urýchli generovanie hesiel a takmer vôbec neovplyvní úspešnosť vygenerovaného slovníka pri lámaní hesiel, čo je dokázané v kapitole 6.

Na odstránenie dlhých základných štruktúr slúži skript `remove_long_base.sh` napísaný v jazyku *bash*. Tento skript vyžaduje dva vstupné parametre: cestu k súboru so základnými štruktúrami *Grammar.txt* a maximálnu povolenú dĺžku základných štruktúr. Skript následne prejde všetky štruktúry v súbore a odstráni tie, ktoré sú dlhšie ako maximálna povolená dĺžka.

Filtrovanie základných štruktúr

Algoritmus 2 Filtrovanie základných štruktúr.

Vstup: *bs*

1: $prob\_before\_mowing \leftarrow get\_total\_grammar\_prob()$
2: $prob\_diff \leftarrow 0$
3: **while** $prob\_diff < bs$ **do**
4: $\quad grammar\_len \leftarrow sizes[Grammar][Grammar.txt]$
5: $\quad ruleset[Grammar][Grammar.txt].pop(grammar\_len - 1)$
6: $\quad prob\_after\_mowing \leftarrow get\_total\_grammar\_prob()$
7: $\quad prob\_diff \leftarrow prob\_before\_mowing - prob\_after\_mowing$
8: **end while**

Algoritmus 2 popisuje sofistikované odstraňovanie najmenej pravdepodobných základných štruktúr. Tento algoritmus využíva niekoľko premenných:

bs – filtrovací krok pre základné štruktúry (base step), ktorý udáva o koľko sa môže celková pravdepodobnosť základných štruktúr znížiť v jednej iterácii,

prob_before_mowing – celková pravdepodobnosť základných štruktúr predtým, ako sa začnú odstraňovať,

prob_after_mowing – celková pravdepodobnosť základných štruktúr po odstránení aspoň jedného pravidla,

prob_diff – rozdiel medzi `prob_before_mowing` a `prob_after_mowing`.

Princípom algoritmu je efektívne vymazať naraz niekoľko základných štruktúr aby program nemusel počítať celkový počet hesiel po každej vymazanej základnej štruktúre. Koľko základných štruktúr sa vymaže udáva práve **bs**. Nejde o konkrétny počet základných štruktúr ale o rozdiel celkovej pravdepodobnosti pred a po ukončení odstraňovania pravidiel. Ako príklad budú slúžiť pravidlá v tabuľke 5.1.

V prvom kroku algoritmu sa vypočíta celková pravdepodobnosť základných štruktúr funkciou *get_total_grammar_prob* a uloží sa do premennej `prob_before_mowing`, ktorá bude v tomto prípade 0,643338. Bs nech je rovné 0,05. Cyklus v algoritme 2 sa bude opakovať

#	Štruktúra	Pravdepodobnosť	#	Štruktúra	Pravdepodobnosť
1.	L6D1	0,07880148	11.	L5D3	0,01587387
2.	L7D1	0,06530734	12.	L6	0,01544120
3.	L6D2	0,06281944	13.	L6D3	0,01538711
4.	L8D1	0,05492306	14.	L4D3	0,01479218
5.	L5D2	0,04664810	15.	L6S1	0,01457584
6.	L5D1	0,04318667	16.	L4D4	0,01284513
7.	L9D1	0,04221315	17.	L7	0,01276400
8.	L7D2	0,04040131	18.	L8	0,01222315
9.	L4D2	0,03783228	19.	L3D3	0,01157414
10.	L8D2	0,03461424	20.	L7S1	0,01111442

Tabuľka 5.1: Príklad základných štruktúr.

dovtedy, kým sa celková pravdepodobnosť nezníži aspoň o 0,05. Pre rýchle a jednoduché odstránenie najmenej pravdepodobného pravidla sa využíva funkcia `pop(grammar_len - 1)`, ktorá odoberie posledný prvok zo štruktúry
`ruleset[Grammar][Grammar.txt]`. Premenná `grammar_len` označuje aktuálny počet základných štruktúr. Po odobratí posledného pravidla č. 20 sa znovu vypočíta celková pravdepodobnosť (0.632224) a rozdiel oproti `prob_before_mowing`, ktorý po prvej iterácii bude rovný 0,011114.

Keďže rozdiel pravdepodobností musí byť aspoň 0,05 tak tento algoritmus odstráni dokopy 5 posledných pravidiel. V gramatike ostane teda 15 základných štruktúr, z ktorých sa podľa diagramu 5.1 následne vypočíta celkový počet hesiel.

Filtrovanie pravidiel kapitalizácie

Každé pravidlo kapitalizácie **C** o dĺžke **n** lineárne zvyšuje počet hesiel, ktorých základná štruktúra obsahuje $\mathbf{L_n}$. To znamená, že ak L_8 obsahuje 3 reťazce (password, sunshine, iloveyou) a C_8 obsahuje 3 pravidlá (N_8, U_1N_7, U_8) tak nástroj PCFG Cracker vygeneruje zo základnej štruktúry L_8 dokopy 9 hesiel:

- password, sunshine, iloveyou,
- Password, Sunshine, Iloveyou,
- PASSWORD, SUNSHINE, ILOVEYOU.

Pravidlá kapitalizácie sú špecifické tým, že prvé 3 majú niekoľkonásobne väčšiu pravdepodobnosť ako zvyšok. Tabuľka 3.5 zobrazuje ako príklad pravidlá kapitalizácie o dĺžke 6. Je preto veľmi efektívne odstrániť najmenej pravdepodobné pravidlá kapitalizácie a znížiť tým celkový počet hesiel, čo dokazujú aj experimenty v kapitole 6.

Algoritmus 3 zobrazuje odstraňovanie pravidiel kapitalizácie pomocou kroku **cs** (capitalization step). Tento krok predstavuje prah pri rozhodovaní, či sa pravidlo odstráni alebo nie. Prah sa zvýši každou iteráciou tohto algoritmu, a to tak, že sa krok **cs** vynásobí poradím aktuálnej iterácie. To znamená, že sa postupne budú odstraňovať pravidlá s čoraz väčšou pravdepodobnosťou.

Algoritmus postupne prechádza cez všetky pravidlá zo všetkých súborov kapitalizácie, ktoré gramatika obsahuje. Premenná `tuple` označuje pravidlo v tvare *(maska, pravdepodobnosť)*. Pravdepodobnosť každého pravidla sa porovná s aktuálnou hodnotou cs a v prípade,

Algoritmus 3 Filtrovanie pravidiel kapitalizácie.

Vstup: *cs*

```
1: removed ← false
2: for all file ∈ ruleset[Capitalization] do
3:    for all tuple ∈ ruleset[Capitalization][file] do
4:       if tuple[1] < cs then
5:          ruleset[Capitalization][file].remove(tuple)
6:          removed ← true
7:       end if
8:    end for
9: end for
10: if removed then
11:    rebuild_size(Capitalization)
12: end if
```

že je menšia, tak sa pravidlo odstráni. V prípade, že sa odstráni aspoň jedno pravidlo, označí sa premenná `removed` hodnotou *true* a na záver algoritmu sa musia znova prepočítať veľkosti súborov s pravidlami kapitalizácie funkciou `rebuild_size`.

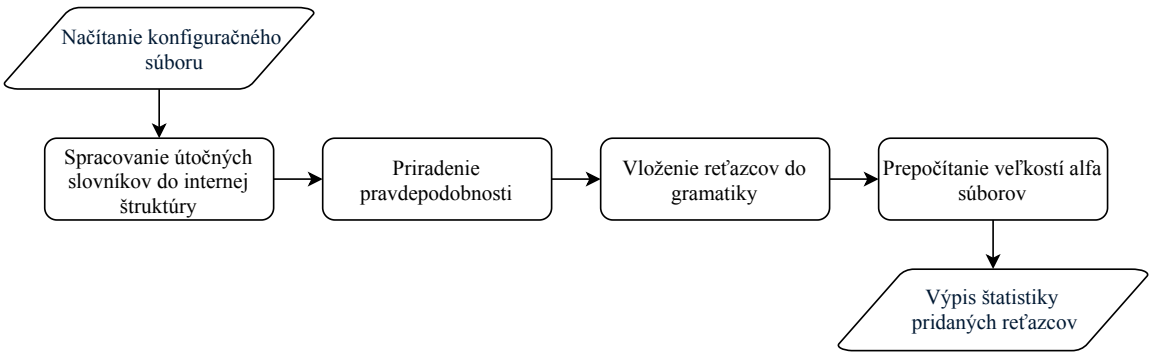
5.5 Útočné slovníky

V slovníkových útokoch sa aplikujú obmieňacie pravidlá na zoznam slov zvaný útočný slovník. Preto, aby útok dokázal prelomiť heslo, je potrebné nielen použiť správne pravidlo, ale aj zahrnúť správne slová do útočných slovníkov. Nástroj PCFG Cracker odvodzuje obmieňacie pravidlá z tréningovej sady reálnych hesiel užívateľov a ukázali sa ako veľmi účinné. V predchádzajúcej časti bolo tiež ukázané, že je možné vyfiltrovať niektoré pravidlá z gramatiky a zredukovať tým výsledný počet vygenerovaných hesiel. Na vylepšenie úspešnosti vytvorených hesiel slúži práve začlenenie viacerých útočných slovníkov do gramatiky.

Útočné slovníky môžu byť niekedy vnímané ako samotné odhady hesiel. Napríklad Bonneau vytvoril slovníky rozličných skupín používateľov Yahoo založené na jazykovom pozadí a definoval zoznam najlepších 1000 skutočných hesiel pre každú skupinu. Autor potom určuje účinnosť takýchto slovníkov oproti iným jazykovým skupinám [3].

Použitie slovníkov so skutočnými heslami je pomerne odlišné od slovníkov, ktoré by mali obsahovať vhodné alfa reťazce do základných štruktúr. V nástroji PCFG Cracker nie je zdvojnásobenie veľkosti útočného slovníka veľký problém z hľadiska pomeru veľkosti slovníka na úspešnosť lámania hesiel. Heslá sa stále budú generovať podľa pravdepodobnosti, hoci pravdepodobnosť výsledných hesiel sa môže zmeniť a tým pádom sa vygenerujú v inom poradí. Okrem toho sa vytvoria nové kombinácie hesiel, pretože pribudnú alfa reťazce, no príliš veľa slov pre rovnakú základnú štruktúru môže zredukovať pravdepodobnosť každého terminálneho hesla a tým pádom sa vygenerujú omnoho neskôr.

V iných prácach sú útočné slovníky použité ako samotné heslá aj ako zdroj pre generovanie variácií týchto hesiel použitím obmieňacích pravidiel [19]. Dell'Amico a spol. [5] vyhodnotili niekoľko slovníkov dostupných v nástroji John the Ripper tým spôsobom, že najprv porovnali počet prelomených hesiel len pomocou samotných slov. Ich výsledok ukazuje, že je výhodnejšie použiť rovnaký typ útočných slovníkov ako typ cieľa (napríklad fínsky slovník pri lámaní fínskych hesiel).



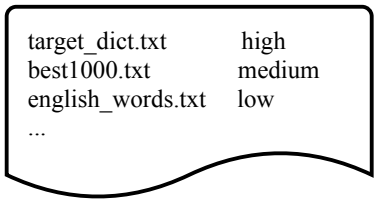
Obr. 5.2: Schéma implementácie útočných slovníkov.

Súčasná implementácia nástroja PCFG Cracker nemá možnosť použiť dodatočné súbory slov ako zdroj alfa reťazcov. Nástroj pri generovaní hesiel použije len tie reťazce, ktoré sa vyskytujú v trénovacom súbore. Tento prístup značne obmedzuje kombinácie hesiel a neumožňuje vykonávať cielené útoky užívateľom vytvoreným slovníkom. Program PCFG Mower poskytuje možnosť začleniť útočné slovníky do gramatiky. Používateľ môže ovplyvniť pravdepodobnosť týchto slov v gramatike tým, že pri každému útočnému slovníku definuje prioritu (vysoká, stredná, nízka).

Obrázok 5.2 zobrazuje schému implementácie útočných slovníkov do programu PCFG Mower. Jednotlivé časti tejto schémy popisujú nasledujúce podsekcie.

Konfiguračný súbor

Útočné slovníky je voliteľná funkcionalita, ktorá sa aplikuje v prípade, že používateľ spustí program s parametrom „-a“. Za týmto parametrom musí nasledovať cesta ku konfiguračnému súboru útočných slovníkov, ktorý na každom riadku obsahuje cestu k slovníku a jeho prioritu oddelenú tabulátorom. Príklad konfiguračného súboru zobrazuje obrázok 5.3.



Obr. 5.3: Príklad konfiguračného súboru útočných slovníkov.

Spracovanie útočných slovníkov

Po načítaní konfiguračného súboru sa útočné slovníky spracujú a uložia do internej štruktúry veľmi podobnej gramatike:

```
dictionaries[file][length][password] = 0
```

kde *dictionaries* je názov štruktúry, *file* je názov útočného slovníka, *length* je dĺžka slova a *password* je samotné slovo. Každý slovník má teda slová reťazce rozdelené podľa dĺžky a každému reťazcu je inicializovaná pravdepodobnosť hodnotou 0. Priority jednotlivých slovníkov sú uložené v štruktúre *priorities* následovne:

```
priorities[file] = "high | medium | low"
```

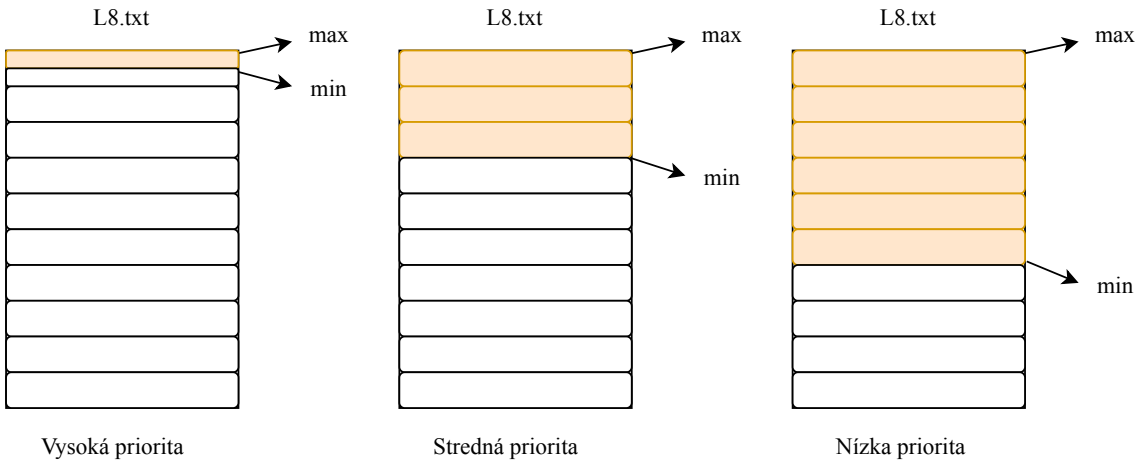
Priradenie pravdepodobnosti

Spracovaným reťazcom z útočných slovníkov sa v tejto časti priradí pravdepodobnosť. Tento proces podrobne opisuje algoritmus 4. Vstupné dáta predstavujú štruktúry `ruleset`, `dictionaries` a `priorities`, ktoré už predstavené. Algoritmus postupne prechádza cez všetky slovníky (`file`) a dĺžky reťazcov (`length`) v štruktúre `dictionaries`.

Vždy po príchode novej dĺžky (riadok 3), algoritmus zistí maximálnu a minimálnu hranicu pravdepodobnosti, ktorú môže reťazcu priradiť a uloží ich do premenných `max_prob` a `min_prob`. Výpočet týchto hraníc prebieha vo funkcii `analyse_alpha_file_prob`, do ktorej vstupuje priorita a dĺžka reťazca. Táto funkcia analyzuje súbor s alfa reťazcami o danej dĺžke. Princíp výpočtu hraníc je zobrazený na obrázku 5.4. Pre každú prioritu je maximálna hranica zvolená ako pravdepodobnosť prvého reťazca v súbore. Minimálna hranica je daná pravdepodobnosťou reťazcu, ktorý sa vyskytuje na určitom percente z pohľadu celkového počtu reťazcov. Táto hodnota je odlišná pre každú prioritu:

- **vysoká priorita**: 5% všetkých reťazcov,
- **stredná priorita**: 30% všetkých reťazcov,
- **nízka priorita**: 60% všetkých reťazcov.

To znamená, že slovám v útočnom slovníku s vysokou prioritou sa priradí taká pravdepodobnosť, aby sa zaradili medzi prvých 5% reťazcov v danom súbore a tým pádom sa tieto reťazce použijú pri generovaní hesiel ako prvé.



Obr. 5.4: Princíp prideľovania pravdepodobnosti podľa priority.

Po získaní maximálnej a minimálnej pravdepodobnosti algoritmus iteruje cez všetky reťazce danej dĺžky a generuje im pravdepodobnosť funkciou `random.uniform(min_prob,`

`max_prob)`, ktorá vygeneruje náhodné číslo v intervale $< min\_prob; max\_prob >$ do premennej `generated_prob`. Vygenerovaná pravdepodobnosť sa následne zaokrúhli na 8 desatinných miest a priradí sa k danému reťazcu do štruktúry `dictionaries`.

Algoritmus 4 Priradenie pravdepodobnosti reťazcom z útočných slovníkov.

Vstup: $ruleset, dictionaries, priorities$

```
1: for all file ∈ dictionaries do
2:   for all length ∈ dictionaries[file] do
3:     max_prob, min_prob ← analyse_alpha_file_prob(priorities[file], length)
4:     for all word ∈ dictionaries[file][length] do
5:       generated_prob ← random.uniform(max_prob, min_prob)
6:       generated_prob ← round(generated_prob, 8)
7:       dictionaries[file][length][word] ← generated_prob
8:     end for
9:   end for
10: end for
```

Druhý prístup prideľovania pravdepodobnosti by mohol byť taký, že každému reťazcu z rovnakého útočného slovníka bude priradená rovnaká pravdepodobnosť. V tomto prípade by generátor dosadzoval do základnej štruktúry všetky reťazce z útočného slovníka naraz, pretože majú rovnakú pravdepodobnosť. Z jednej základnej štruktúry by sa tým pádom mohlo vygenerovať obrovské množstvo hesiel naraz, no zablokovalo by to variácie z inej základnej štruktúry.

Vloženie reťazcov do gramatiky

Aby sa zachovalo generovanie hesiel podľa pravdepodobnosti, musia byť reťazce v jednotlivých alfa súboroch zoradené od najväčšej pravdepodobnosti. Každý reťazec v štruktúre `dictionaries` má inú pravdepodobnosť. To znamená, že každému reťazcu je nutné nájsť správnu pozíciu v gramatike rýchlo a efektívne. Tento proces opisuje algoritmus 5.

Algoritmus rieši problém rýchleho vyhľadávania v štruktúre `ruleset`, vloženie nového reťazca spolu s jeho pravdepodobnosťou, duplicitné reťazce a zoradenie pravidiel. Pre pripomenutie, štruktúra `ruleset[Alpha][length]` obsahuje pole zoradených dvojíc *(reťazec, pravdepodobnosť)*. Binárne vyhľadávanie v takomto poli by malo zložitosť O(log n) a vykonávalo by sa pre každý reťazec z útočných slovníkov, čo by viedlo k obrovskému zvýšeniu času programu PCFG Mower. Algoritmus poskytuje efektívnejší prístup, a to previesť spomínané pole na asociatívne pole (v jazyku *python* je to *dictionary*) do premennej `tmp_dict`:

```
ruleset[Alpha][length] = (word, prob)  =>  tmp_dict[word] = prob
```

Vyhľadávanie v asociatívnom poli má konštantnú zložitosť O(1). Oproti klasickému poľu má jednu nevýhodu – položky poľa už nie sú zoradené.

Algoritmus prechádza postupne cez všetky reťazce danej dĺžky a ich vygenerované pravdepodobnosti. Na riadku 5 sa vyhľadá reťazec `word` medzi kľúčmi vo vytvorenom asociatívnom poli `tmp_dict`. To znamená, že algoritmus zisťuje, či reťazec z útočného slovníka už existuje v gramatike. Ak áno, porovná sa jeho pravdepodobnosť v gramatike s pravdepodobnosťou, ktorá mu bola vygenerovaná v predošlej časti. V prípade, že jeho pravdepodobnosť v gramatike je menšia, tak sa aktualizuje. Ak sa reťazec v gramatike nenachádza, vytvorí sa v asociatívnom poli nový kľúč s danou pravdepodobnosťou.

Algoritmus 5 Pridanie reťazcov z útočných slovníkov do gramatiky.

Vstup: $ruleset, dictionaries$

```
1: for all file ∈ dictionaries do
2:   for all length ∈ dictionaries[file] do
3:     tmp_dict ← dict(ruleset[Alpha][length])
4:     for all word, prob ∈ dictionaries[file][length] do
5:       if word ∈ tmp_dict then
6:         if tmp_dict[word] < prob then
7:           tmp_dict[word] ← prob
8:         end if
9:       else
10:        tmp_dict[word] ← prob
11:      end if
12:    end for
13:    sorted_list ← sorted(tmp_dict)
14:    ruleset[Alpha][length] ← sorted_list
15:  end for
16: end for
```

Po spracovaní všetkých reťazcov danej dĺžky sa asociatívne pole musí znova zoradiť a previesť na pole dvojíc. Takéto zoradené pole označuje premenná `sorted_list`. Zoradené pole s novými reťazcami sa vloží do štruktúry s gramatikou.

Kapitola 6

Experimenty

Táto kapitola opisuje testovanie implementovaných vylepšení pre nástroj PCFG Cracker. Hlavným cieľom experimentov je dokázať, že vytvorený program PCFG Mower má praktický prínos pri lámaní hesiel pomocou pravdepodobnostných bezkontextových gramatík. Kapitola obsahuje niekoľko typov experimentov s využitím rozličnej konfigurácie programu PCFG Mower. Sú použité voľne dostupné slovníky uniknutých hesiel reálnych užívateľov. Zredukované gramatiky a ich vygenerované slovníky sa porovnávajú medzi sebou aj s originálnou verziou od Weira a spol. [21].

V sekcii 6.1 sú opísané nástroje, slovníky a hardvér, ktoré boli použité na testovanie. V sekcii 6.2 je predstavený skript na automatizáciu celého procesu testovania. Sekcia 6.3 zobrazuje štatistiky a úspešnosť gramatík pri použití filtrovania pravidiel. Prínos útočných slovníkov opisuje sekcia 6.4.

6.1 Použité nástroje

Testovací hardvér

Všetky experimenty prebiehali na serveri s nasledujúcimi parametrami:

OS: Ubuntu 18.04.2 LTS

Procesor: Intel(R) Core(TM) i5-4690K Quad-Core 3,5 GHz

RAM: 32 GB

Disk: Samsung SSD 860 EVO 1TB

Grafická karta: GeForce GTX 1080 Ti

Go Manager

V rámci spolupráce v rovnakej oblasti reimplementoval D. Mikuš generátor hesiel v nástroji PCFG Cracker (PCFG Manager) do jazyka *Go* [12]. Táto nová verzia obsahuje rôzne optimalizácie, rieši efektivitu stromovej štruktúry pri generovaní uzlov a odstraňuje veľký počet nedostatkov z originálnej verzie. Nový Go PCFG Manager poskytuje vyšší level paralelizácie pomocou špeciálnych *Go* vláken (angl. goroutines). Ďalšia dôležitá optimalizácia sa prejavuje pri výpise hesiel tým, že nevypisuje heslá po jednom. Jednotlivé terminálne štruktúry z jedného pre-terminálu si drží v pamäti a vypíše ich až po spracovaní konkrétneho pre-terminálu. Taktiež ponúka dva nové parametre:

- **-g**: počet vláken použitých pri vytváraní terminálnych štruktúr,
- **-m**: maximálny počet hesiel, po ktorom sa program ukončí.

Tabuľka 6.1 porovnáva pôvodnú verziu programu PCFG Manager s novou verziou napísanú v jazyku *Go*. V tabuľke sú zobrazené experimentálne výsledky generovania hesiel s použitím dvoch trénovacích slovníkov: *Darkweb2017-10000* a *Rockyou-75*. Taktiež je zobrazený rozdiel medzi rozdielnymi typmi diskov (HDD, SSD). Všetky merania boli prebiehali presne 3 minúty.

Trénovací slovník	PCFG Manager	HDD	SSD
Darkweb	Python	3 022 923	2 948 532
	Go	24 592 908	24 609 579
	Zrýchlenie	8,1x	8,3x
Rockyou-75	Python	184 18 684	20 843 491
	Go	490 635 443	842 695 475
	Zrýchlenie	26,6x	40,4x

Tabuľka 6.1: Počet vygenerovaných hesiel a zrýchlenie generátora hesiel PCFG Manager.

Nová verzia programu PCFG Manager je 8 až 40 krát rýchlejšia oproti originálnej verzii. Pri použití gramatiky vytvorenú zo slovníka *Darkweb* je zrýchlenie najnižšie, pretože obsahuje dlhé a zložité základné štruktúry. Zo vstupného slovníka *Rockyou-75* vygeneroval Go PCFG Manager omnoho viac hesiel a rozdiel medzi HDD a SSD je výraznejší. Pre účely experimentov bol použitý práve táto nová verzia PCFG Manager.

Trénovacie a referenčné slovníky hesiel

Ako zdroj dát pre experimenty slúži repozitár SecLists[1], ktorý obsahuje niekoľko uniknutých zoznamov hesiel a rebríčky najpoužívanejších hesiel v konkrétnych rokoch. Všetky použité slovníky sú zobrazené v tabuľke 6.2. Pre jednoduchší zápis je každému slovníku pridelený identifikátor (ID). Tabuľka ďalej zobrazuje počet hesiel a veľkosť jednotlivých slovníkov.

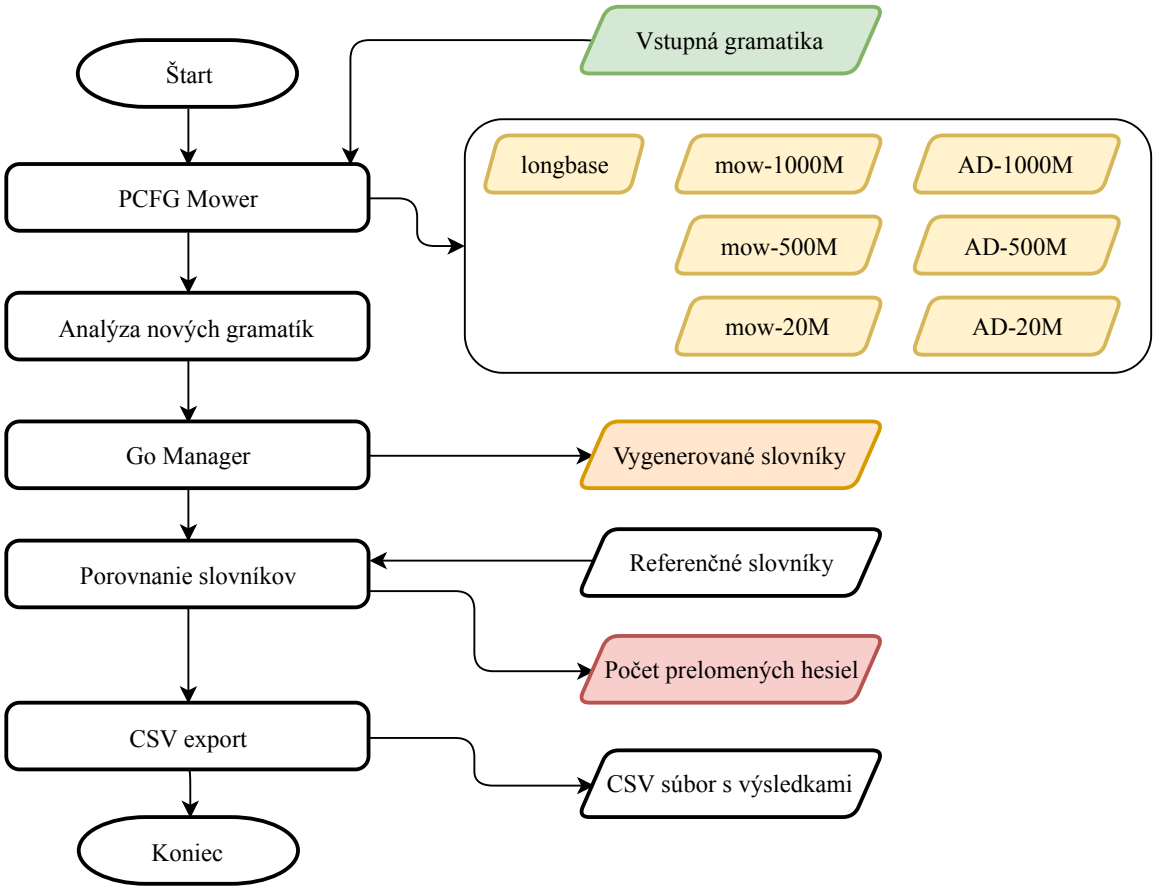
ID	Názov	Počet hesiel	Veľkosť
dw	Darkweb2017-10000.txt	10 000	82 KB
r65	rockyou-65.txt	30 290	344 KB
ms	myspace.txt	37 126	354 KB
tl	tuscl.txt	38 820	324 KB
pr	probab-v2-top12000.txt	12 645	100 KB
ls	Lizard-Squad.txt	11 782	121 KB
hm	hotmail.txt	8 930	86 KB
b1	best110.txt	110	1 KB
b2	best1050.txt	1 050	8,9 KB
10m	10-million-password-list-top-10000.txt	10 000	74,7 KB

Tabuľka 6.2: Slovníky hesiel použité na experimenty.

[1] https://github.com/danielmiessler/SecLists

6.2 Automatizácia testov

Manuálny proces testovania je veľmi časovo náročný. Kvôli tomuto problému vznikol skript *test_runner.py*, ktorý vykonáva všetko od vytvorenia nových zredukovaných gramatík až po porovnanie vygenerovaných slovníkov. Obrázok 6.1 znázorňuje princíp automatických experimentov.



Obr. 6.1: Schéma automatického testovania.

Program prijíma na vstup natrénovanú gramatiku z nástroja PCFG Trainer [21]. Túto vstupnú gramatiku použije na vytvorenie piatich upravených gramatík pomocou programu PCFG Mower:

longbase – V gramatike sú odstránené dlhé základné štruktúry, t.j. tie, ktoré obsahujú viac ako 5 neterminálov.

mow-1000M – Gramatika vytvorená filtrovaním pravidiel s limitom 1000 miliónov hesiel a predvolené hodnoty bs a cs (0,001).

mow-500M – Gramatika vytvorená filtrovaním pravidiel s limitom 500 miliónov hesiel a predvolené hodnoty bs a cs (0,001).

mow-20M – Gramatika vytvorená filtrovaním pravidiel s limitom 20 miliónov hesiel a predvolené hodnoty bs a cs (0,001).

AD-1000M – Pri vytváraní tejto gramatiky boli použité útočné slovníky best110.txt s prioritou high a best1050.txt s prioritou medium. Limit počtu hesiel je 1000 miliónov.

AD-500M – Pri vytváraní tejto gramatiky boli použité útočné slovníky best110.txt s prioritou high a best1050.txt s prioritou medium. Limit počtu hesiel je 500 miliónov.

AD-20M – Pri vytváraní tejto gramatiky boli použité útočné slovníky best110.txt s prioritou high a best1050.txt s prioritou medium. Limit počtu hesiel je 20 miliónov.

Každú vytvorenú gramatiku skript analyzuje a získa štatistiky dôležité pre experimenty, ktoré sú detailne opísané v sekcii 6.3. Následne sa nad všetkými gramatikami spustí nový generátor hesiel Go Manager s limitom 10 minút. Výsledok tohto kroku sú vygenerované slovníky hesiel. Vygenerované slovníky sa porovnajú s vybranými referenčnými slovníkmi a uloží sa počet uhádnutých hesiel. Všetky štatistiky a výsledky nazbierané počas behu skriptu sa exportujú do csv súboru, ktorý je možný zobraziť v ktoromkoľvek tabuľkovom editore. Výsledky tohto skriptu je možné vidieť v nasledujúcich sekciách.

6.3 PCFG Mower – filtrovanie pravidiel

Prvá časť experimentov sa zameriava na štatistiky a úspešnosť modifikovaných gramatík. Tabuľka 6.3 zobrazuje výsledky trénovania, modifikácie gramatík, vygenerované slovníky hesiel a úspešnosť pri lámaní hesiel rozličných referenčných slovníkov. Pre každé meranie je nastavený limit 10 minút, pretože generovanie hesiel z pôvodnej gramatiky by trvalo hodiny až dni. Počas tohto limitu sa vytvorí dostatočný počet hesiel pre experimenty. Ak Go PCFG Manager presiahol dĺžku 10 minút, generovanie bolo zastavené a výsledky sa ďalej analyzovali.

Prvý stĺpec (tr) zobrazuje slovníky, ktoré boli použité na trénovanie originálnych gramatík. Pre každý trénovací slovník je zobrazených 5 gramatík. Prvý riadok reprezentuje originálnu gramatiku bez modifikácií. V gramatike *longbase* boli odstránené dlhé základné štruktúry (dlhšie ako 5 neterminálov). Ostatné 3 gramatiky boli vytvorené programom PCFG Mower z gramatiky *longbase*. Názov *mow-n* znamená, že že PCFG Mower bol spustený s limitom *n*. Experimenty prebehli s tromi rôznymi limitmi: 1 000 000 000 (1000M), 500 000 000 (500M) a 20 000 000 (20M) hesiel. Vo všetkých prípadoch boli premenné *bs* a *cs* nastavené na rovnakú hodnotu 0,001 pre dosiahnutie rovnakého filtrovania pre každú gramatiku. Zmeny v gramatike po každom behu programu PCFG Mower sú zobrazené na jednotlivých riadkoch. Tabuľka zobrazuje počet zachovaných základných štruktúr (ZŠ) a počet pravidiel kapitalizácie (Kap) pre každú originálnu aj modifikovanú gramatiku.

Ďalšia časť stĺpcov informuje o metrikách vygenerovaného slovníka. Tabuľka zobrazuje čas, ktorý trval programu Go PCFG Manager na generovanie hesiel. V prípade, že generátor dosiahol časový limit 10 minút tak je tento stĺpec označený ako *10m**. Ďalšie zobrazené metriky vygenerovaných slovníkov sú veľkosť súboru a počet hesiel v tomto súbore v miliónoch, označené ako MOP (angl. milions of passwords).

Zvyšok tabuľky zobrazuje úspešnosť pri hádaní hesiel na referenčných slovníkoch. Úspešnosť vyjadruje počet percent hesiel z referenčného slovníka, ktoré sa zhodujú s heslami vo vygenerovanom slovníku. Posledný stĺpec zobrazuje ASRI (dopad na priemernú úspešnosť). Táto metrika je špeciálne vytvorená ako jednotný ukazovateľ zmeny úspešnosti pre danú modifikáciu. ASRI je vypočítaný následovne:

$$ASRI = \frac{\sum_{i=1}^{n}(SR_i^{mod} - SR_i^{orig})}{n},$$

kde SR_i^{orig} je úspešnosť na referenčnom slovníku i s originálnou gramatikou, SR_i^{mod} je úspešnosť na referenčnom slovníku i s modifikovanou gramatikou a n je počet referenčných slovníkov. V tomto prípade, $n = 4$. Metrika ASRI pomáha analyzovať dopad modifikovaných gramatík. Kladná hodnota ASRI značí, že má gramatika zvýšenú úspešnosť, zatiaľ čo záporná hodnota značí pokles úspešnosti.

	Gramatika			Vygenerovaný slovník			Úspešnosť				
tr	Názov	ZŠ	Kap	Čas	Veľkosť	MOP	pr	ms	dw	r65	ASRI
dw	original	323	83	10m*	731 MB	78	45,03 %	26,83 %	98,27 %	41,39 %	
	longbase	288	83	10m*	12 GB	1,110	45,01 %	26,91 %	98,35 %	41,40 %	+0,04 %
	mow-1000M	106	40	25s	3,3 GB	373	44,54 %	24,47 %	96,42 %	38,36 %	-1,93 %
	mow-500M	106	40	25s	3,3 GB	373	44,54 %	24,47 %	96,42 %	38,36 %	-1,93 %
	mow-20M	86	32	2s	77 MB	9	44,18 %	24,12 %	95,65 %	38,00 %	-2,39 %
r65	original	256	39	10m*	1,5 GB	151	72,34 %	37,63 %	88,25 %	99,84 %	
	longbase	223	39	9m 54s	25 GB	2,210	72,30 %	37,63 %	88,14 %	99,81 %	-0,05 %
	mow-1000M	161	36	3m 31s	11 GB	980	72,17 %	37,17 %	87,73 %	99,61 %	-0,35 %
	mow-500M	123	31	1m 31s	4,5 GB	409	72,01 %	36,62 %	87,23 %	99,35 %	-0,71 %
	mow-20M	79	20	3,5s	130 MB	13,8	70,98 %	34,26 %	85,80 %	97,16 %	-2,47 %
ms	original	1574	179	10m*	5,7 GB	616	47,47 %	93,68 %	69,14 %	46,42 %	
	longbase	1430	179	10m*	9,5 GB	1,030	47,45 %	94,38 %	69,07 %	46,42 %	+0,11 %
	mow-1000M	110	25	3m	9,2 GB	941	46,37 %	82,40 %	66,74 %	43,04 %	-4,42 %
	mow-500M	78	24	1m	3,1 GB	334	45,13 %	79,67 %	64,71 %	42,62 %	-6,03 %
	mow-20M	21	23	2s	126 MB	15	33,25 %	61,17 %	54,28 %	35,58 %	-18,11 %
tl	original	1290	242	10m*	4,5 GB	520	55,27 %	36,87 %	69,85 %	43,86 %	
	longbase	1158	242	10m*	7,6 GB	870	55,23 %	37,15 %	69,79 %	43,87 %	+0,05 %
	mow-1000M	91	20	2m 43s	7,5 GB	884	54,06 %	30,94 %	66,08 %	40,37 %	-3,60 %
	mow-500M	48	19	1m 8s	1,8 GB	200	53,77 %	29,05 %	64,19 %	39,39 %	-4,86 %
	mow-20M	24	18	2s	133 MB	17	52,08 %	22,27 %	55,61 %	35,64 %	-10,07 %

Tabuľka 6.3: Úspešnosť originálnej a modifikovaných gramatík (* - bol dosiahnutý časový limit)

Výsledky v tabuľke dokazujú, že odstránenie dlhých základných štruktúr vedie k obrovskému zrýchleniu generovania hesiel a tým pádom dokáže gramatika *longbase* vygenerovať niekoľkonásobne viac hesiel v intervale 10 minút. Najväčšie zrýchlenie bolo dosiahnuté na slovníkoch *dw* a *r65*, pretože obsahujú veľmi zložité heslá, ktoré vytvárajú zložité základné štruktúry. Po modifikácii vygenerovala *longbase* gramatika až 14 krát viac hesiel za rovnaký čas ako pôvodná gramatika. Trénovanie na slovníkoch *ms* a *tl* vytvorí jednoduché gramatiky, takže zrýchlenie nie je až také výrazné. Odstránenie dlhých základných štruktúr nemá takmer žiadny vplyv na úspešnosť, čo potvrdilo predpoklad, že ich výskyt v gramatike je zanedbateľný. Zo 16 meraní, iba 8 viedlo k zníženiu úspešnosti a to maximálne o 0,06 %. ASRI je väčšinou kladné, pretože v 6 prípadoch odstránenie dlhých základných štruktúr zlepšilo úspešnosť o 0,7 % vďaka väčšiemu počtu vygenerovaných hesiel.

Ďalšie modifikácie pôvodnej gramatiky analyzujú či filtrovanie najmenej pravdepodobných pravidiel prináša nejaké výhody. Vo všetkých prípadoch modifikácií *mow-n* sa podarilo generátoru Go PCFG Manager spracovať celú gramatiku za menej ako 4 minúty. Cieľom týchto modifikácií je zredukovať čas a veľkosť vygenerovaných slovníkov. Najlepšie výsledky dosiahli znova slovníky *dw* a *r65*, pri ktorých sa podarilo zredukovať veľkosť z 12 GB (*longbase*) na 112 MB a z 25 GB na 130 MB s poklesom úspešnosti pod 4 % vo všetkých prípadoch. Modifikácie *mow-1000M* a *mow-500M* pre slovník *dw* su rovnaké, pretože gramatika obsahuje mnoho pravidiel s rovnakou pravdepodobnosťou, ktoré po odstránení rapídne zníži celkový počet počet hesiel. Tento problém je možné vyriešiť nastavením iných krokov bs a cs, no pre zachovanie rovnakých podmienok pre všetky merania boli použité predvo-

lené hodnoty. Pre slovníky *ms* a *tl* sa taktiež podarilo znížiť čas a veľkosť slovníkov, no modifikácie *mow-20M* sú príliš striktné pre uspokojujúce výsledky.

Analýza gramatiky rockyou-65

Modifikácie gramatiky natrénovanej na slovníku *rockyou-65.txt* dosahujú veľmi dobré výsledky, ktoré budú podrobnejšie analyzované v tejto sekcii. Tabuľka 6.4 zobrazuje niektoré vlastnosti modifikovaných gramatík dôležité pre experimenty, medzi ktoré patrí počet a celková pravdepodobnosť základných štruktúr (ZŠ), počet a celková pravdepodobnosť pravidiel kapitalizácie (PK) a celkový počet hesiel, ktoré dokáže gramatika vygenerovať. Počet základných štruktúr udáva rôznorodosť hesiel v trénovacom slovníku a ich celková pravdepodobnosť je v originálnej gramatike vždy 100 %. Celková pravdepodobnosť pravidiel kapitalizácie sa líši podľa počtu jednotlivých dĺžok týchto pravidiel. Gramatika rockyou-65 má presne 17 súborov s pravidlami kapitalizácie – pre každú dĺžku alfa reťazcov musí existovať kapitalizačné pravidlo s rovnakou dĺžkou, t.j. pre L_1 musí existovať C_1 atď. 17 súborov s pravidlami kapitalizácie udáva celkovú pravdepodobnosť 1700 %. Zaujímavosť v tejto tabuľke je celkový počet hesiel 68 469 * 10^{25} pri použití originálnej gramatiky. Takéto množstvo hesiel by sa generovalo niekoľko rokov.

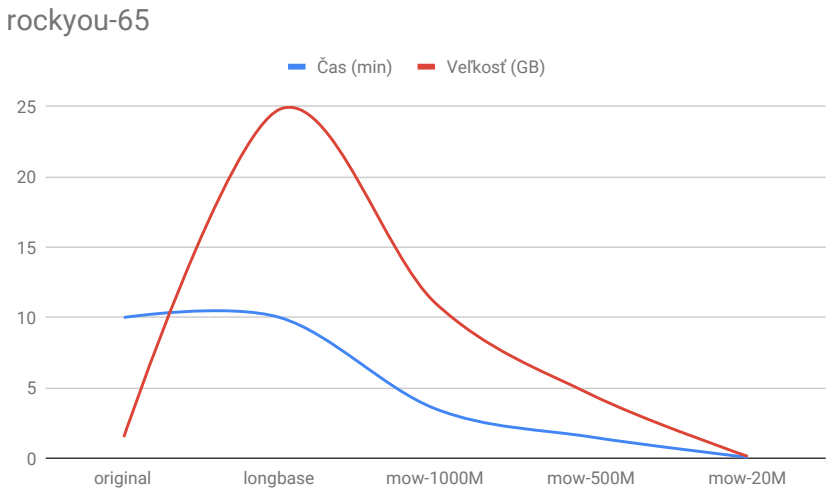
Modifikácia	Počet ZŠ	Pravd. ZŠ	Počet PK	Pravd. PK	Počet hesiel
original	256	100 %	39	1700 %	$68\,469 * 10^{25}$
longbase	223	99,8 %	39	1700 %	2 209 975 040
mow-1000M	161	99,6 %	36	1699 %	980 098 035
mow-500M	123	99,4 %	31	1697 %	409 683 766
mow-20M	79	98,8 %	20	1681 %	13 856 771

Tabuľka 6.4: Vlastnosti modifikovaných gramatík rockyou-65.

Pre jednotlivé modifikácie sú najdôležitejšie metriky vygenerovaných slovníkov: čas, veľkosť a úspešnosť. Grafy týchto metrík sú zobrazené na nasledujúcich obrázkoch. Na obrázku 6.2 je možné vidieť čas a veľkosť vygenerovaných slovníkov pre jednotlivé modifikácie. Aj v tomto prípade platí limit 10 minút pre generovanie hesiel. Obrázok 6.4 zobrazuje počet prelomených hesiel voči referenčným slovníkom.

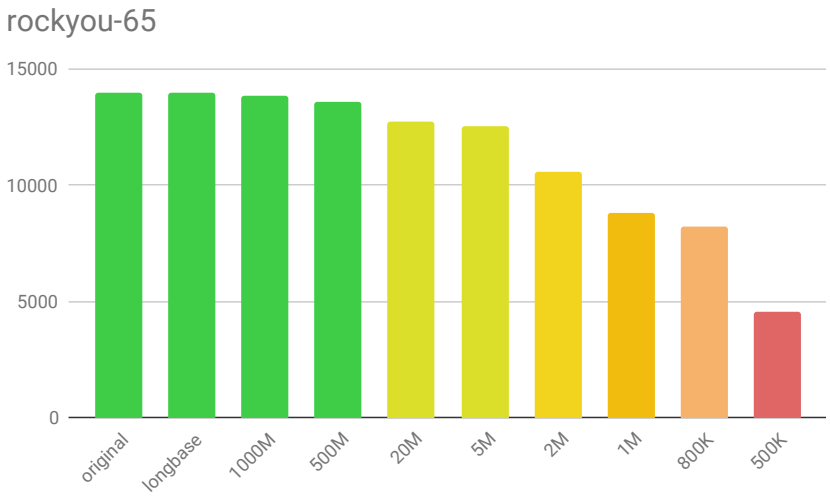
Modifikácia *longbase* sa dá označiť za prelomovú v rámci všetkých experimentov. Prináša veľmi veľkú praktickú hodnotu pri pravdepodobnostnom generovaní hesiel. Odstránením 33 dlhých základných štruktúr z celkových 256 sa urýchlilo generovanie hesiel až 14-násobne. Celková pravdepodobnosť základných štruktúr klesla len o 0,2 %, čo znamená, že to boli zanedbateľné štruktúry. Podľa tabuľky 6.4 sa týmto jednoduchým úkonom zredukoval celkový počet hesiel z tejto gramatiky na 2,2 miliardy, čo je obrovská zmena oproti originálnej gramatike. Všetky variácie hesiel z modifikovanej gramatiky *longbase* sa dokonca stihli vytvoriť do limitu 10 minút, konkrétne 9 minút a 54 sekúnd (viď tabuľku 6.3). Vyššia rýchlosť generovania hesiel znamená automaticky väčšiu veľkosť výstupného slovníka, ktorá dosiahla približne 25 GB. Táto hodnota sa dá považovať za zbytočne vysokú, pretože úspešnosť sa oproti originálnej gramatike takmer vôbec nezmenila.

Z analýzy *longbase* modifikácie vyplýva, že každá pravdepodobnostná gramatika má určitú hranicu počtu hesiel, po ktorej sa už neoplatí ďalej generovať heslá. Hľadanie takejto hranice znázorňuje obrázok 6.3, ktorý obsahuje počet uhádnutých hesiel v referenčnom slovníku myspace. Jednotlivé stĺpce v tomto grafe reprezentujú modifikácie gramatiky rockyou-65, t.j. každá modifikácia má nastavený iný limit počtu hesiel a tým pádom má inú veľkosť



Obr. 6.2: Čas a veľkosť vygenerovaných slovníkov modifikovaných gramatík rockyou-65.

vygenerovaného slovníka. V grafe je možné vidieť, že po dosiahnutí 500 miliónov hesiel sa už ďalej neoplatí pokračovať, pretože pomer veľkosti výstupného slovníka a jeho úspešnosti už nie je uspokojujúci. Veľkosť vygenerovaného slovníka modifikovanej gramatiky *longbase* je takmer 5 krát väčšia ako slovník gramatiky *mow-500M*, no rozdiel v úspešnosti je len 1 %.



Obr. 6.3: Počet uhádnutých hesiel v referenčnom slovníku myspace v závislosti na veľkosti vygenerovaného slovníka rockyou-65.

Jedným z hlavných cieľov tejto práce je zredukovať veľkosť vygenerovaných slovníkov. O to sa starajú modifikácie *mow-n*, ktoré dokázali systematicky zmenšiť 25 GB (*longbase*) na 130 MB (*mow-20M*). S veľkosťou priamo úmerne súvisí aj čas potrebný na vygenerovanie týchto slovníkov. Modifikácii *mow-20M* zabralo spracovanie gramatiky a vytvorenie všetkých hesiel len 3,5 sekundy. V porovnaní s originálnou gramatikou, ktorej by to trvalo

roky je to obrovský praktický prínos. Úspešnosť všetkých modifikácií na obrázku 6.4 je prijateľná.

V tabuľke 6.4 je možné vidieť klesanie celkovej pravdepodobnosti pravidiel pri ich postupnom odstraňovaní programom PCFG Mower. Táto vlastnosť gramatiky je kľúčová pri filtrovaní pravidiel. Modifikácia *mow-20M* obsahuje už len 79 základných štruktúr z pôvodných 256, no ich celková pravdepodobnosť klesla len o 1,2 %. Podobne to je aj s pravidlami kapitalizácie, ktorých celková pravdepodobnosť klesla z 1700 % na 1681 %. Tento údaj dokazuje, že sa podarilo z pôvodnej gramatiky odstrániť veľkú časť zbytočných pravidiel, ktoré majú minimálny dopad na úspešnosť vygenerovaných hesiel.

Obr. 6.4: Počet uhádnutých hesiel v rôznych slovníkoch použitím modifikovaných gramatík rockyou-65.

6.4 PCFG Mower – útočné slovníky

Druhá časť experimentov sa zameriava na prínos útočných slovníkov, ich použitie spolu s filtrovaním pravidiel a ich vplyv na poradie generovaných hesiel. Tabuľka 6.5 zobrazuje vlastnosti a úspešnosť modifikovaných gramatík a porovnanie s klasickým filtrovaním pravidiel. Ku každej originálnej gramatike bola vytvorená *longbase* modifikácia a z nej dvojice s rovnakým limitom n: *mow-n* a *AD-n*. Štruktúra tejto tabuľky je veľmi podobná predošlej (6.3). Keďže útočné slovníky ovplyvňujú počet alfa reťazcov tak je tento údaj zobrazený ako ďalšia dôležitá metrika gramatík. Modifikácie *AD-n* sú vytvorené programom PCFG Mower s limitom n a s použitím útočných slovníkov best110.txt s prioritou high a best1050.txt s prioritou medium. Obidva tieto slovníky sú dostupné v repozitári SecLists (sekcia 6.1).

Prvá skupina gramatík bola natrénovaná zo slovníka ***Lizard-Squad.txt*** (ls). Pôvodná gramatika obsahuje 9568 alfa reťazcov. Útočné slovníky majú dokopy 1160 hesiel, no niektoré z týchto hesiel sa už nachádzali v gramatike. Program PCFG Mower reálne pridal do gramatiky 667 nových reťazcov. Na gramatiku s pridanými heslami sa následne aplikuje filtrovanie pravidiel. Z tabuľky je možné vidieť, že pridanie nových reťazcov do gramatiky mierne zvýšilo čas potrebný na vygenerovanie všetkých hesiel a taktiež aj ich počet. Najdôležitejší údaj pri testovaní útočných slovníkov je úspešnosť modifikovaných gramatík.

V každom meraní sa podarilo zvýšiť úspešnosť modifikovaných gramatík *AD-n* približne o 4 %. Tým pádom sa zvýšilo aj výsledné ASRI. Modifikovaná gramatika *AD-20M* dokázala vygenerovať všetky heslá len za 4 sekundy, veľkosť vytvoreného slovníka je len 216 MB a úspešnosť je o 0,46 % vyššia ako pri pôvodnej gramatike.

mow-1000M vs AD-1000M:

- čas: 1m 44s → 2m 25s,
- MOP: 523 → 537,
- ASRI: -3,03 % → +1,4 %.

mow-500M vs AD-500M:

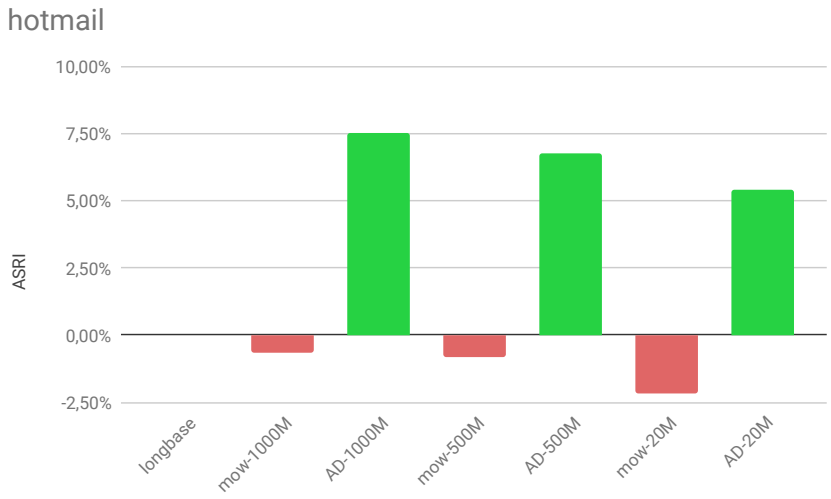
- čas: 55s → 1m 30s,
- MOP: 287 → 297,
- ASRI: -3,31 % → +1,14 %.

mow-20M vs AD-20M:

- čas: 3s → 4s,
- MOP: 19 → 19,5,
- ASRI: -3,91 % → +0,46 %.

Druhá skupina gramatík bola vytvorená zo slovníka ***myspace.txt*** (ms). Modifikácie tejto gramatiky už boli zobrazené v predošlých experimentoch. Súbory *myspace.txt* a *tuscl.txt* sú 2 najväčšie slovníky, nad ktorými prebiehali experimenty. Tieto slovníky vytvoria zložitú gramatiku s veľkým počtom pravidiel. Kritickým faktorom v prípade gramatiky *ms* je veľký počet alfa reťazcov, preto musí program PCFG Mower odstrániť príliš veľa základných štruktúr pri filtrovaní pravidiel. Pridanie útočných slovníkov do tejto gramatiky prinieslo 490 nových reťazcov, ktoré v modifikáciách AD-1000M a AD-500M dokázali zvýšiť ASRI približne o 2 %. V modifikácii AD-20M zostalo veľmi málo základných štruktúr, ktoré už nedokázali pokryť potrebné variácie hesiel a ani nové reťazce z útočných slovníkov už nedokázali vylepšiť úspešnosť na prijateľnú hranicu.

Tretia skupina gramatík *hm* je vytvorená zo súboru ***hotmail.txt***, ktorý obsahuje najmenej hesiel spomedzi všetkých testovacích slovníkov. Podstatou experimentov v tejto skupine je ukázať ako dokážu útočné slovníky ovplyvniť menšie a jednoduchšie typy gramatík. Modifikácia *longbase* dokázala vytvoriť až 1126 miliónov hesiel za 10 minút, čo je najviac zo všetkých testovacích gramatík. V porovnaní s inými gramatikami (napr. *ms*) je úspešnosť jednotlivých meraní veľmi nízka. Útočné slovníky pridali do gramatiky 730 nových alfa reťazcov. Porovnanie modifikácií *mow-1000M* a *AD-1000M* ukazuje, že generovanie hesiel trvalo takmer dva krát dlhšie (3m → 5m 30s) a počet vytvorených hesiel je o 53 miliónov väčší. Veľmi pozitívne bola ovplyvnená úspešnosť modifikácie *AD-1000M*, kde metrika ASRI dosiahla až +7,53 %, čo je o 8 % viac ako *mow-1000M*. Ostatné modifikácie *AD-500M* a *AD-20M* majú taktiež výrazne zvýšenú úspešnosť. Vizuálne porovnanie ASRI pre jednotlivé modifikácie gramatiky *hotmail* zobrazuje obrázok 6.5. Všetky *AD-n* modifikácie majú veľmi dobré výsledky vo všetkých smeroch – čas, veľkosť aj úspešnosť. Najlepšie dopadla modifikácia AD-20M, ktorá vytvorila za 6 sekúnd slovník o veľkosti 224 MB, ktorý má o 5,44 % lepšiu úspešnosť ako originálna gramatika.



Obr. 6.5: Porovnanie ASRI modifikovaných gramatík hotmail.

	Gramatika				Vygenerovaný slovník			Úspešnosť				
tr	Názov	ZŠ	Kap	Alfa	Čas	Veľkosť	MOP	pr	ms	dw	r65	ASRI
ls	original	1415	386	9568	10m*	6,5 GB	598	22,25 %	18,44 %	41,04 %	22,85 %	
	longbase	1122	386	9568	10m*	8 GB	738	22,22 %	18,45 %	40,99 %	22,84 %	-0,02 %
	mow-1000M	117	51	9568	1m 44s	5,2 GB	523	19,74 %	15,35 %	36,82 %	20,55 %	-3,03 %
	AD-1000M	117	51	10235	2m 25s	5,3 GB	537	23,69 %	17,62 %	44,35 %	24,43 %	+1,4 %
	mow-500M	101	49	9568	55s	2,8 GB	287	19,57 %	15,16 %	36,20 %	20,39 %	-3,31 %
	AD-500M	101	49	10235	1m 30s	2,9 GB	297	23,58 %	17,43 %	43,79 %	24,28 %	+1,14 %
	mow-20M	64	42	9568	3s	213 MB	19	19,22 %	14,52 %	35,25 %	19,93 %	-3,91 %
	AD-20M	62	42	10235	4s	216 MB	19,5	23,21 %	16,72 %	42,73 %	23,76 %	+0,46 %
ms	original	1574	179	22587	10m*	5,7 GB	616	47,47 %	93,68 %	69,14 %	46,42 %	
	longbase	1430	179	22587	10m*	9,5 GB	1030	47,45 %	94,38 %	69,07 %	46,42 %	+0,15 %
	mow-1000M	110	25	22587	3m	9,2 GB	941	46,37 %	82,40 %	66,74 %	43,04 %	-4,42 %
	AD-1000M	110	25	23077	4m	10 GB	960	48,49 %	82,41 %	70,02 %	44,59 %	-2,67 %
	mow-500M	78	24	22587	1m	3,2 GB	334	45,13 %	79,67 %	64,71 %	42,62 %	-6,03 %
	AD-500M	78	24	23077	1m 18s	3,3 GB	340	47,42 %	79,76 %	68,10 %	44,21 %	-4,19 %
	mow-20M	21	23	22587	2s	126 MB	15	33,25 %	61,17 %	54,28 %	35,58 %	-17,99 %
	AD-20M	21	23	23077	3s	128 MB	15	35,18 %	61,29 %	57,23 %	36,81 %	-16,43 %
hm	original	940	153	6602	10m*	8,5 GB	852	12,72 %	9,15 %	26,42 %	16,12 %	
	longbase	838	153	6602	10m*	13,5 GB	1126	12,69 %	9,15 %	26,37 %	16,11 %	-0,02 %
	mow-1000M	151	88	6602	3m	8,8 GB	835	12,53 %	8,50 %	25,14 %	15,72 %	-0,63 %
	AD-1000M	150	84	7332	5m 30s	9,4 GB	888	19,14 %	13,94 %	38,04 %	23,43 %	+7,53 %
	mow-500M	142	64	6602	1m 40s	5,5 GB	494	12,36 %	8,37 %	24,82 %	15,64 %	-0,80 %
	AD-500M	127	52	7332	3m	4,7 GB	432	18,58 %	13,24 %	37,16 %	22,57 %	+6,78 %
	mow-20M	86	34	6602	3s	213 MB	18	11,87 %	7,16 %	23,02 %	13,68 %	-2,17 %
	AD-20M	83	34	7332	6s	224 MB	19	18,42 %	11,76 %	35,64 %	20,35 %	+5,44 %

Tabuľka 6.5: Vplyv útočných slovníkov na vlastnosti gramatiky a jej úspešnosť (* - bol dosiahnutý časový limit)

Úspešnosť útočných slovníkov bez filtrovania pravidiel

Tento typ experimentov ukazuje prínos útočných slovníkov bez filtrovania pravidiel. Keďže je možné priradiť prioritu jednotlivým útočným slovníkom tak môže užívateľ ovplyvniť poradie generovania s novými reťazcami. Princíp experimentu je následovný:

1. Vytvoriť gramatiku programom PCFG Trainer.

2. Definovať konfiguračný súbor pre útočné slovníky.

3. Vytvoriť novú gramatiku aplikovaním útočných slovníkov pomocou programu PCFG Mower.

4. Nad obidvomi gramatikami spustiť generátor hesiel s časovým limitom 10 minút.

5. Porovnať úspešnosť jednotlivých meraní.

Ako trénovacie slovníky boli zvolené *rockyou-65*, *myspace* a *hotmail*. Konfigurácia útočných slovníkov bola definovaná následovne: best110.txt s prioritou high, best1050.txt s prioritou medium a 10-million-password-list-top-10000.txt s prioritou low. Nad pôvodnými gramatikami sa spustil program PCFG Mower, ktorý vytvoril nové modifikácie s pridanými reťazcami bez filtrovania pravidiel. To znamená, že jediný rozdiel medzi pôvodnou a modifikovanou gramatikou je počet alfa reťazcov. Generátor hesiel všetkých modifikácií mal nastavený limit 10 minút.

	Gramatika			Vygenerovaný slovník		Úspešnosť			
tr	Názov	Alfa	L^p	Veľkosť	MOP	pr	ms	dw	ASRI
r65	original	17845	195	1,5 GB	151	72,34 %	37,63 %	88,25 %	
	AD-RP	22923	10245	287 MB	28	80,74 %	37,20 %	92,79 %	+4,2 %
	AD-SP	22923	195	2,6 GB	260	80,81 %	39,71 %	93,38 %	+5,2 %
ms	original	22587	232	5,7 GB	616	47,47 %	93,68 %	69,14 %	
	AD-RP	29075	10275	404 MB	41	69,28 %	79,24 %	87,97 %	+8,7 %
	AD-SP	29075	232	4,5 GB	489	69,32 %	93,30 %	88,86 %	+13,7 %
hm	original	6602	95	8,5 GB	852	12,7 %	9,15 %	26,42 %	
	AD-RP	16013	10234	795 MB	72	61,39 %	31,73 %	83,84 %	+42,8 %
	AD-SP	16013	95	16 GB	1355	61,42 %	33,07 %	84,38 %	+43,5 %

Tabuľka 6.6: Vplyv útočných slovníkov na úspešnosť gramatiky bez filtrovania pravidiel.

Tabuľka 6.6 zobrazuje výsledky experimentov tohto typu. Modifikovaná gramatika **AD-RP** (angl. attack dictionaries - random probability) obsahuje nové reťazce pridané podľa princípu v sekcii 5.5. To znamená, že každý reťazec z útočných slovníkov má priradenú náhodnú pravdepodobnosť podľa priority slovníka. Vo všetkých troch modifikáciách AD-RP sa výrazne znížila veľkosť vygenerovaného slovníka, t.j. rýchlosť generovania hesiel. Toto negatívne správanie zapríčinil obrovský počet pravdepodobnostných skupín alfa reťazcov L^p, ktoré boli detailne opísané v sekcii 3.7. V tomto prípade znamená každý pridaný reťazec novú pravdepodobnostnú skupinu. Počet pravdepodobnostných skupín alfa reťazcov v modifikácii AD-RP je takmer 53 krát väčší ako v pôvodnej gramatike, čo značne ovplyvní výpočetný čas algoritmu *Deadbeat Dad*. Aj so značne menším počtom vygenerovaných hesiel má modifikácia AD-RP lepšiu úspešnosť oproti pôvodnej gramatike.

V modifikácii **AD-SP** (attack dictionarries - same probability) bol použitý opačný princíp prideľovania pravdepodobnosti novým reťazcom. Každému reťazcu rovnakej dĺžky je pridelená rovnaká pravdepodobnosť podľa priority. Táto pravdepodobnosť sa určí podľa pravidla na konkrétnej pozícii:

- vysoká priorita: 3 % všetkých pravidiel,
- stredná priorita: 20 % všetkých pravidiel,
- nízka priorita: 40 % všetkých pravidiel.

To znamená, že všetky nové reťazce o dĺžke 8 budú mať rovnakú pravdepodobnosť ako už existujúce pravidlo, preto sa počet pravdepodobnostných skupín L^p v modifikácii AD-SP nijak nelíši oproti pôvodnej gramatike. Takýmto spôsobom je možné pridať do gramatiky nové reťazce a neznížiť rýchlosť generovania hesiel.

Všetky tri útočné slovníky použité v tomto experimente majú dokopy 11160 slov. Niektoré z nich sa už v gramatikách vyskytovali a preto sa reálny počet pridaných reťazcov líši v každej gramatike. Do pôvodnej gramatiky *r65* bolo pridaných 5078 reťazcov, do *ms* 6488 a *hm* má 9411 nových reťazcov. Všetky slová v použitých útočných slovníkoch sa dajú označiť ako „silné“, pretože obsahujú najpravdepodobnejšie heslá za posledné roky. Čím väčší je počet pridaných reťazcov v gramatike, tým viac sa zlepší úspešnosť modifikovanej gramatiky. Gramatika *hm* je praktický príklad toho ako môžu útočné slovníky výrazne vylepšiť aj zdanlivo malú a „slabú“ gramatiku.

Vo všetkých troch skupinách gramatík bola použitím útočných slovníkov zvýšená úspešnosť. V modifikácii *r65* AD-SP sa zvýšilo výsledné ASRI o 5,2 % a vďaka väčšiemu počtu alfa reťazcov sa vygeneroval takmer dvojnásobný počet hesiel. Modifikácia *ms* AD-SP má zvýšené ASRI o 13,7 %, no celkový počet hesiel sa nezvýšil. Veľmi dobré výsledky dosiahla modifikácia hm AD-SP, ktorá zdvojnásobila celkový počet hesiel a zároveň zvýšila ASRI až o 43,5 %.

6.5 Zhrnutie výsledkov

Program PCFG Mower poskytuje niekoľko techník na zdokonalenie generovania hesiel podľa pravdepodobnosti. Pomocou týchto techník je možné niekoľkonásobne urýchliť proces generovania hesiel, zmenšiť veľkosť vygenerovaného slovníka pri zachovaní prijateľnej úspešnosti a zvýšiť úspešnosť pridaním nových slov do gramatiky. Zrýchleniu generátora pomohlo najviac odstránenie dlhých základných štruktúr z gramatiky. V analyzovaných gramatikách spôsobovali tieto dlhé základné štruktúry obrovské výkonnostné problémy a neprinášali takmer žiadny prínos. Pravidlá s dlhými základnými štruktúrami majú väčšinou pridelenú bezvýznamnú pravdepodobnosť, no komplikujú beh zložitého algoritmu Deadbeat Dad. Vo všetkých experimentoch sa po odstránení dlhých štruktúr výrazne urýchlil proces generovania hesiel.

Pomocou filtrovania pravidiel dokáže program PCFG Mower limitovať celkový počet hesiel, ktoré môže gramatika vygenerovať. Generátor tým pádom nestráca čas spracovaním zbytočných základných štruktúr. Bolo dokázané, že každá gramatika má tzv. hranicu počtu hesiel, po ktorej sa už neoplatí ďalej vytvárať heslá, pretože úspešnosť stúpa veľmi pomaly vzhľadom na rýchlo rastúcu veľkosť vygenerovaného slovníka. Každá gramatika má túto hranicu rozdielnu a pomáha ju nájsť vytvorený skript pre automatické testovanie. Táto technika dosahuje obrovské zredukovanie vygenerovaného slovníka s takmer rovnakou pravdepodobnosťou ako pôvodná, neupravená gramatika. Ako príklad slúži gramatika rockyou-65, v ktorej sa podarilo zredukovať výstupný slovník z 1,5 GB (10 min) na 130 MB (3,5 sek) s poklesom priemernej úspešnosti len 2,47 %.

Pridanie útočných slovníkov do gramatiky prinieslo pozitívny vplyv na úspešnosť vytvorených hesiel. Spolu s filtrovaním pravidiel je možné vytvoriť silnejšiu modifikáciu pôvodnej gramatiky s limitovaným počtom hesiel, ktorá dosahuje lepšiu úspešnosť. Táto kombinácia všetkých funkcionalít programu PCFG Mower upraví gramatiku tak, aby bola rýchlejšia, úspešnejšia a generovala malé slovníky hesiel v zanedbateľnom čase. Útočné slovníky majú využitie aj bez filtrovania pravidiel. Svojou prioritou dokážu zmeniť poradie generovaných

hesiel a pridať tak nové variácie hesiel. Gramatika myspace dokázala touto technikou zvýšiť svoju úspešnosť o 13,7 %.

Kapitola 7

Záver

Hlavným cieľom tejto práce je navrhnúť zdokonalenie pravdepodobnostných metód nástroja PCFG Cracker [21], ktoré odstráni súčasné problémy obrovských výstupných slovníkov a pomalého generovania hesiel. V práci sú úspešne identifikované kritické miesta, ktoré spôsobujú tieto problémy. Je vytvorený návrh na zredukovanie vygenerovaných slovníkov a celkové zrýchlenie generovania hesiel pomocou filtrovania vstupnej gramatiky, odstránením dlhých základných štruktúr v gramatike a implementáciou útočných slovníkov do súčasného riešenia nástroja PCFG Cracker.

Navrhnuté vylepšenia sú implementované v programe PCFG Mower, ktorý slúži ako voliteľný modul medzi fázou trénovania pravdepodobnostnej gramatiky a generátorom hesiel. Program poskytuje pokročilú metódu filtrovania vstupnej gramatiky. Táto metóda odstráni tie najmenej pravdepodobné základné štruktúry a pravidlá kapitalizácie. Týmto prístupom sa podarilo niekoľkonásobne zmenšiť veľkosť výstupného slovníka pri zachovaní prijateľnej úspešnosti vygenerovaných hesiel. Po odstránení dlhých základných štruktúr, ktoré spôsobovali obrovské výkonnostné problémy v algoritme *Deadbeat Dad* [20], sa výrazne urýchlil celkový beh generátora hesiel.

Súčasná verzia nástroja PCFG Cracker využíva na vytváranie variácií hesiel len tie reťazce, ktoré sa vyskytujú v trénovacom súbore a nepodporuje generovanie hesiel pomocou sekundárnych slovníkov, čo značne obmedzuje rozsah generovaných hesiel. Program PCFG Mower umožňuje pridať do vytvorenej gramatiky nové reťazce zo súborov označované ako útočné slovníky. Každý útočný slovník má konfigurovateľnú prioritu, ktorá ovplyvňuje pravdepodobnosť pridaných reťazcov. Pomocou tejto techniky je možné vytvoriť robustnejšiu a cielenú gramatiku, ktorá bude obsahovať užívateľom definované alfa reťazce. Pridaním útočných slovníkov do pravdepodobnostnej gramatiky sa zvýšila úspešnosť vygenerovaných slovníkov o 5 až 43 %. Kombináciou všetkých techník, ktoré poskytuje PCFG Mower je možné vytvoriť silnejšiu gramatiku bez zbytočných pravidiel, ktorá vygeneruje niekoľkonásobne menší slovník v zanedbateľnom čase s vyššou priemernou úspešnosťou.

Ďalší výskum v tejto oblasti by sa mohol zamerať na analýzu algoritmu *Deadbeat Dad*, identifikovať jeho slabé miesta a navrhnúť potrebnú optimalizáciu. Spracovanie dlhých základných štruktúr týmto algoritmom je tak náročné, že sa v niektorých prípadoch generátor hesiel úplne zasekne. Možným riešením by mohol byť návrh nového algoritmu, ktorý bude generovať potomkov v derivačnom strome a kompletne tak nahradí algoritmus *Deadbeat Dad*. Ďalší námet na budúce vylepšenie je použitie útočných slovníkov v praxi, výber správnych slovníkov a ich prínos pri cielenom útoku a naviazať tak na prácu S. H. Yazdi [22]. Taktiež je možné pokračovať s touto problematikou v programe PCFG Mower a implementovať podporu aj pre iné typy pravidiel (čísla, špeciálne symboly, klávesové vzory).

Dodatok A

Obsah CD

Priložené CD má nasledujúci obsah:

/doc – užívateľská príručka,

/src – zdrojové kódy programu PCFG Mower, skript pre automatické testovanie,

/rules – pravdepodobnostné gramatiky použité v tejto práci,

/latex – zdrojové súbory pre Latex,

/xlisti00_DP.pdf – písomná správa vo formáte PDF.

Literatúra

[1] Bishop, M.; Klein, D. V.: Improving system security via proactive password checking. *Computers & Security*, ročník 14, č. 3, 1995: s. 233–249.

[2] Bonneau, J.: The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*, May 2012, ISSN 2375-1207, s. 538–552, doi:10.1109/SP.2012.49.

[3] Bonneau, J.: The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE Symposium on Security and Privacy*, May 2012, ISSN 2375-1207, s. 538–552, doi:10.1109/SP.2012.49.

[4] Crumpacker, J. R.: *Distributed Password Cracking*. Diplomová práca, Naval Postgraduate School, 2009.

[5] Dell' Amico, M.; Michiardi, P.; Roudier, Y.: Password Strength: An Empirical Analysis. In *2010 Proceedings IEEE INFOCOM*, March 2010, ISSN 0743-166X, s. 1–9.

[6] Gazdík, P.: *Využití heuristik při obnově hesel pomocí GPU*. Bakalárska práca, Vysoké učení technické v Brně, 2015.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18210

[7] Group, T. O.: John the Ripper password cracker [online]. [cit. 2019-01-09].
URL https://www.openwall.com/john/

[8] Handschuh, H.: *SHA-0, SHA-1, SHA-2 (Secure Hash Algorithm)*. Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 1190–1193, doi:10.1007/978-1-4419-5906-5_615.
URL https://doi.org/10.1007/978-1-4419-5906-5_615

[9] Houshmand, S.; Aggarwal, S.; Flood, R.: Next Gen PCFG Password Cracking. *IEEE Transactions on Information Forensics and Security*, ročník 10, č. 8, Aug 2015: s. 1776–1791, ISSN 1556-6013, doi:10.1109/TIFS.2015.2428671.

[10] Hranický, R.: *Digitální forenzní analýza s použitím distribuovaného prostředí*. Dizertačná práca, Vysoké Učení Technické V Brně, 2016.

[11] Kos, O.: *Obnova hesel v distribuovaném prostředí*. Diplomová práca, Vysoké učení technické v Brně, 2016.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=18213

[12] Mikuš, D.: *Distribuované generovanie hesiel pomocou pravdepodobnostných gramatík*. Diplomová práca, Vysoké učení technické v Brně, 2019.

[13] Montoro, M.: Cain & Abel [online]. [cit. 2019-01-09].
URL http://www.oxid.it

[14] Narayanan, A.; Shmatikov, V.: Fast Dictionary Attacks on Passwords Using Time-space Tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-226-7, s. 364–372, doi:10.1145/1102120.1102168.
URL http://doi.acm.org/10.1145/1102120.1102168

[15] R. Hranický, O. R. V. V., P. Matoušek: Experimental Evaluation of Password Recovery in Encrypted Documents. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, 2016, ISBN 978-989-758-167-0, s. 299–306, doi:10.5220/0005685802990306.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11052

[16] RABINER, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Readings in Speech Recognition*, editácia A. Waibel; K.-F. Lee, San Francisco: Morgan Kaufmann, 1990, ISBN 978-1-55860-124-6, s. 267 – 296, doi:https://doi.org/10.1016/B978-0-08-051584-7.50027-9.
URL
http://www.sciencedirect.com/science/article/pii/B9780080515847500279

[17] Schmied, J.: *GPU akcelerované prolamování šifer*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2014.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=16858

[18] Shuanglei, Z.: Project RainbowCrack [online]. [cit. 2019-01-09].
URL http://project-rainbowcrack.com/

[19] V. Klein, D.: "Foiling the Cracker": ASurvey of, and Improvements to, Password Security. 06 1992.

[20] Weir, M.: *Using Probabilistic Techniques to Aid in Password Cracking Attacks*. Dizertačná práca, Florida State University, 2010.

[21] Weir, M.; Aggarwal, S.; d. Medeiros, B.; aj.: Password Cracking Using Probabilistic Context-Free Grammars. In *2009 30th IEEE Symposium on Security and Privacy*, May 2009, ISSN 1081-6011, s. 391–405.

[22] Yazdi, S. H.: *Probabilistic Context-Free Grammar Based Password Cracking: Attack, Defense and Applications*. Dizertačná práca, Florida State University, 2015.

[23] Češka, M.; Vojnar, T.; Smrčka, A.; aj.: Teoretická informatika. 2018, str. 53.
URL https:
//www.fit.vutbr.cz/study/courses/TIN/public/Texty/TIN-studijni-text.pdf