VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

LÁMÁNÍ HESEL POMOCÍ ALGORITMU PRINCE V SYSTÉMU FITCRACK

PASSWORD CRACKING USING PRINCE ALGORITHM AND FITCRACK SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DÁVID BOLVANSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK HRANICKÝ**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Bolvanský Dávid, Bc.**
Program: Informační technologie Obor: Počítačové sítě a komunikace
Název: **Lámání hesel pomocí algoritmu PRINCE v systému Fitcrack**
Password Cracking Using PRINCE Algorithm and Fitcrack System
Kategorie: Paralelní a distribuované výpočty

Zadání:

1. Seznamte se s architekturou a implementací systému Fitcrack.
2. Seznamte se s algoritmem Probability Infinite Chained Elements (PRINCE) a zvažte, jak jej realizovat distribuovaně.
3. Po konzultaci s vedoucím navrhněte rozšíření systému Fitcrack, které umožní realizovat distribuovaný útok pomocí algoritmu PRINCE.
4. Navržení řešení implementujte a otestujte.
5. Navrhněte sadu experimentů, které srovnají vaše řešení s existujícími typy útoků. Zaměřte se na faktory jako výkon, doba výpočtu, režie, škálovatelnost, apod.
6. Experimenty realizujte a zhodnoťte dosažené výsledky.

Literatura:

- HRANICKÝ Radek, ZOBAL Lukáš, RYŠAVÝ Ondřej and KOLÁŘ Dušan. Distributed Password Cracking with BOINC and Hashcat. *Digital Investigation*, vol. 2019, no. 30, pp. 161-172. ISSN 1742-2876.
- HRANICKÝ Radek, ZOBAL Lukáš, VEČEŘA Vojtěch a MÚČKA Matúš. The Architecture of Fitcrack Distributed Password Cracking System. FIT-TR-2018-03, Brno, 2018.
- Další dle dohody s vedoucím

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2, část bodu 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hranický Radek, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 29. října 2019

Abstrakt

Algoritmus PRINCE je rýchlejšou a pokročilejšou verziou kombinačného útoku. Nedistribuované lámanie hesiel často naráža na svoje limity a jeho použiteľnosť na reálne úlohy sa znižuje kvôli stúpajúcim nárokom na výpočtové zdroje zariadenia. Cieľom tejto práce je navrhnúť distribuovanú verziu útoku pomocou algoritmu PRINCE ako rozšírenie systému Fitcrack, ktorý sa zameriava na distribuované lámanie hesiel. Predkladaný návrh je následne implementovaný a integrovaný do systému Fitcrack. Práca sa venuje útoku PRINCE na sade experimentov, kde sa skúma vplyv rôznych konfiguračných možností, ktoré útok ponúka. Súčasťou experimentálnej časti je porovnanie útoku PRINCE so slovníkovým a kombinačným útokom. Cieľom je nájsť prípady, kedy je útok PRINCE lepší ako ostatné útoky. Integrované riešenie útoku PRINCE v systéme Fitcrack je na záver porovnané s riešením implementovaným v systéme Hashtopolis.

Abstract

The PRINCE algorithm is a faster and more advanced version of a combination attack. Non-distributed password breaking often encounters its limits, and its applicability to real tasks decreases due to the increasing demand for computing resources of the device. The aim of this work is to design a distributed version of the the PRINCE attack as an extension of Fitcrack system, which focuses on distributed password cracking. The proposed design is implemented and integrated into the Fitcrack system. The work examines the PRINCE attack on a set of experiments, which examines the impact of various configuration options. Part of the experimental part is a comparison of the PRINCE attack with the dictionary and combination attack. The purpose of the comparison is to find cases where the PRINCE attack is better than other attacks. Finally, the integrated PRINCE attack solution in the Fitcrack system is compared with the solution implemented in the Hashtopolis system.

Kľúčové slová

Fitcrack, lámanie hesiel, distribuovaný útok, hashcat, OpenCL, GPU, princeprocessor, PRINCE, reťazce, prvky

Keywords

Fitcrack, password cracking, distributed attack, hashcat, OpenCL, GPU, princeprocessor, PRINCE, chains, elements

Citácia

BOLVANSKÝ, Dávid. *Lámání hesel pomocí algoritmu PRINCE v systému Fitcrack*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický

Lámání hesel pomocí algoritmu PRINCE v systému Fitcrack

Prehlásenie

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

.....
Dávid Bolvanský
30. júla 2020

Poďakovanie

Rád by som vyslovil poďakovanie vedúcemu mojej diplomovej práce Ing. Radkovi Hranickému za pomoc, rady a venovaný čas pri písaní tejto práce. Ďakujem ostatným členom Fitcrack tímu za technické rady týkajúce sa systému Fitcrack.

Obsah

Kapitola 1

Úvod

Mnoho softvérových služieb obsahuje autentifikačný systém, ktorý sa spolieha na kombináciu používateľského mena a hesla. Používatelia majú tendenciu si vyberať heslá, ktoré sú ľahko zapamätateľné a neobsahujú dostatočnú náhodnosť [4]. Takéto heslá sú veľmi predvídateľné, čo umožňuje útočníkovi vygenerovať rôzne heslá a postupne sa s nimi skúsiť prihlasovať. Pomocou získaného hesla môže útočník získať prístup k citlivým informáciám alebo môže zneužiť získané oprávnenia v systéme.

Okrem lámania hesiel za účelom páchania nezákonnej činnosti sa lámanie hesiel používa aj za účelom hľadania zabudnutého hesla alebo pri forenznej analýze. Občas je potrebné získať prístup k zabezpečeným informáciám, ktoré môžu pomôcť pri vyšetrovaní trestnej činnosti. Často používaným spôsobom je práve lámanie hesiel. Principálne ide o jednoduchú činnosť - generujú sa heslá, ktoré sa priebežne skúšajú, až pokým nie je nájdené správne heslo. Proces lámania hesiel býva neraz značne výpočtovo náročný, keďže heslá, ktoré zabezpečujú citlivé informácie, sú dlhé a komplikované.

Na lámanie hesiel sa používajú typicky všetky dostupné prostriedky ako sú CPU a GPU. Jedno zariadenie však vo väčšine prípadov vôbec nestačí kvôli vysokým výpočtovým nárokom na zdroje zariadenia [16]. Riešením je použiť systém s distribuovanou architektúrou, kde sa celý proces lámania rozdelí medzi jednotlivých klientov [14]. Príkladom tohto typu systému je systém Fitcrack [18].

Algoritmus PRINCE je pokročilejšou formou kombinačného útoku [29]. Algoritmus je všeobecne veľmi rýchly a výrazný rozdiel sa prejavuje najmä pri pomalých hešovacích algoritmoch a pri viacslovných kandidátnych heslách. Útok PRINCE v súčasnosti podporuje systém Hashtopolis [7], ktorý je náročnejší na používanie a určený skôr pre expertov.

Práca si kladie za cieľ navrhnúť a implementovať distribuovaný útok pomocou algoritmu PRINCE, ktorý bude integrovaný do systém Fitcrack, čím sa rozšíria možnosti systému o ďalší distribuovaný útok. Používateľ systému bude môcť lámať heslá pomocou útoku PRINCE a naplno využívať silné stránky algoritmu PRINCE pre svoje potreby. Vďaka webovému rozhraniu systému Fitcrack bude použitie útoku PRINCE veľmi jednoduché a priamočiare. Systém Fitcrack nebude na konfiguráciu útoku PRINCE vyžadovať expertné znalosti ako v systéme Hashtopolis a zároveň ponúkne plnú kontrolu nad algoritmom PRINCE. Okrem základného nastavenia úlohy bude webové rozhranie ponúkať aj možnosti na pokročilé nastavenie útoku.

Ďalším cieľom práce je analýza vplyvu rôznych konfiguračných možností algoritmu PRINCE na množstvo vygenerovaných kandidátnych hesiel. Hlavná časť experimentov sa venuje analýze škálovateľnosti útoku PRINCE. Ich cieľom je preskúmať, ako efektívne sa útok distribuuje medzi klientov. Experimenty s rôznymi časmi pre pracovnú jednotku skú-

majú vplyv veľkosti pracovnej jednotky na efektivitu a dobu výpočtu úlohy s útokom PRINCE. Súčasťou experimentálnej časti je aj porovnanie navrhnutého distribuovaného útoku so slovníkovým a kombinačným útokom na sade rôznych experimentov za účelom nájdenia prípadov, kedy je útok PRINCE lepší ako ostatné útoky. Prínosom tejto práce je aj porovnanie implementácií útoku PRINCE medzi systémami Fitcrack a Hashtopolis.

Celá práca sa skladá zo ôsmich kapitol. V kapitole 2 je predstavená platforma BOINC slúžiaca na distribuované výpočty. Popis systému Fitcrack, ktorý sa používa na distribuované lámanie hesiel, sa nachádza v kapitole 3. Kapitola 4 sa zaoberá algoritmom PRINCE. Návrh realizácie útoku PRINCE v rámci systému Fitcrack popisuje kapitola 5. Kapitola 6 sa venuje popisu implementácie vytvoreného návrhu a testovaniu. Popis vykonaných experimentov a zhodnotenie ich výsledkov sa nachádza v kapitole 7. Posledná kapitola 8 obsahuje zhrnutie výsledkov tejto práce a navrhuje možné budúce vylepšenia a rozšírenia.

Kapitola 2

Platforma BOINC

BOINC (*Berkeley Open Infrastructure for Network Computing*) je nekomerčný program určený záujemcom o poskytnutie voľného výkonu svojho počítača pre rôzne projekty prevažne univerzitného charakteru. Jedná sa o softvérovú platformu špeciálne vyvinutú pre distribuované výpočty využívajúcu dobrovoľne poskytnuté zdroje počítačov pripojených do siete Internet [3]. Systém BOINC je vhodný na použitie v rôznych projektoch a jedným z nich je projekt SETI@home pre ktorý bola vlastne táto platforma vyvinutá. Ak by bol systém BOINC jediným počítačom, ktorý by zahŕňal viac ako 310 000 účastníkov a 800 000 zariadení, bol by to štvrtý najvýkonnejší superpočítač na svete [12].

Väčšina BOINC projektov využíva výpočtové zdroje od dobrovoľníkov, ktoré sú spravované centralizovaným serverom, do ktorého sa pracovné úlohy po dokončení tiež vracajú. Riadiaci server pre konkrétny projekt rozdeľuje bloky práce na požiadanie, zaznamenáva získané výsledky a sleduje úsilie každého účastníka. Dobrovoľné zdroje môžu pochádzať z rôznych typov systémov, od GPGPU (viacúčelové GPU), cez viac výkonných CPU až po všadeprítomné mobilné zariadenia. Dobrovoľníci si môžu nainštalovať softvér BOINC do svojho počítača alebo mobilného zariadenia a a začnú tak prispievať svojimi voľnými zdrojmi svojih zariadení do konkrétneho počítačového projektu. U mobilných zariadení sa zdroje používajú iba v prípade, ak má zariadenie nabitú batériu na najmenej 90%.

Dobrovoľnícke výpočty sú najvhodnejšie pre vysoko náročné výpočty, kde pracovné zaťaženie pozostáva z veľkých skupín alebo prúdov úloh a ich cieľom je vysoká miera dokončenia úlohy. Tento typ výpočtov je menej vhodný pre pracovné zaťaženia, ktoré majú extrémne požiadavky na pamäť, úložisko alebo pre ktoré je pomer sieťovej komunikácie voči samotnému počítaniu mimoriadne vysoký. Dobrovoľnícke výpočty sa líšia od iných foriem vysokovýkonných výpočtov, ako sú *grid computing* alebo *cloud computing*. Každý z týchto faktorov predstavuje výzvy, ktorým musí táto platforma čeliť.

Počítače dobrovoľníkov sú anonymné, nedôveryhodné, neprístupné a nekontrolovateľné. Niektoré môžu začať vykazovať zvláštne správanie, ktoré nie je možné zastaviť alebo opraviť. Problémové správania musia byť odhalené čo najskôr a vyradené z výpočtov. Počítače môžu vrátiť úmyselne alebo aj neúmyselne nesprávne výsledky výpočtov, a preto je nutné aby ich správnosť bola následne overovaná. Miera dobrovoľníckych výpočtov je veľká - až milióny počítačov a milióny úloh za deň. Z tohto dôvodu musí byť softvér na serveroch efektívny a škálovateľný.

Systém BOINC sa používa ako základ pre množstvo distribuovaných výpočtových projektov. Medzi najznámejšie projekty [12] patria:

- **SETI@home** - Projekt je zameraný na hľadanie inteligentného mimozemského života vo vesmíre prostredníctvom analýzy vzoriek signálov z najväčšieho rádioteleskopu na svete v Arecibo. Vďaka svojmu zameraniu patrí tento projekt medzi najpopulárnejšie v rámci infraštruktúry BOINC.

- **Rosetta@home** - Jedná sa o projekt, ktorého cieľom je predpovedanie trojrozmerných tvarov rôznych proteínov. Modelovaním jednotlivých bielkovinových reťazcov sa snaží nájsť tvar s najnižším energetickým obsahom, ktorý by mal dosiahnuť skutočný proteín v bunke. Hotové modely posiela do medzinárodnej proteínovej databanky. Tie sú zadarmo prístupné širokej vedeckej obci na ďalší výskum. Projekt si sľubuje početné využitie, a to napríklad pri objavovaní vhodných účinných liekov pre mnohé choroby ako sú malária, AIDS, rakovina, Alzheimerova choroba a iné choroby súvisiace s proteínmi a poškodením DNA.

- **Asteroids@home** - Projekt vznikol na Astronomickom ústave UK v spolupráci s Czech National Team. Cieľom projektu je využiť potenciál distribuovaných výpočtov k riešeniu časovo náročnej úlohy, kde sa skúma inverzia svetelných kriviek planétok. Ako vstupné dáta slúžia fotometrie planétok z astrometrických projektov a výstupom sú trojrozmerné modely planétok.

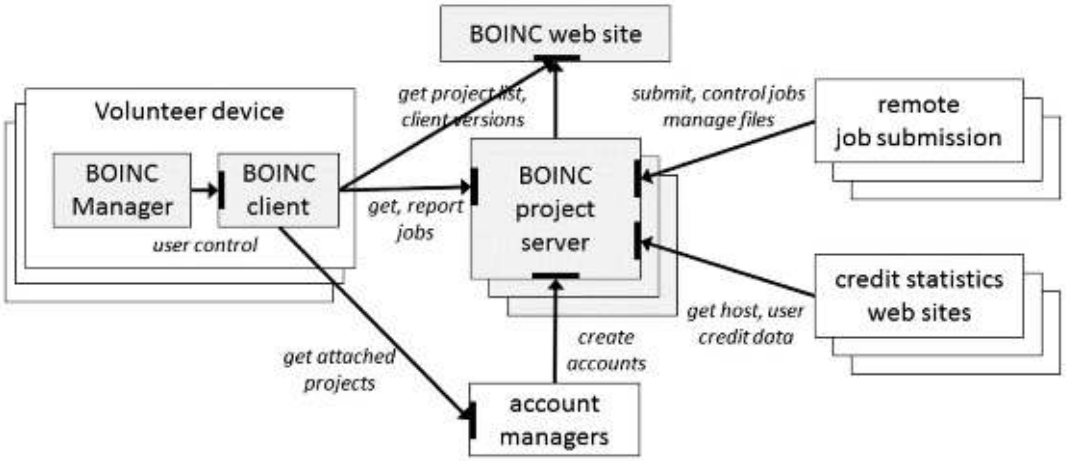
- **Einstein@home** - Cieľom projektu je hľadanie dôkazov o kontinuálnych zdrojoch gravitačných vĺn, ktoré by mohli identifikovať objekty, ako sú napríklad rotujúce neosymetrické neutrónové hviezdy. Je určený na vyhľadávanie signálov z pulzarov v údajoch z LIGO Gravitational Wave Laser Interference Observatory v USA a GEO600 Gravitational Wave Observatory v Nemecku. Pretože sa predpokladá, že niektoré pulzary nie sú úplne sférické, podľa všeobecnej relativity by tieto hviezdy mali emitovať charakteristické gravitačné vlny, ktoré majú LIGO a GEO600 zistiť v blízkej budúcnosti.

- **Cosmology@home** - Snahou tohto projektu je nájsť kozmologický model, ktorý najlepšie popisuje náš vesmír.

- **Quake Catcher Network** - Projekt spracováva údaje zhromaždené senzormi a mobilnými zariadeniami na identifikáciu seizmickej aktivity.

2.1 Komponenty BOINC

Architektúra BOINC zahŕňa viacero komponentov komunikujúcich prostredníctvom rozhraní RPC založeného na technológii HTTP. Je navrhnutá tak, aby boli modulárna a rozšíriteľná. Obrázok 2.1 zobrazuje komponenty BOINC, ktoré bude opísané ďalej. Tmavé (tieňované) časti predstavujú distribúciu softvéru BOINC, svetlé časti predstavujú komponenty vyvinuté tretími stranami.



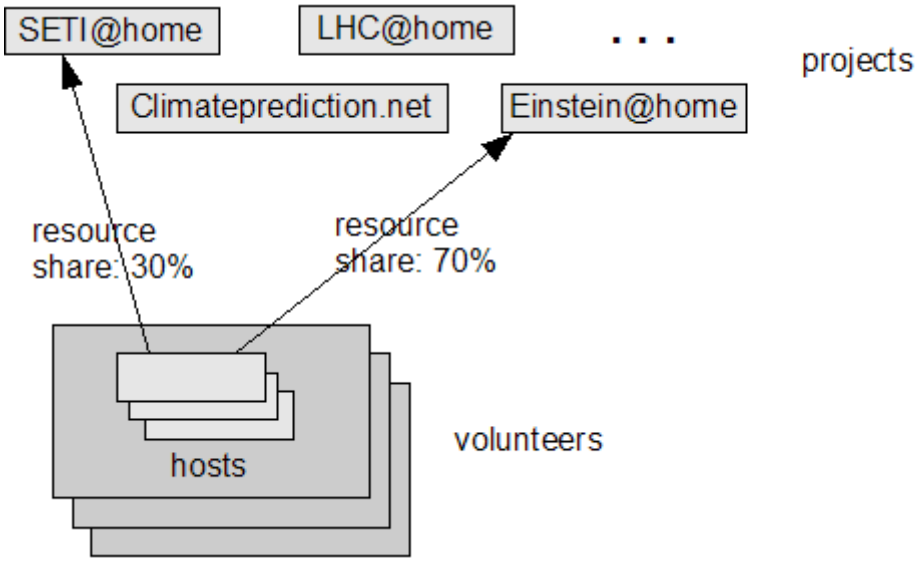
Obrázok 2.1: Komponenty a RPC rozhrania systému BOINC [3].

2.1.1 Projekty a dobrovoľníci

V terminológii BOINC je projekt entita, ktorá používa platformu BOINC na vykonávanie výpočtov. Každý projekt má server, založený na serverovom softvéri BOINC, ktorý distribuuje úlohy na zariadenia dobrovoľníkov. Projekty sú autonómne a nezávislé. Projekt môže slúžiť jednej výskumnej skupine (SETI@home), skupine výskumných skupín používajúcich spoločnú sadu aplikácií (Climateprediction.net, nanoHUB@home) alebo skupine spolu neprepojených vedcov (IBM World Community Grid, BOINC@TACC). BOINC poskytuje HTTP API na vzdialenú správu úloh. Projekty následne môžu jednoducho vytvárať a spravovať úlohy pomocou webové rozhrania (napr. vedecké brány) alebo prostredníctvom existujúcich systémov, napr. systém HTCondor.

Projekty môžu prevádzkovať verejnú webovú stránku. Serverový softvér BOINC poskytuje webové skripty pracujúce nad databázou, ktoré podporujú prihlásenie dobrovoľníka, zobrazenie nástenky so správami, profily, výsledkové tabuľky, odznaky a rôzne iné funkcie. Projekt je identifikovaný adresou URL svojej webovej stránky.

Dobrovoľník je vlastníkom výpočtového zariadenia (stolný počítač, notebook, tablet alebo smartfón), ktorým sa chce podieľať na dobrovoľných výpočtoch. Najskôr si dobrovoľník musí nainštalovať BIONC klienta na svoje zariadenie. Ďalej si musí vybrať projekty, ktoré chce podporiť, a vytvoriť si účet pre každý projekt. Následne pripojí BIONC klienta ku každému účtu. U každého projektu si taktiež môže zvoliť relatívne množstvo výpočtových zdrojov (*resource share*) [9], ktoré budú danému projektu pridelené.



Obrázok 2.2: Dobrovoľníci sa môžu pripojiť ku ktorejkoľvek skupine projektov a každému projektu môžu priradiť určité množstvo zdrojov [9].

2.1.2 Komunikácia medzi klientom a serverom

Po pripojení sa k projektu systém BOINC pravidelne odosiela príkazy RPC na server projektu, aby oznámil, ktoré úlohy boli dokončené a zároveň aby požiadal o nové úlohy. Klient následne stiahne aplikáciu spolu so vstupnými súbormi a začne vykonávať úlohu. Po jej dokončení nahrá výstupné súbory na server. Súbory nemusia byť nahrané len na server projektu, súbory môžu byť nahrané aj na iné servery. Celá komunikácia prebieha pomocou HTTP spojenia inicializovaného klientom.

2.1.3 Správcovia účtov

BOINC poskytuje rámec (*framework*) pre správcov webových účtov (AM). BOINC klient namiesto toho, aby bol explicitne pripojený k projektom, môže byť pripojený k správcovi účtu, ktorému periodicky posiela príkazy. Odpoveď obsahuje zoznam projektov a odpovedajúcich účtov, ku ktorým je klient pripojený. Klient sa následne pripojí k týmto projektom a odpojí sa od iných.

2.1.4 Nastavenia výpočtov

Dobrovoľníci môžu špecifikovať, ako sa majú využívať ich výpočtové zdroje. Nastavenia výpočtov je možné upraviť prostredníctvom webového rozhrania na projekte alebo v správcovi účtu. Tieto nastavenia sú následne propagované do všetkých počítačov, ktoré sú pripojené k tomuto účtu. Nastavenia možno tiež upraviť lokálne na počítači.

2.2 Architektúra serveru

Platforma BOINC prevádzkuje server pozostávajúci z jedného alebo viacerých serverov, na ktorý beží serverový softvér BOINC. Tieto systémy môžu bežať na vyhradenom hardvéri,

vo virtuálnom stroji alebo na cloudových uzloch. Ako je možné vidieť na obrázku 2.3, server obsahuje mnoho komunikujúcich procesov [2].

Architektonicky je významná väzba servera BOINC na relačnú databázu (zvyčajne MySQL alebo MariaDB). Databáza obsahuje tabuľky pre väčšinu vyššie uvedených abstrakcií: účty dobrovoľníkov, hostitelia, aplikácie, verzie aplikácií, úlohy, atď.

Príkazy RPC od klienta sú obsluhované plánovačom. Je nutné mat aktívnych niekoľko inštancií plánovača, keďže sa zvyčajne spracováva viacero príkazov naraz. Hlavnou funkciou plánovača je odoslanie inštancií úloh klientom.

Veľkou výhodou platformy BOINC je vysoká škálovateľnosť servera. Démoni môžu bežať na samostatných hostiteľoch, všetky serverové funkcie okrem databázového servera možno rozdeliť medzi viacero procesov, ktoré bežia na jednom alebo na rôznych hostiteľoch. Tieto vlastnosti spolu v kombinácii s rôznymi možnosťami škálovania databázových serverov umožňujú serverom BOINC zvládať pracovné zaťaženie rádovo v miliónoch úloh za deň. Serverový softvér BOINC obsahuje aj skripty a webové rozhrania na vytváranie projektov, zastavenie / spustenie projektov, aktualizáciu aplikácií, a pod.

Server sa skladá z mnohých procesov, väčšinou asynchrónnych s ohľadom na klientské požiadavky, ktoré komunikujú prostredníctvom databázy. Nevýhodou tohto prístupu je vysoké zaťaženie na databázovom serveri. Je možné si predstaviť alternatívny dizajn, v ktorom takmer všetky funkcie vykonáva plánovač, synchrónne s požiadavkami klienta. Ten prístup by mal nižšie režijné náklady na databázu. Na druhej strane, súčasný dizajn má niekoľko dôležitých výhod [2]:

- **Odolnosť voči poruchám**

  Ak databázu používa iba komponenta Assimilator a ak nie je databáza k dispozícii, zablokuje sa iba Assimilator. Ostatné komponenty naďalej pracujú a databáza sa správa ako fronta pre Assimilator. Práca pre Assimilator sa nahromadí v databáze a po oprave chyby môže byť zameškaná práca dokončená.

- **Odolnosť voči prepadu výkonu**

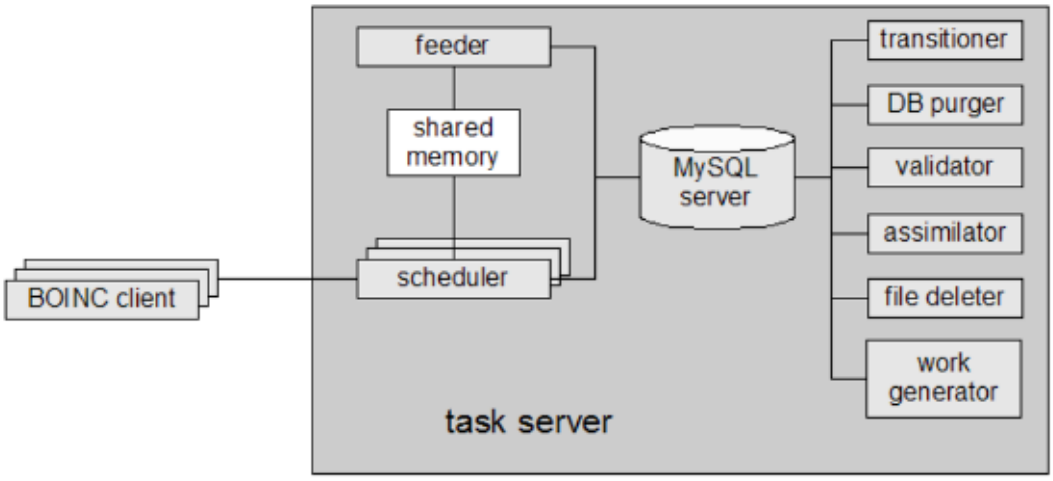
  Ak komponenty serverovej časti (napr. Validator alebo Assimilator) sú pomalé a zaostávajú, komponenty viditeľné klientom (Feeder a Scheduler) nie sú ovplyvnené.

- **Distribuovateľnosť**

  Rôzne komponenty sa dajú ľahko distribuovať a / alebo replikovať.

2.3 Podsystémy serveru

BOINC server sa skladá z viacerých podsystémov, ktoré sú znázornené na obrázku 2.3.



Obrázok 2.3: Komponenty BOINC servera [9].

Work generator

Jedná sa o generátor, ktorý vytvára nové úlohy a ich vstupné súbory. Generátor práce je nečinný, ak počet neodoslaných inštancií presahuje hraničnú hodnotu, čo obmedzuje množstvo potrebného ukladacieho priestoru na disku pre vstupné súbory.

Scheduler

Plánovač spracováva požiadavky od servera. Každá žiadosť obsahuje popis hostiteľa, zoznam dokončených úloh a žiadosť o ďalšiu prácu. Odpoveď obsahuje zoznam inštancií a k nim odpovedajúcich úloh. Spracovanie žiadosti zahŕňa množstvo databázových operácií: čítanie a aktualizácia záznamov o úlohe, účte, tíme a hostiteľovi. Plánovač je implementovaný ako CGI program spustený z webového servera Apache a mnoho inštancií môže bežať súčasne.

Validator

Komponenta Validator porovnáva inštancie úloh a vyberá kanonickú inštanciu predstavujúcu správny výstup. BOINC poskytuje validátor, ktorý porovnáva výstupné súbory bajt po bajte. Iné aplikácie môžu použiť vlastné validátory (validačné funkcie).

Assimilator

Komponenta Assimilator spracováva prácu, ktorá bola dokončená, čo znamená, že má buď kanonickú inštanciu alebo u nej nastala trvalá chyba. Spracovanie úspešne dokončenej úlohy môže zahŕňať zápis výstupov do databáze aplikácie alebo archiváciu výstupných súborov.

Transitioner

Komponenta Transitioner skúma úlohy, u ktorých došlo k zmene (napr. oznámená dokončená inštancia úlohy). V závislosti od situácie môže generovať nové inštancie, označiť úlohu ako trvale chybnú, spustiť validáciu alebo asimiláciu úlohy a iné akcie.

Feeder

Komponenta Feeder zjednodušuje prístup plánovača do databáze a zaisťuje výlučný prístup k zdieľaným zdrojom.

File deleter

Komponenta File deleter maže vstupné a výstupné súbory po dokončení úloh, ktoré už nie sú viac potrebné.

Database purger

Komponenta Database purge maže nepotrebné záznamy o dokončených úlohách z databázy.

2.4 Architektúra klienta

Klientsky softvér BOINC pozostáva z troch programov, ktoré komunikujú prostredníctvom RPC cez TCP spojenie:

1. **BOINC klient (BOINC Client)**

   BOINC klient riadi vykonávanie úloh a prenos súborov. Inštalátor zabezpečí klientovi spúšťanie pri spustení počítača. Exportuje sadu RPC príkazov do šetriča obrazovky a GUI.

2. **Grafické používateľské rozhranie (BOINC Manager)**

   Grafické používateľské rozhranie umožňuje dobrovoľníkom kontrolovať a monitorovať výpočet. Môžu napríklad pozastaviť / obnoviť výpočty a riadiť jednotlivé úlohy. Taktiež si môžu zobraziť aktuálny stav výpočtov, odhadovaný zostávajúci čas do dokončenia úloh a zoznam udalostí s internými a ladiacimi informáciami.

3. **Šetrič obrazovky**

   Voliteľný šetrič obrazovky zobrazuje v pohotovostnom režime počítača aplikačnú grafiku na vykonávanie úloh.

Vývojári tretích strán implementovali rôzne grafické používateľské rozhrania. BoincTasks je grafické rozhranie pre Windows, pomocou ktorého je možné spravovať viacerých klientov. Pre hostiteľov bez displeja BOINC dodáva terminálový program poskytujúci rovnaké funkcie ako BIONC Manager.

BOINC klient je viacvláknový. Každé RPC pripojenie s GUI je obsluhované samostatným vláknom. Všetky vlákna sa synchronizujú pomocou jedného zámku. Hlavné vlákno uvoľní tento zámok pri čakaní alebo pri vykonávaní operácií so súbormi, čo zaisťuje rýchlu odozvu na RPC príkazy pre GUI [9].

Kapitola 3

Systém Fitcrack

Fitcrack je distribuovaný systém používaný na obnovu hesiel z rôznych typov šifrovaných súborov vrátane dokumentov, archívov, diskových zväzkov alebo kryptografických hešov [18]. Systém Fitcrack bol vytvorený v rámci výskumnej skupiny NES@FIT[1] na Fakulte informačných technológií VUT v Brne. Predstavuje experimentálny prototyp, ktorý sa primárne používa na výskum a testovanie nových techník na obnovu hesiel. Systém Fitcrack je voľne dostupný pod MIT licenciou v repozitári[2] na platforme Github.

Fitcrack vďaka integrácii s nástrojom Hashcat podporuje širokú škálu formátov a proces obnovy hesiel je vysoko efektívny [15]. Výpočet je hardvérovo akcelerovaný pomocou technológie OpenCL, vďaka čomu môže obnova hesiel prebiehať na všetkých procesoroch kompatibilných s OpenCL. Fitcrack je založený na systéme BOINC, ktorý slúži na distribuované výpočty. Využíva adaptívny plánovací podsystém pre najefektívnejšie využitie zdrojov distribuovanej siete. Výpočtové uzly sa môžu počas výpočtového procesu pripojiť alebo odpojiť.

3.1 Lámanie hesiel

Lámanie hesiel je proces obnovy hesiel z dát, ktoré boli uložené alebo prenesené počítačovým systémom. Pri lámaní sa snažíme uhádnuť heslo a následne ho skontrolovať oproti dostupnému kryptografickému hešu hesla. Systém Fitcrack používa nástroj hashcat pre lámanie hesiel, ktorý sa spúšťa na jednotlivých klientoch.

Hashcat je najrýchlejší a najpokročilejší nástroj na obnovu hesiel na svete [26], ktorý na výpočty využíva technológiu OpenCL. OpenCL špecifikuje programovacie jazyky založené na jazykoch C99 a C++11 na programovanie zariadení podporujúcich OpenCL a poskytuje aplikačné programovacie rozhrania na ovládanie platformy a vykonávanie programov na výpočtových zariadeniach. Programy využívajú štandardné rozhranie pre paralelné výpočty pomocou paralelizmu založeného na úlohách a dátach.

Hashcat podporuje vyše 200 vysoko optimalizovaných hešovacích algoritmov, päť rôznych typov útokov a niekoľko druhov zariadení podporujúcich OpenCL. Od roku 2015 je nástroj voľne dostupný[3] pod MIT licenciou. Hashcat podporuje operačné systémy Linux, macOS a Windows.

[1] https://www.fit.vut.cz/research/group/nes@fit/.cs
[2] https://github.com/nesfit/fitcrack
[3] https://github.com/hashcat/hashcat

3.1.1 Typy útokov

Nástroj hashcat podporuje nasledovné typy útokov, ktoré sú taktiež integrované do systému Fitcrack:

- **Slovníkový útok**

  Slovníkový útok používa textový súbor nazývaný slovník, v ktorom sú uložené jednotlivé heslá. Každé heslo je uvedené na jednom riadku. Pri slovníkovom útoku nástroj hashcat postupne číta jednotlivé riadky, z hesiel počíta kryptografické heše a porovnáva ich s hľadanými hešmi. Matematicky je možné chápať slovník s heslami ako usporiadanú neprázdnu konečnú množinu D, kde usporiadanie je dané poradím hesiel v slovníku. Celkový počet kandidátnych hesiel p je rovný mohutnosti tejto množiny:

  $$p = |D| \tag{3.1}$$

- **Kombinačný útok**

  Kombinačný útok používa dva slovníky s heslami, ktoré sa označujú ako ľavý a pravý slovník. Kandidátne heslá, ktoré sa budú skúšať, vznikajú konkatenáciou hesiel z ľavého a pravého slovníka. Nech D_1 je ľavý slovník a D_2 pravý slovník. Celkový počet kandidátnych hesiel p môže byť vypočítaný nasledovne ako súčin mohutností oboch množín:

  $$p = |D_1| * |D_2| \tag{3.2}$$

- **Útok hrubou silou**

  Útok hrubou silou spočíva vo vytvorení všetkých permutácií danej dĺžky nad danou abecedou, resp. množinou znakov. Nástroj hashcat však tento koncept rozširuje zavedením podpory pre masky. Maska umožňuje špecifikovať podobu generovaných hesiel. Upresňuje, ktoré znaky sa môžu vyskytovať na ktorých pozíciách. Maska je definovaná ako postupnosť zástupných symbolov. Nech $G_1, G_2, \ldots, G_n$ sú špecifické množiny symbolov reprezentované jednotlivými zástupnými symbolmi v maske. Príkladom špecifických množín symbolov sú číslice, malé písmená anglickej abecedy alebo ASCII znaky. Celkový počet kandidátnych hesiel p je rovný súčinu mohutností týchto množín:

  $$p = \prod_{i=1}^{n} |G_i| \tag{3.3}$$

- **Hybridný útok**

  Hybridné útoky predstavujú kombináciu prístupov slovníkového útoku a útoku hrubou silou s využitím masky. Podobne ako u kombinačného útoku dochádza k vytváraniu kandidátnych hesiel konkatenáciou dvoch oddelených častí: ľavej a pravej. Jedna z nich je tvorená heslami zo slovníka, druhá je generovaná na základe masky. Podľa spôsobu generovania časti kandidátneho hesla rozlišujeme dva podtypy hybridných útokov:

  – **Hybridný útok s použitím slovníku a masky**
  Ľavá časť hesla sa generuje zo slovníka a pravá z masky.

– **Hybridný útok s použitím masky a slovníku**
Ľavá časť hesla sa generuje z masky a pravá zo slovníka.

Kandidátne heslá vznikajú konkatenáciou týchto dvoch častí. Celkový počet kandidátnych hesiel p je možné vypočítať nasledovne:

$$p = |D| * \prod_{i=1}^{n} |G_i| \tag{3.4}$$

kde D je použitý slovník a G_i je špecifická množina symbolov reprezentovaná v poradí i-tým zástupným symbolom v maske.

3.1.2 Metriky hodnotenia útokov

Pomocou metrík je možné hodnotiť a porovnávať útoky medzi sebou. Systém Fitcrack pracuje s nasledovnými metrikami:

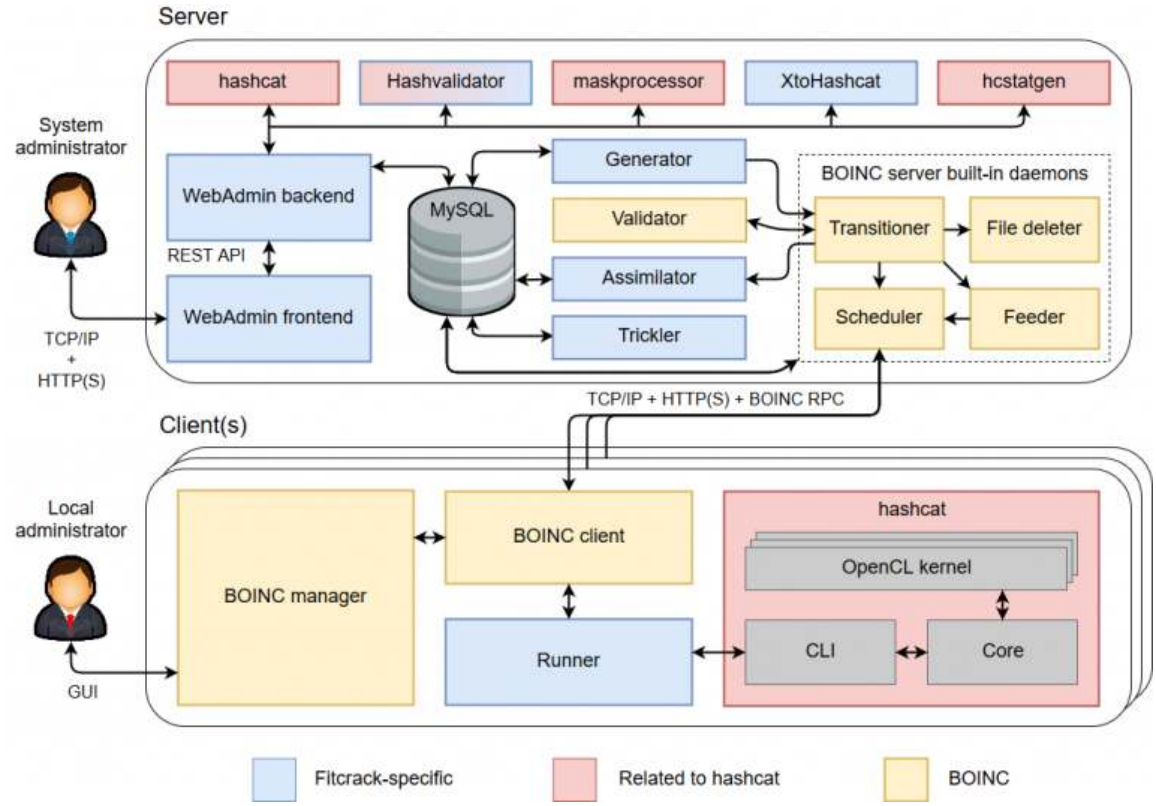
- **Doba réžie**
  Doba, ktorá zahŕňa čas potrebný na pripravenie a distribúciu útoku po sieti.
- **Doba lámania hesiel**
  Doba, počas ktorej sa lámu kandidátne heslá.
- **Doba výpočtu**
  Doba výpočtu úlohy s útokom je daná ako súčet doby réžie s dobou lámania hesiel.
- **Efektivita**
  Efektivita predstavuje percento času, kedy procesory počítali úlohu, a zároveň sa do tohto času nezapočítava čas pre komunikáciu alebo doba, počas ktorej boli procesory v nečinnosti [21]. Vypočíta sa pomocou vzorca:

$$p = \frac{\sum_{i=1}^{N} t_x}{N * T_{fin}} \tag{3.5}$$

kde N je počet klientov, ktorí sa podieľali na výpočte úlohy, t_x je doba, počas ktorej klienti počítali úlohu a T_{fin} je celkové trvanie úlohy od jej spustenia až po jej ukončenie.

3.2 Architektúra systému

Fitcrack sa radí medzi systémy, ktoré využívajú architektúru klient-server. Server a klienti sú vzájomne prepojení sieťou TCP/IP, čo umožňuje spustiť úlohy na obnovu hesiel cez Internet na uzloch, ktoré sa nachádzajú na geograficky vzdialených miestach. Server slúži na správu lámacích úloh, klienti majú za úlohu vykonávať samotný proces lámania hesiel. Klienti komunikujú so serverom pomocou plánovacieho serverového protokolu založeného na RPC cez HTTPS [1]. Súčasná architektúra systému Fitcrack je znázornená na obrázku 3.1. Každá strana pozostáva z viacerých podsystémov, ktorých činnosť bude ozrejmená v ďalších podkapitolách.



Obrázok 3.1: Architektúra systému Fitcrack [18].

3.3 Architektúra servera

Fitcrack server je zodpovedný za správu lámacích úloh a za prideľovanie úloh klientom. Na obrázku 3.1 je v hornej časti znázornená architektúra servera. Na rozdiel od Fitcrack klienta, ktorý podporuje operačné systémy Linux a Windows, server v súčasnej dobe podporuje iba operačný systém Linux.

3.4 Podsystémy servera

Fitcrack server sa skladá z viacerých podsystémov:

Webadmin frontend

Webový klient (frontend) je vytvorený vo frameworku Vue.js a umožňuje správcovi spravovať rôzne časti systému. Na karte *Jobs* môže správca pridávať, upravovať a spravovať všetky úlohy. Karta *Hosts* poskytuje prehľad nad pripojenými klientov. Každý heš je možné zobraziť v súhrne na karte *Hashes*. Karta *Dictionaries* slúži na správu a pridávanie slovníkov s heslami. Prehľadné nástenky (*dashboards*) umožňujú jednoduchú analýzu dát uložených v databáze systému Fitcrack [25].

Vue.js

Vue je progresívny framework na tvorbu používateľských rozhraní [32]. Je jedným z moderných frameworkov, ktorý v sebe kombinuje technológie a koncepty pôvodne predstavené

v frameworkoch AngularJS a React, ale zároveň snaží vyhýbať nedostatkom, ktoré tieto dva frameworky majú.

Na rozdiel od iných monolitických štruktúr je Vue navrhnutý od základov tak, aby bol postupne prispôsobiteľný. Základná knižnica je zameraná iba na zobrazovaciu vrstvu a je ľahké ju použiť s inými knižnicami alebo integrovať do existujúcich projektov. Vue sa zameriava na abstrakciu reaktivity a umožňuje automaticky aktualizovať zobrazenie pri zmene dát. Funkcionalita a vzhľad sa integrujú do komponent, ktoré možno skladať do väčších celkov, a tak vytvárať webovú aplikáciu.

Webadmin backend

Serverová časť (backend) je napísaná v jazyku Python 3 a je založená na technológii Flask. Komunikácia so serverom prebieha pomocou rozhrania WSGI. Server implementuje všetky potrebné koncové body REST API, ktoré používa webový klient. Serverová časť pomocou technológie SQLAlchemy prevádzkuje databázu MySQL, ktorá slúži ako úložisko pre všetky dáta súvisiace s lámaním.

Popis použitých technológií:

- **Flask**

  Flask [11] je webový framework napísaný v jazyku Python. Bol vyvinutý Arminom Ronacherom, ktorý viedol tím medzinárodných nadšencov jazyku Python s názvom Poocco. Framework je založený na sade nástrojov Werkzeug WSGI a na technológii Jinja2.

- **WSGI**

  WSGI (*Web Server Gateway Interface*) je špecifikácia spoločného rozhrania medzi webovými servermi a webovými aplikáciami.

- **Jinja2**

  Systém Jinja2[4] je populárny systém pracujúci so šablónami pre jazyk Python. Kombinuje šablónu so špecifickým zdrojom dát a vykresľuje dynamickú webovú stránku.

- **REST**

  REST (*Representational State Transfer*) [10] je štýl architektúry softvéru, ktorý definuje sadu obmedzení, ktoré sa majú použiť pre vytváranie webových služieb. Webové služby, ktoré zodpovedajú architektonickému štýlu REST, poskytujú interoperabilitu medzi počítačovými systémami na internete. RESTful webové služby umožňujú požadujúcim systémom prístup a manipuláciu s textovými reprezentáciami webových zdrojov pomocou jednotnej a preddefinované sady bezstavových operácií. RESTful systémy využívajú bezstavový protokol a štandardné operácie, ktorých cieľom je poskytnutie vysokého výkonu, spoľahlivosti a schopnosti rastu opakovaným používaním súčastí, ktoré možno spravovať a aktualizovať bez toho, aby to ovplyvnilo systém ako celok.

- **SQLAlchemy**

[4] https://jinja.palletsprojects.com/

SQLAlchemy[5] je knižnica, ktorá uľahčuje komunikáciu medzi programami v jazyku Python a databázami. Najčastejšie sa táto knižnica používa ako nástroj pre objektovo relačné mapovanie (ORM), ktorý prekladá triedy jazyka Python do tabuliek v relačných databázach a automaticky prevádza volania funkcií na príkazy SQL. SQLAlchemy poskytuje štandardné rozhranie, ktoré vývojárom umožňuje vytvárať databázovo agnostický kód na komunikáciu s celým radom databázových systémov.

Hashvalidator

Nástroj, ktorý kontroluje správnu syntax vstupných hešov.

maskprocessor

Program, ktorý slúži na generovanie slovníkov z masiek v hybridných útokoch.

XtoHashcat

Nástroj, ktorý dokáže automaticky zistiť formát vstupného šifrovaného média a extrahovať z neho heš.

Generator

Generátor je serverový démon zodpovedný za vytváranie nových pracovných jednotiek pre hostiteľov. Existujú dva typy pracovných jednotiek, ktoré generátor vytvára: benchmark a bežná úloha.

- **Benchmark**

  Benchmark slúži na zmeranie výkonu výpočtového klienta. Po prvotnom pripojení klienta do systému generátor pre neho vytvorí kompletný benchmark, ktorý zmeria výkon klienta pri lámaní všetkých podporovaných typov heší. Tento výsledok je následne uložený do databázy a je ďalej využívaný pre odhady času trvania úloh.

- **Bežná úloha**

  Pracovná jednotka s bežnou úlohou obsahuje informácie o pridelenej práci pre klienta. Podľa typu útoku generátor pripraví a vytvorí všetky potrebné súbory pre klienta. Následne sú tieto súbory odoslané klientovi. Klient po ich prijatí môže začať pracovať na časti úlohy, ktorá mu bola pridelená. Veľkosť pracovnej jednotky, tj. počet skúšaných hesiel, sa určuje podľa výkonu klienta.

Generátor komunikuje so zvyškom servera iba pomocou databázy. Generátor tiež vytvára vstupné súbory, ktoré sa odosielajú hostiteľom. Počet týchto súborov sa líši v závislosti od typu útoku. Vždy sa odosielajú dátové a konfiguračné súbory obsahujúce vstupné dáta s hešmi a potrebné metadáta.

Validator

Fitcrack používa predvolený BOINC validátor, ktorý kontroluje iba syntax výsledku.

[5] https://www.sqlalchemy.org/

Assimilator

Assimilator je serverový démon, ktorý analyzuje výsledky poskytnuté hostiteľmi. V závislosti od typu výsledku je schopný upravovať databázu alebo rušiť bežiace pracovné jednotky.

Trickler

Komponenta Trickler bola vytvorená na výmenu informácií medzi serverom a klientmi počas lámania hesiel. Používa sa na pravidelné odosielanie správ o stave spracovania pracovnej jednotky.

Transitioner

Transitioner je predvolený démon, ktorý udržuje databázu synchronizovanú. Systém Fitcrack používa predvolenú implementáciu zo systému BOINC bez akýchkoľvek úprav.

Scheduler

Scheduler je zodpovedný za komunikáciu s hostiteľmi. Aj keď systém Fitcrack používa predvolenú implementáciu plánovača, vykonali sa určité úpravy, aby bolo možné zistiť, ktorí hostitelia sú aktívni a bežia.

Feeder

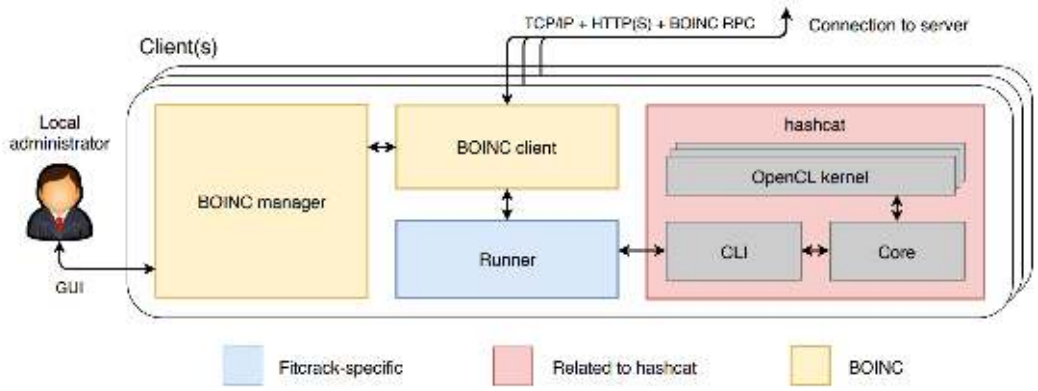
Systém Fitcrack používa predvolenú implementáciu démona Feeder. Úzko spolupracuje s plánovačom a je zodpovedný za distribúciu častí zdieľanej pamäte.

File deleter

Systém Fitcrack používa predvolenú implementáciu démona File deleter. Jeho zodpovednosť spočíva v mazaní vstupných a výstupných súborov u dokončených úloh.

3.5 Architektúra klienta

Klienti predstavujú uzly, na ktorých prebieha proces lámania hesiel. Fitcrack klient je možné spustiť na akomkoľvek počítači so systémom Windows alebo Linux s najmenej jedným zariadením kompatibilným s OpenCL a s nainštalovanými správnymi ovládačmi. Jediný softvér, ktorý je potrebné nainštalovať, je BOINC klient a voliteľne aj program BOINC Manager poskytujúci grafické užívateľské rozhranie pre BOINC klienta. Po pripojení BOINC klientov k serveru sa automaticky stiahnu všetky potrebné binárne súbory. Binárne súbory zahŕňajú dve aplikácie: Hashcat ako nástroj na lámanie hesiel a program Runner. Architektúra Fitcrack klienta je znázornená na obrázku 3.2.



Obrázok 3.2: Architektúra klienta [18].

3.5.1 Runner

Runner je program zapuzdrujúci nástroj hashcat, ktorý bol navrhnutý na použitie buď ako samostatný nástroj, ktorý zjednodušuje ovládanie nástroja hashcat, alebo ako middleware program v systéme BOINC. Program Runner napísaný v jazyku C++98. Kompiluje sa do statického binárneho súboru pre operačné systémy Linux a Windows.
Runner po spustení BOINC klientom vykonáva nasledovné operácie:

1. prečíta súbor `config` a na základe neho vytvorí argumenty pre hashcat,
2. spustí hashcat s vytvorenými argumenty,
3. sleduje proces lámania,
4. zbiera výsledky a vytvára výstupný súbor, ktorý je odoslaný do systému BOINC.

Všetky informácie potrebné pre aplikáciu sú uložené v niekoľkých súboroch, ktoré musia byť v rovnakom adresári ako spustiteľný súbor.

3.6 Iné systémy na obnovu hesiel

Okrem systému Fitcrack existuje niekoľko ďalších systémov na obnovu hesiel. Najpopulárnejším z nich je systém Hashtopolis, ktorý je hlavnou konkurenciou pre systém Fitcrack.

3.6.1 Hashview

Hashview[6] je nástroj pre odborníkov v oblasti bezpečnosti, ktorý im pomáha organizovať a automatizovať opakujúce sa úlohy súvisiace s obnovou hesiel. Jedná sa o webovú aplikáciu, cez ktorú je možné spravovať nástroj hashcat. Primárne využitie spočíva v automatizácii úloh a získavania štatistík v reálnom čase. Od verzie 0.7.1 podporuje distribuovanú obnovu hesiel.

[6] https://github.com/hashview/hashview

3.6.2 Cracklord

Cracklord[7] je systém, ktorý bol navrhnutý s hlavným cieľom, aby poskytoval škálovateľný, distribuovaný systém pre obnovu hesiel, ako aj pre všetky ďalšie úlohy vyžadujúce veľa výpočtových zdrojov [20]. CrackLord obsahuje mechanizmus, ktorý umožňuje vyvážiť záťaž z viacerých hardvérových systémov do jedinej fronty v rámci dvoch primárnych služieb: *Resource* a *Queue*. Tieto služby úlohy nezrýchlia, ale uľahčia ich správu. Systém sa zameriava na tri primárne prípady použitia:

- **Skupinová správa úloh**

  Či už sa jedná o skupinu študentov, penetračných testerov, auditorov či administrátorov, ich priamy prístup k zdrojom nie je udržateľné riešenie z dlhodobého hľadiska. Používatelia pristupujú pomocou SSH na servery a ovládajú systémy. Na serveroch môže vzniknúť situácia, že iný používateľ náhodne preruší ich úlohy, čím môže znehodnotiť ich doterajšiu prácu.

- **Centralizovaný prístup k distribuovaným zdrojom**

  Ďalším cieľom systému je poskytnúť možnosť umiestnenia úloh na jedno miesto a umožniť paralelný beh viacerých úloh ich distribúciou medzi výkonné servery. Alternatívne by bolo možné vytvoriť výpočtový cluster, avšak toto riešenie sa v systéme Cracklord nevyužíva, a pre paralelný beh úloh je vhodnejšie použitie viacerých serverov.

- **Prístup k zdrojom v cloude**

  Cracklord umožňuje pridávanie cloudových výpočtových zariadení do systému, keďže nie každá organizácia či skupina ľudí má prístup k výkonnému hardvéru, no zároveň je tento hardvér pre nich potrebný na rôzne výpočty.

3.6.3 Hashtopolis

Hashtopolis je multiplatformový nástroj s architektúrou klient-server na distribúciu úloh medzi viaceré počítače [7]. Systém je voľné dostupný v repozitári[8] na platforme Github.

3.6.4 Porovnanie systémov Fitcrack a Hashtopolis

Hashtopolis a Fitcrack sú podobné systémy, ktoré sa zameriavajú na distribuované lámanie hesiel. Obsahom tejto kapitoly je analýza oboch systémov a popis ich vlastností v rôznych skúmaných kategóriách.

Klientská časť:

- **Fitcrack**

  Na klientský počítač s OS Windows alebo Linux je potrebné stiahnuť a nainštalovať program BOINC Manager, aby sa klient mohol zapojiť do distribuovaného lámania v systéme Fitcrack. Následne je nutné v programe BOINC Manager pripojiť sa k adrese URL projektu a autentizovať sa pomocou používateľského mena a hesla. Konfigurácia pre projekt Fitcrack je automaticky načítaná. Automaticky sa stiahnu

[7] https://github.com/jmmcatee/cracklord
[8] https://github.com/s3inlc/hashtopolis

všetky potrebné súbory. Vďaka použitiu platformy BOINC je možné podrobne nastaviť, ako používateľ chce používať počítač na výpočty. V nastaveniach môže určiť, koľko percent CPU / RAM / kapacity disku chce vyhradiť pre účely výpočtov. Systém Fitcrack taktiež umožňuje používateľovi pomocou voliteľného konfiguračného súboru určiť, ktoré GPU sa majú použiť na výpočty a s akou záťažou (tzv. *workload profile*). Pomocou tohto súboru taktiež môže špecifikovať ďalšie argumenty, s ktorými bude nástroj hashcat spustený pri každej úlohe na danom klientovi.

- **Hashtopolis**

  Na klientský počítač je nutné stiahnuť a nainštalovať program Hashtopolis Agent, ktorý existuje v dvoch implementáciách - v jazyku Python je dostupný pre OS Windows a Linux (nová, odporúčaná verzia agenta) a v jazyku C# pre OS Windows. Klienta je potrebné zaregistrovať pomocou unikátneho kľúča (tzv. *vouncher*), ktorý je generovaný na serveri. Pomocou tohto kľúča sa klient autentizuje voči serveru. Všetko ostatné sa automaticky stiahne a nakonfiguruje.

Serverová časť:

- **Fitcrack**

  Fitcrack server sa skladá z viacerých aktívne bežiacich služieb (démonov), databáze MySQL a aplikácie na správu úloh WebAdmin, ktorá sa skladá z dvoch častí. Prvá časť je serverová, napísaná v jazyku Python, ktorá využíva technológiu SQLAlchemy. Druhou časťou je webový klient, tj. webové rozhranie vytvorené pomocou Vue.js. Webové rozhranie nie je jediná možnosť ako pracovať s celým systémom. Ďalšou možnosťou je použitie aplikačného rozhrania Fitcrack API, ktoré rozširuje možnosti práce so systémom, a môže byť využité napríklad na automatizáciu akcií s úlohami.

- **Hashtopolis**

  Architektúra servera je monolotická a nedelí sa na serverovú časť a na webového klienta. Server je tvorený jedinou aplikáciou napísanou v jazyku PHP. Všetky akcie sa vykonávajú na základe požiadavky od používateľa či klienta, ktorý môže ovládať Hashtopolis len pomocou webovej aplikácie, keďže použitá architektúra a chýbajúce aplikačné rozhranie neumožňujú ovládanie systému iným programom. Hashtopolis podobne ako systém Fitcrack využíva databázu MySQL na ukladanie dát.

Práca so systémom:

- **Fitcrack**

  Na začiatku vytvárania úlohy si používateľ vyberá typ hešu a následne zadáva do systému heše, ktoré chce prelomiť. Fitcrack podporuje viacero spôsobov, ako vložiť heše. Prvou možnosťou je ich vloženie do textové poľa, druhou je možnosť nahrania súboru s hešmi do systému a systém si zo súboru načíta heše. Tretou možnosťou je získanie hešu jeho extrahovaním z používateľom nahraných dokumentov MS Office, PDF alebo archívov RAR, ZIP a 7z. Následne si používateľ vyberá typ útoku. Fitcrack podporuje 6 typov útokov a pre každý útok existuje špeciálna karta vo webovom rozhraní, kde si môže používateľ nastaviť daný útok podľa svojich potrieb. V ďalšej fáze si používateľ vyberá klientov, ktorí majú danú úlohu počítať. V poslednom kroku si môže používateľ nastaviť veľkosť pracovnej jednotky definovaním maximálneho času

vyhradeného na jednu pracovnú jednotku a taktiež môže naplánovať, kedy sa má úloha spustiť alebo ukončiť.

Odhad očakávanej doby trvania úlohy sa zobrazuje už pri tvorbe úlohy ak užívateľ zadá aspoň typ hešu a útoku. Počas počítania úlohy tento odhad priebežne aktualizuje a spresňuje. Jeden klient môže byť priradený aj k viacerým úlohám, a teda bude pracovať na viacerých úlohách súčasne. V takomto prípade dochádza k striedavému prideľovaniu pracovných jednotiek od rôznych úloh.

U každého útoku je zobrazená história dosiahnutých stavov úlohy. Z histórie je možné napríklad zistiť, kedy bola úloha spustená, pozastavená, dokončená, a pod. Zmena stavu úlohy je taktiež oznámená pomocou notifikačnej správy.

Fitcrack obsahuje unikátne funkcie, ktoré nie sú obsiahnuté v systéme Hashtopolis a ani v žiadnom inom. Medzi tieto funkcie patrí správa Markovových reťazcov a ich automatická tvorba z existujúceho slovníka. Podobne u PCFG je možné zobrazovať, spravovať a vytvárať nové gramatiky z už existujúceho slovníka. Systém taktiež umožňuje jednoduchú správu masiek, slovníkov a pravidiel. Ich obsah si používateľ môže zobraziť.

- **Hashtopolis**

  Hashtopolis neumožňuje žiadnu z foriem zadania hešov počas tvorby úlohy. Používateľ je nútený pred vytvorím úlohy v časti *Hashlists* vložiť heše do textového poľa alebo nahrať súbor s hešmi. Následne si pri tvorbe úlohy používateľ vyberie pridaný zoznam hešov. Automatická extrakcia hešov z dokumentov / archívov nie je podporovaná.

  Na rozdiel od systému Fitcrack nerozlišuje konkrétne typy útokov. Používateľ u každej úlohy musí vyplniť textové pole „Attack command“, kde zadá spúšťacie argumenty pre nástroj hashcat. Ďalej vyberie súbory, ktoré sa majú poslať klientovi. Používateľ si taktiež môže vybrať externý generátor hesiel. V súčasnosti je implementovaná podpora pre generátor princeprocessor.

  Pri tvorbe úlohe systém nezobrazuje používateľovi predpokladaný čas trvania úlohy. Táto informácia je dostupná až po spustení úlohy a priebežne sa aktualizuje. Na rozdiel od systému Fitcrack avšak umožňuje použitie viacerých verzií nástroja hashcat a používateľ si môže vybrať špecifickú verziu nástroja pre danú úlohu. Obmedzením u tohto systému je, že jeden klient môže byť v jeden moment priradený k maximálne jednej úlohe a teda nemôže pracovať na viacerých úlohách, ako je tomu u systému Fitcrack. U klientov je možné nastaviť, či sa jedná je dôveryhodného alebo nedôveryhodného klienta. Nedôveryhodní klienti nebudú dostávať heše alebo slovníky, ktoré boli označené ako dôveryhodné.

  Používateľ si môže nastaviť chovanie systému v prípade výskytu chyby u konkrétneho klienta:

  - deaktivácia klienta
  - klient pracuje ďalej a ukladá chyby
  - klient pracuje ďalej a zahadzuje chyby

  Hashtopolis taktiež podporuje systém notifikácií. Na rozdiel od systému Fitcrack prináša možnosť zasielania upozornení na e-mailovú adresu, Discord a Slack.

Plánovanie a distribúcia úloh:

- **Fitcrack**

  Používateľ nastavuje čas pre pracovnú jednotku, čo je požadovaný čas, koľko by mal trvať výpočet jednej pracovnej jednotky na jednom klientovi. Pre detekciu výkonu klienta sa používa benchmark, u ktorého klient dostane úlohu a počíta ju niekoľko sekúnd. Na základe výstupu nástroja hashcat sa zistí ako rýchlo dokáže klient lámať danú úlohu. Fitcrack implementuje adaptívny plánovací algoritmus, ktorý vytvára každú pracovnú jednotku na mieru konkrétnemu klientovi vzhľadom na jeho momentálny výkon a celkový stav výpočtu úlohy. Parametre algoritmu si môže používateľ prispôsobiť v nastaveniach systému.

  Algoritmus zohľadňuje nasledujúce aspekty:

  - Výkon uzla sa môže meniť a systém sa tomu prispôsobí.
  - Algoritmus počíta s možným dodatočným pripojením ďalších klientov za behu.
  - Na začiatku úlohy sa tvoria menšie pracovné jednotky, aby sa odstránili možné nepresnosti pri benchmarku a systém sa postupne stabilizoval.
  - Ku koncu úlohy sa opäť veľkosť pracovných jednotiek zmenšuje, aby došlo k maximálnemu využitiu výpočtových zdrojov. Cieľom je vyhnúť sa situácii, kedy úlohu počíta len časť klientov a ostatní klienti sú bez práce.

  Pre každý typ útoku sa využíva unikátna stratégia výpočtu a distribúcie úloh s daným typom útoku. Cieľom je dosiahnutie vysokej efektivity lámania hesiel vďaka optimálnemu využitiu dostupného hardvéru, ktorý sa podieľa na lámaní. U slovníkového útoku dochádza k fragmentácii slovníka, u hybridných útokov sa môže rozgenerovať časť masky, a pod.

  Fitcrack zohľadňuje aj interné optimalizácie v nástroji hashcat, kedy vypočítaný celkový počet kandidátnych hesiel nemusí zodpovedať reálnemu počtu hesiel. Reálny počet možných hesiel Fitcrack počíta sám, informuje o ňom používateľa a využíva ho k efektívnejšiemu plánovaniu výpočtu. Systém tiež zohľadňuje využitie kryptografické soli, čo predstavuje nutnosť prepočítavať algoritmus pre každý heš.

- **Hashtopolis**

  Podobne ako u systému Fitcrack sa pre detekciu výkonu klienta používa benchmark. Používateľ si môže vybrať medzi dvoma typmi benchmarkov: prvý typ nazývaný *Speed test* je založený na prepínači `--speed-only` u nástroja hashcat. Druhý typ benchmarku sa označuje ako *Runtime benchmark*, v ktorom sa úloha spustí na krátky čas na danom klientovi, a z výstupu nástroja hashcat sa získa rýchlosť lámania, tj. výkonnosť klienta.

  Používateľ si aj v systéme Hashtopolis môže zvoliť veľkosť pracovnej jednotiek, namiesto pojmu *workunit* sa avšak používa pojem *chunk*. Hashtopolis nerozlišuje medzi typmi útokov, na každý využíva rovnakú stratégiu distribúcie založenú na použití prepínačov `--skip` a `--limit` u nástroja hashcat. Z tohto dôvodu je možnosť zlepšenia výkonu / distribúcie útokov nízka, keďže nie je možné využiť znalosti o type útoku na vytvorenie špecifickejších a optimalnejších algoritmov distribúcie úloh pre daný typ útoku. Výpočet celkového počtu kandidátnych hesiel deleguje nástroju hashcat.

Podpora útoku PRINCE:

- **Fitcrack**

  Systém Fitcrack v súčasnosti neobsahuje útok PRINCE a cieľom práce je integrácia tohto útoku do systému.

- **Hashtopolis**

  Hashtopolis podporuje distribuovaný útok PRINCE. Implementácia útoku na strane klienta je založená na prepojení externého generátora - princeprocessor, ktorý generuje heslá pomocou algoritmu PRINCE, s nástrojom hashcat pomocou rúr.

  Distribúcia útoku medzi klientov je založená na použití prepínačov `--skip` a `--limit` u externého generátora hesiel. Z pohľadu používateľa môže byť problémom konfigurácia útoku PRINCE v tomto systéme. Namiesto jednoduchého a ľahko použiteľného používateľského rozhrania je možné konfigurovať útok len pomocou ručného zadania argumentov, s ktorými sa má spustiť externý generátor hesiel.

Obrázok 3.3: Ukážka konfigurácie útoku PRINCE v systéme Hashtopolis.

Na obrázku 3.3 je zobrazená ukážka konfigurácie útoku PRINCE v systéme Hashtopolis. Používateľ musí špecifikovať názov vstupného slovníka a zadať argumenty pre externý generátor. Preklep či iná chyba nie je pri tejto forme konfigurácie útoku nereálna a systém v prípade problému nezobrazuje detailnejší popis problému. Na druhej strane, toto riešenie ponúka používateľovi väčšiu kontrolu nad externým generátorom a používateľ môže jednoducho vyskúšať aj neštandardné či experimentálne konfiguračné možnosti nástroja princeprocessor.

Distribuovaný slovníkový útok:

Autori systému Fitcrack publikovali v technickej správe [17] prípadovú štúdiu, kde porovnávali Fitcrack so systémom Hashtopolis na distribuovanom slovníkovom útoku. Riešenie distribúcie slovníkového útoku v systéme Fitcrack je oproti systému Hashtopolis iné a náročnejšie na implementáciu, no na experimentoch vrámci tejto štúdie sa ukázalo ako oveľa efektívnejšie a rýchlejšie.

Vďaka fragmentácii slovníkov dochádza u Fitcracku k odstráneniu počiatočnej réžie nutnej na prenos kompletných slovníkov. Najmä u reťazového spracovania rastie čas prakticky lineárne s veľkosťou slovníka. U systému Hashtopolis dochádza k zbytočnému prenosu celého slovníka všetkým klientom.

Kapitola 4

Algoritmus PRINCE

PRINCE (*PRobability INfinite Chained Elements*) je moderný algoritmus na generovanie hesiel, ktorý je možné použiť na pokročilé kombinačné útoky [29]. Algoritmus bol predstavený [28] na konferencii *Passwords14*. Jens Steube navrhol tento algoritmus tak, aby namiesto dvoch rôznych slovníkov a následného generovania všetkých možných kombinácii slov PRINCE používal iba jeden slovník a z neho vytváral reťazce (*chains*) kombinovaných slov. Tieto reťazce môžu obsahovať jedno až N slov pochádzajúcich zo vstupného slovníka, ktoré sú vzájomné pospájané. Slovo „Infinite“ v názve upozorňuje na fakt, že algoritmus PRINCE beží až dokým nevyčerpá množinu možných hesiel, k čomu dochádza po veľmi dlhej dobe [27]. Referenčná implementácia algoritmu PRINCE je voľné dostupná[1] pod MIT licenciou. Aktuálna implementácia[2] je jediným zdrojom informácii o fungovaní algoritmu, keďže jeho formálny popis neexistuje.

4.1 Základné komponenty algoritmu

Algoritmus PRINCE sa skladá z nasledovných komponent: prvky, reťazce a množina možných hesiel [6].

Prvok (element)

Prvok je najmenšia entita, ktorá reprezentuje neupravenú položku (riadok, slovo) vstupného slovníka. Všetky prvky sú zoradené podľa relevantnosti / frekvencie a zoskupené podľa dĺžky do databázy prvkov. V tabuľke 4.1 sú uvedené príklady prvkov.

Slovo	Tabuľka
123456	6
password	8
1	1
qwerty	6
...	...

Tabuľka 4.1: Príklad prvkov [29].

[1] https://github.com/hashcat/princeprocessor
[2] https://github.com/hashcat/princeprocessor/tree/bffda8cca9d47cc5567909f4ce7794aeaa81c4e1

Reťazec (chain)

Reťazec s dĺžkou L je usporiadaná sekvencia dĺžok prvkov, ktorých suma sa rovná L. Napríklad reťazec o dĺžky 8 môže byť (1, 6, 1) alebo (8). Existuje $2^{(L-1)}$ odlišných reťazcov pre dĺžku L.

Napríklad, reťazec s dĺžkou 4 sa skladá z prvkov:

- 4 písmenové slovo
- 2 písmenové slovo + 2 písmenové slovo
- 1 písmenové slovo + 3 písmenové slovo
- 1 písmenové slovo + 1 písmenové slovo + 2 písmenové slovo
- 1 písmenové slovo + 2 písmenové slovo + 1 písmenové slovo
- 1 písmenové slovo + 1 písmenové slovo + 1 písmenové slovo + 1 písmenové slovo
- ...

Veľkosť množiny možných hesiel (keyspace)

Algoritmus 1 popisuje výpočet veľkosti množiny možných hesiel u algoritmu PRINCE. Veľkosť množiny možných hesiel reťazca sa rovná počtu možných hesiel, ktoré môžu vzniknúť kombináciou všetkých možných prvkov podľa zoradenej sekvencie dĺžok v danom reťazci.

Napríklad, ak existuje X prvkov s dĺžkou 2 a Y prvkov o dĺžke 6 vo vstupnom slovníku, potom veľkosť množiny možných hesiel pre reťazec (6, 2) je $X * Y$. V tabuľke 4.1 je uvedený príklad veľkostí množín hesiel pre slovník *rockyou*[3].

Reťazec	Prvky	Veľkosť množiny možných hesiel
3 + 1	335 * 45	15 075
1 + 3	45 * 335	15 075
4	17 889	17 889
2 + 2	335 * 335	112 225
2 + 1 + 1	335 * 45 * 45	678 375
1 + 2 + 1	45 * 335 * 45	678 375
1 + 1 + 2	45 * 45 * 335	678 375
1 + 1 + 1 + 1	45 * 45 * 45 * 45	4 100 625

Tabuľka 4.2: Veľkosti množín hesiel pre reťazce dĺžky 4 (slovník *rockyou*) [29].

[3] https://www.kaggle.com/wjburns/common-password-list-rockyoutxt

Algoritmus 1: Výpočet veľkosti množiny možných hesiel v algoritme PRINCE[2].

Vstup: minimálna dĺžka hesiel (pw_min), maximálna dĺžka hesiel (pw_max), databáza reťazcov (chains_db), pole s počtom hesiel pre každú dĺžku (pw_ks_cnt)
Výstup: veľkosť množiny možných hesiel (total_ks_cnt)

```
1  total_ks_cnt = 0;
2  for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
3      chains_for_len = chains_db[pw_len];
4      chains_for_len_ks = 0;
5      for chains_idx = 0; chains_idx <
        chains_for_len.chains_cnt; chains_idx = chains_idx + 1 do
6          chain = chains_for_len.chains[chains_idx];
7          elems_for_chain = chain.elems;
8          chain.ks_cnt = 1;
9          for idx = 0; idx < chain.cnt; idx = idx + 1 do
10             key = chain[idx];
11             chain.ks_cnt = chain.ks_cnt * chains_db[key].elems_cnt;
12         chains_for_len_ks = chains_for_len_ks + chain.ks_cnt;
13     total_ks_cnt = total_ks_cnt + chains_for_len_ks;
14     if skip > 0 then
15         pw_ks_cnt[pw_len] = chains_for_len_ks
```

4.2 Popis algoritmu

Po prečítaní vstupného slovníka PRINCE sa uloží každé slovo (*element, prvok*) do tabuľky, ktorá sa skladá zo slov rovnakej dĺžky. Tento krok je možné popísať algoritmom 2.

Algoritmus 2: Uloženie slova zo vstupného slovníka do vnútornej dátovej reprezentácie slov v algoritme PRINCE[2].

Vstup: slovo zo vstupného slovníka (word), databáza reťazcov (chains_db)
Výstup: pridané slovo (word)

```
1  word_len = length(word);
2  chains_for_len = chains_db[word_len];
3  chains_for_len.elems[chains_for_len.elems_cnt] = word;
4  chains_for_len.elems_cnt = chains_for_len.elems_cnt + 1;
```

Podľa toho, akú štruktúru slovníka používateľ zvolí, útok PRINCE môže slúžiť na štandardný slovníkový útok (napríklad pridávanie / vkladanie číslic do vstupných slov), kombinačný útok alebo útok hrubou silou (skúšanie všetkých kombinácií písmen). Nové heslá sa generujú kombinovaním slov zo vstupného slovníka. Používateľ nemôže priamo a ani nepriamo špecifikovať štruktúru hesiel [19]. V algoritme PRINCE sa vytvárajú reťazce pozostávajúce z 1 až N rôznych prvkov. Inicializáciu týchto reťazcov popisuje algoritmus 3.

Algoritmus 3: Inicializácia reťazcov v algoritme PRINCE[2].

Vstup: minimálna dĺžka hesiel (pw_min), maximálna dĺžka hesiel (pw_max), minimálny počet prvkov v reťazci (elem_cnt_min), maximálny počet prvkov v reťazci (elem_cnt_max), databáza reťazcov (chains_db)

Výstup: inicializované reťazce

```
1  for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
2      chains_for_len = chains_db[pw_len];
3      pw_len1 = pw_len − 1;
4      chains_cnt = 1 << pw_len1;
5      new_chain = create_new_chain();
6      for chains_idx = 0; chains_idx < chains_cnt; chains_idx = chains_idx + 1
        do
7          chain_gen_with_idx(new_chain, pw_en1, chains_idx);
8          valid1 = chain_valid_with_db(new_chain, chains_db);
9          if valid1 == 0 then
10             continue;
11         valid2 = chain_valid_with_cnt_min(new_chain, elem_cnt_min);
12         if valid2 == 0 then
13             continue;
14         eff_elem_cnt_max = 0;
15         if elem_cnt_max > 0 then
16             eff_elem_cnt_max = elem_cnt_max;
17         else
18             eff_elem_cnt_max = pwlen + elem_cnt_max;
19             if eff_elem_cnt_max <= elem_cnt_min then
20                 continue;
21         valid3 = chain_valid_with_cnt_max(new_chain, eff_elem_cnt_max);
22         if valid3 == 0 then
23             continue;
24         chains_for_len.chains[chains_for_len.chains_cnt] = new_chain;
25         chains_for_len.ks_cnt = 0;
26         chains_for_len.ks_pos = 0;
27         chains_for_len.chains_cnt = chains_for_len.chains_cnt + 1;
28     clear(chains_for_len.cur_chain_ks_poses);
```

V aktuálnej referenčnej implementácii algoritmu PRINCE je v predvolenom nastavení N rovné hodnote 8. Hodnotu N si môže používateľ nastaviť pomocou možnosti `--elem-cnt-min`. Na obmedzenie počtu prvkov je určená možnosť `--elem-cnt-max`, ktorá môže byť užitočná pre útoky hrubou silou. Ak je napríklad nastavená na 4, algoritmus PRINCE bude kombinovať až do limitu 4 rôznych prvkov. Pri vstupnom slove „a“ môže vygenerovať nové heslo „aaaa“, ale už nie „aaaaa“. Algoritmus radí reťazce podľa veľkostí množín možných hesiel od najmenšej po najväčšiu. Radenie reťazcov v algoritme PRINCE je možné popísať algoritmom 4.

Algoritmus 4: Radenie reťazcov podľa veľkostí množín možných hesiel od najmenšej po najväčšiu v algoritme PRINCE[2].

Vstup: minimálna dĺžka hesiel (pw_min), maximálna dĺžka hesiel (pw_max), databáza reťazcov (chains_db)
Výstup: reťazce zoradené podľa veľkosti množiny možných hesiel od najmenšej po najväčšiu

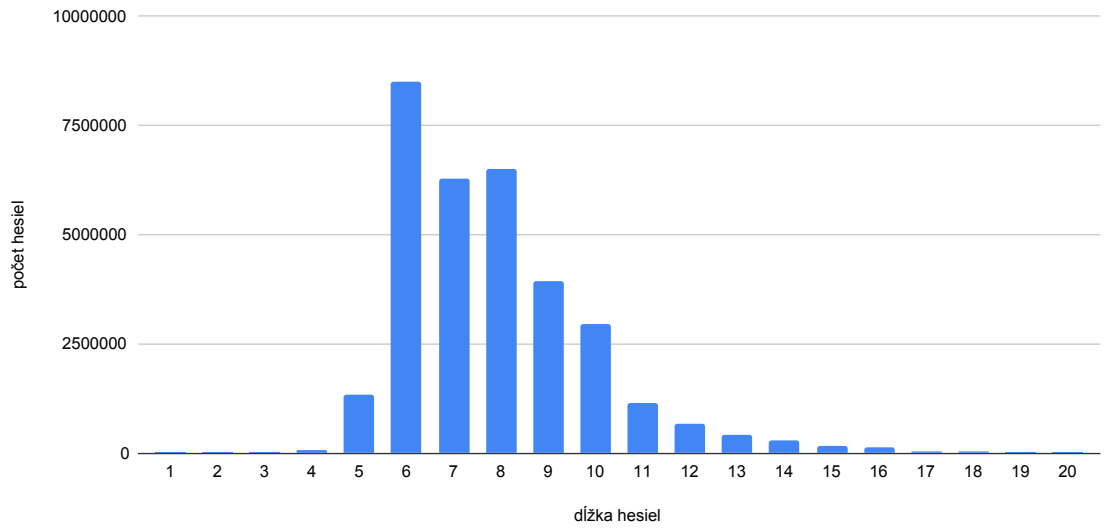
```
1 for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
2     chains_for_len = chains_db[pw_len];
3     sort(chains_for_len, [](chain1, chain2){return
        chain1.ks_cnt < chain2.ks_cnt});
```

Algoritmus si musí vybrať dĺžku pre nové možné heslá, ktoré sa budú generovať. Distribúcia dĺžok slov vo vstupnom slovníku je známa - PRINCE si vytvára vlastné štatistiky pre vstupný slovník. Tvorba týchto štatistík je popísaná algoritmom 5. Znalosť najbežnejších používaných dĺžkach hesiel pomáha vykonávať efektívne útoky, ktoré sú založené na dĺžke hesiel [22].

Algoritmus 5: Výpočet distribúcie dĺžok hesiel vo vstupnom slovníku u algoritmu PRINCE[2].

Vstup: minimálna dĺžka hesiel (pw_min), maximálna dĺžka hesiel (pw_max), databáza reťazcov (chains_db), dolné obmedzenie dĺžky hesiel (IN_LEN_MIN), horné obmedzenie dĺžky hesiel (IN_LEN_MAX)
Výstup: distribúcia dĺžok hesiel vo vstupnom slovníku (wordlen_dist)

```
1 pw_len = IN_LEN_MIN;
2 for pw_len = IN_LEN_MIN; pw_len < pw_max; pw_len = pw_len + 1 do
3     chains_for_len = chains_db[pw_len];
4     if pw_len <= IN_LEN_MAX then
5         wordlen_dist[pw_len] = chains_db[pw_len].elems_cnt;
6     else
7         wordlen_dist[pw_len] = 1;
```



Obrázok 4.1: Distribúcia dĺžok hesiel zo stránky *rockyou*. Prevzaté a upravené z prezentácie [29].

Na obrázku 4.1 je zobrazená distribúcia dĺžok hesiel získaná z úniku hesiel zo stránky *Rockyou*. Dáta ukazujú, aká je najčastejšia dĺžka hesiel, a táto znalosť môže byť použitá pre útok, kde útočník môže efektívne využiť čas a zdroje a skúšať len heslá s napríklad 6 až 10 znakmi. V prípade hesiel z *Rockyou* je najbežnejšou dĺžkou dĺžka 6. Algoritmus PRINCE najskôr vygeneruje určité množstvo možných hesiel práve tejto dĺžky a potom prejde na nasledujúcu najbežnejšiu dĺžku. Toto je dôležitý aspekt, ktorý treba vziať do úvahy, pokiaľ ide o obsah vstupného slovníka, a preto by mal čo najviac zodpovedať realistickému rozdeleniu dĺžok hesiel.

Generovanie hesiel od určitej pozície a v určitom počte je podporované algoritmom PRINCE. Pred behom jadra algoritmu PRINCE je nutné spustiť pomocný algoritmus 6 aby sa na základe stanovenej pozície a počtu hesiel upravila[4] vnútorná dátová reprezentácia slov. Algoritmus 7 popisuje jadro algoritmu PRINCE, tj. hlavný cyklus, v ktorom sa generujú kandidátne heslá. V algoritme sú použité pomocné funkcie z referenčnej implementácie[2].

[4] https://github.com/hashcat/princeprocessor/blob/bb05e722f96a2ef3d60c6493ea9aeb6174f11aa0/src/pp.c

Algoritmus 6: Podpora generovania hesiel od určitej pozície a v určitom počte u algoritmu PRINCE[2].

Vstup: počet preskočených hesiel (skip), počet vygenerovaných hesiel (limit), distribúcia dĺžok hesiel vo vstupnom slovníku (wordlen_dist)
Výstup: upravená vnútorná dátová reprezentácia

```
1  if limit > 0 then
2      total_ks_cnt = skip + limit;
3  if skip > 0 then
4      skip_left = skip;
5      main_loops = 0;
6      outs_per_main_loop = 0;
7      for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
8          pw_ks_pos[pw_len] = 0;
9          outs_per_main_loop = outs_per_main_loop + wordlen_dist[pw_len];
10     while true do
11         main_loops = skip_left/outs_per_main_loop;
12         if main_loops == 0 then
13             break;
14         for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
15             if pw_ks_pos[pw_len] < pw_ks_cnt[pw_len] then
16                 tmp = main_loops * wordlen_dist[pw_len];
17                 pw_ks_pos[pw_len] = pw_ks_pos[pw_len] + tmp;
18                 skip_left = skip_left - tmp;
19                 if pw_ks_pos[pw_len] > pw_ks_cnt[pw_len] then
20                     tmp = pw_ks_pos[pw_len] - pw_ks_cnt[pw_len];
21                     skip_left = skip_left + tmp;
22         outs_per_main_loop = 0;
23         for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
24             if pw_ks_pos[pw_len] < pw_ks_cnt[pw_len] then
25                 outs_per_main_loop =
                    outs_per_main_loop + wordlen_dist[pw_len];
26     total_ks_pos = skip - skip_left;
27     for pw_len = pw_min; pw_len < pw_max; pw_len = pw_len + 1 do
28         chains_for_len = chains_db[pw_len];
29         tmp = pw_ks_pos[pw_len];
30         for chains_idx = 0; chains_idx <
            chains_for_len.chains_cnt; chains_idx = chains_idx + 1 do
31             chain = chains_for_len.chains[chains_idx];
32             if tmp < chain.ks_cnt then
33                 chain.ks_cnt = tmp;
34                 set_chain_ks_poses(chain, chains_db, tmp,
                    _for_len.cur_chain_ks_poses);
35             chains_for_len.chains_pos = chains_for_len.chains_pos + 1;
```

Algoritmus 7: Generovanie hesiel pomocou algoritmu PRINCE[2].

Vstup: počet preskočených hesiel (skip), počet vygenerovaných hesiel (limit), databáza reťazcov (chains_db), poradie podľa dĺžok hesiel (pw_orders)

Výstup: kandidátne heslá

```
1  iter_max = 0;
2  while total_ks_pos < total_ks_cnt do
3      for order_pos = 0; order_pos < order_cnt; order_pos = order_pos + 1 do
4          pw_len = pw_orders[order_pos].len;
5          outs_pos = 0;
6          chains_for_len = chains_db[pw_len], outs_cnt =
             wordlen_dist[pw_len];
7          while outs_pos < outs_cnt do
8              chains_cnt = chains_for_len.chains_cnt;
9              chains_pos = chains_for_len.chains_pos;
10             if chains_pos == chains_cnt then
11                 break;
12             chain = chains_for_len.chains[chains_pos];
13             total_ks_left = total_ks_left − total_ks_pos;
14             iter_max = chain.ks_cnt − chain.ks_pos;
15             if total_ks_left < iter_max then
16                 iter_max = total_ks_left;
17             tmp = outs_left = outs_cnt − outs_pos;
18             if tmp < iter_max then
19                 iter_max = tmp;
20             tmp = total_ks_pos + iter_max;
21             if tmp > skip then
22                 iter_pos = 0;
23                 if total_ks_pos < skip then
24                     iter_pos = tmp = skip − total_ks_pos;
25                     tmp = chain.ks_pos + tmp;
26                     set_chain_ks_poses(chain, chains_db, tmp,
                         chains_for_len.cur_chain_ks_poses);
27                 pw = chain_set_pw_init(chain, chains_db,
                     chains_for_len.cur_chain_ks_poses)
                     iter_pos_save = iter_max − iter_pos;
28                 while iter_pos < iter_max do
29                     generate(pw);
30                     chain_set_pw_increment(chain, chains_db,
                         chains_for_len.cur_chain_ks_poses, pw);
31                     iter_pos = iter_pos + 1;
32             else
33                 tmp = chain.ks_pos + iter_max;
34                 set_chain_ks_poses(chain, chains_db, tmp,
                     chains_for_len.cur_chain_ks_poses);
35             outs_pos = outs_pos + iter_max;
36             total_ks_pos = total_ks_pos + iter_max;
37             chain.ks_pos = chain.ks_pos + iter_max;
38             if chain.ks_pos == chain.ks_cnt then
39                 chains_for_len.chains_pos = chains_for_len.chains_pos + 1;
40                 clear(chains_for_len.cur_chain_ks_poses);
41             if total_ks_pos == total_ks_cnt then
42                 break;
43         if total_ks_pos == total_ks_cnt then
44             break;
```

Činnosť algoritmu PRINCE bude predstavená na nasledujúcom príklade, kde vstupný slovník bude obsahovať štyri riadky so slovami „A“, „BB“, „CCC“, „DD“.

1. **Získanie prvkov zo vstupného slovníka**

   Obsah vstupného slovníka:

   ```
   A~BB
   CCC
   DD
   ```

   Každý riadok (slovo) reprezentuje jeden prvok, takže v našom príklade máme štyri prvky „A“, „BB“, „CCC“, „DD“. Načítané slovo sa uloží do vnútornej dátovej reprezentácie slov podľa algoritmu 2.

2. **Vytvorenie reťazcov pozostávajúcich z 1 až N rôznych prvkov (pre jednoduchosť N = 3)**

   Reťazce pozostávajúce z 1 až N rôznych prvkov sa vytvoria podľa algoritmu 3.

   Ukážka reťazcov:

   A - reťazec (1) o dĺžke 1
   BB - reťazec (2) o dĺžke 2
   AA - reťazec (1, 1) o dĺžke 2
   DD - reťazec (2) o dĺžke 2
   CCC - reťazec (3) o dĺžke 3
   ABB - reťazec (1, 2) o dĺžke 3
   ...
   ACCC - reťazec (1, 3) o dĺžke 4
   ...
   BBCCC - reťazec (2, 3) o dĺžke 5

   ...
   CCCADD - reťazec (3, 1, 2) o dĺžke 6
   ...
   ACCCCCC - reťazec (1, 3, 3) o dĺžke 7

   ...
   BBCCCCCC - reťazec (2, 3, 3) o dĺžke 8
   ...
   CCCCCCCCC - reťazec (3, 3, 3) o dĺžke 9

3. **Výpočet veľkosti množiny možných hesiel**

   Tento krok je realizovaný algoritmom 1, ktorý popisuje výpočet veľkosti množiny možných hesiel.

   Ukážka veľkostí množín možných hesiel pre niektoré reťazce:

   reťazec (1) - veľkosť je 1
   reťazec (1, 1) - veľkosť je $1 * 1 = 1$
   reťazec (2, 2) - veľkosť je $2 * 2 = 4$

reťazec (1, 2, 3) - veľkosť je $1 * 2 * 1 = 2$
reťazec (2, 2, 2) - veľkosť je $2 * 2 * 2 = 8$
reťazec (2, 3, 2) - veľkosť je $2 * 1 * 2 = 4$

Výpočet pre reťazec (2, 3, 2) prebieha nasledovným spôsobom. Máme X prvkov o dĺžke 2, Y prvkov o dĺžke 1, Z prvkov o dĺžke 2. Výsledok je vypočítaný ako X * Y * Z. V našom prípade X = Z = 2 a Y = 1. Veľkosť množiny možných hesiel pre reťazec (2, 3, 2) je 4.

4. **Výstup algoritmu - vygenerované heslá**

   Reťazce sú podľa algoritmu 4 zoradené podľa veľkostí množín možných hesiel od najmenšej po najväčšiu. Následne sa spúšťa algoritmus 7, ktorý generuje kandidátne heslá.

   Pre náš príklad výstupom algoritmu bude nasledovný zoznam možných hesiel:

   ```
   AA, BB, DD, A, CCC, AAA, ABB, ADD, BBA, DDA, ACCC, CCCA, AABB, AADD
   ABBA, ADDA, BBAA, DDAA, BBBB, DDBB, BBDD, DDDD, AACCC, ACCCA, ...
   ```

4.3 Nástroj princeprocessor

Nástroj princeprocessor obsahuje referenčnú implementáciu algoritmu PRINCE.

Činnosť nástroja

Nástroj v prvom kroku načíta slová zo vstupného slovníka a uloží ich do pamäte. Následne vygeneruje reťazce s prvkami pre každú dĺžku hesiel a zahadzuje reťazce, ktoré obsahujú prvok, ktorý ukazuje na neexistujúcu dĺžku hesla. Zreťazené prvky sa radia podľa veľkosti množiny možných hesiel odpovedajúcej konkrétnemu reťazcu. Nástroj iteruje nad množinou možných hesiel a v každej iterácii vykonáva nasledovné kroky:

1. vyberá ďalší reťazec danej dĺžky
2. generuje heslá pomocou tohto reťazca
3. vypisuje heslá na štandardný výstup alebo do súboru

Použitie nástroja

Samotné použitie nástroja princeprocessor je jednoduché a pozostáva z troch krokov:

1. Stiahnutie nástroja princeprocessor z Github repozitáru [5] a jeho zostavenia
2. Výber vstupného slovníka. Je možné použiť známe slovníky (napr. *rockyou*) alebo špecifické slovníky optimalizované pre daný cieľ.
3. Lámanie možných hesiel získaných z nástroja princeprocessor

   ```
   ./pp64 < wordlist.txt | ./oclHashcat hash.txt
   ```

[5] https://github.com/hashcat/princeprocessor

Používateľ si u nástroja princeprocessor môže nastaviť nasledovné možnosti ovplyvňujúce vygenerovaný výstup s heslami:

- minimálnu / maximálnu dĺžku hesla
- rozsah možných hesiel
- minimálnu / maximálnu dĺžku prvku (elementu)
- povoliť permutáciu prvého písmena každého slova v slovníku
- zakázať kontrolu duplicitných hesiel
- výstupný súbor (v predvolenom nastavení sa heslá generujú na štandardný výstup)

Vstavaná možnosť nastaviť rozsah generovaných hesiel je zaujíma, keďže túto funkcionalitu je možné využiť pre distribúciu útoku PRINCE [8]. Každý klient môže dostať rozsah možných hesiel, ktoré bude lámať. Výhodou je, že je možné presne špecifikovať rozsah pre každého klienta zvlášť: výkonnejší a rýchlejší klienti môžu lámať väčší počet hesiel oproti tým pomalším.

4.4 Nástroj PRINCE-LING

Nástroj PRINCE-LING[6] (*PRINCE Language Indexed N-Grams*) generuje slovníky na základe natrénovaných pravdepodobnostných bezkontextových gramatík pomocou nástroja PCFG Trainer[7]. Tieto slovníky sú vytvárané špeciálne pre útok PRINCE. Slová v slovníku sú generované podľa poradia daného ich pravdepodobnosťami a s ohľadom na to, ako užitočné sú tieto heslá pri útoku PRINCE.

4.4.1 Výhody a nevýhody útoku PRINCE

Oproti iným typom útokov má PRINCE niektoré zaujímave alebo unikátne výhody. Algoritmus použitý v útoku PRINCE je navrhnutý s myšlienkou jednoduchosti, vďaka čomu je použitie algoritmu nenáročné. Nevyžaduje žiadnu špeciálnu syntax slov v slovníku. Generuje kvalitné kandidátne heslá. Primárnym cieľom algoritmu je umožniť používateľom jednoduchý prechod z klasických kombinačných útokov. Vďaka jednoduchej myšlienke algoritmu je útok pomocou algoritmu PRINCE rýchly a efektívny.

Implementačným obmedzením z časových a pamäťových dôvodov pre dlhé výstupy je obmedzenie maximálnej možnej dĺžky reťazcov. Problémom môže byť aj generovanie duplicitných hesiel - tento problém je vyriešený v novších verziách nástroja princeprocessor, kde sa kontrolujú možné duplicitné heslá. V nástroji princeprocessor je možné túto kontrolu vypnúť pomocou špeciálneho prepínača `--dupe-check-disable`.

4.4.2 Varianty útoku PRINCE

Podľa spôsobu použitia nástroja princeprocessor s inými nástrojmi je možné hovoriť o nasledovných variantách útoku PRINCE:

[6] https://github.com/lakiw/pcfg_cracker/blob/master/prince_ling.py
[7] https://github.com/lakiw/pcfg_cracker/blob/master/trainer.py

- **Klasický PRINCE útok**

  Pomocou nástroja princeprocessor sa generujú kandidátne heslá a tieto heslá sa následne lámu pomocou nástroja hashcat.

- **PRINCECEPTION útok**

  U PRINCECEPTION útoku [24] sa pomocou nástroja princeprocessor generujú heslá na vstup druhej inštancie nástroja princeprocessor, ktorého výstupom sú kandidátne heslá, ktoré sa budú lámať pomocou nástroja hashcat.

- **Purple Rain útok**

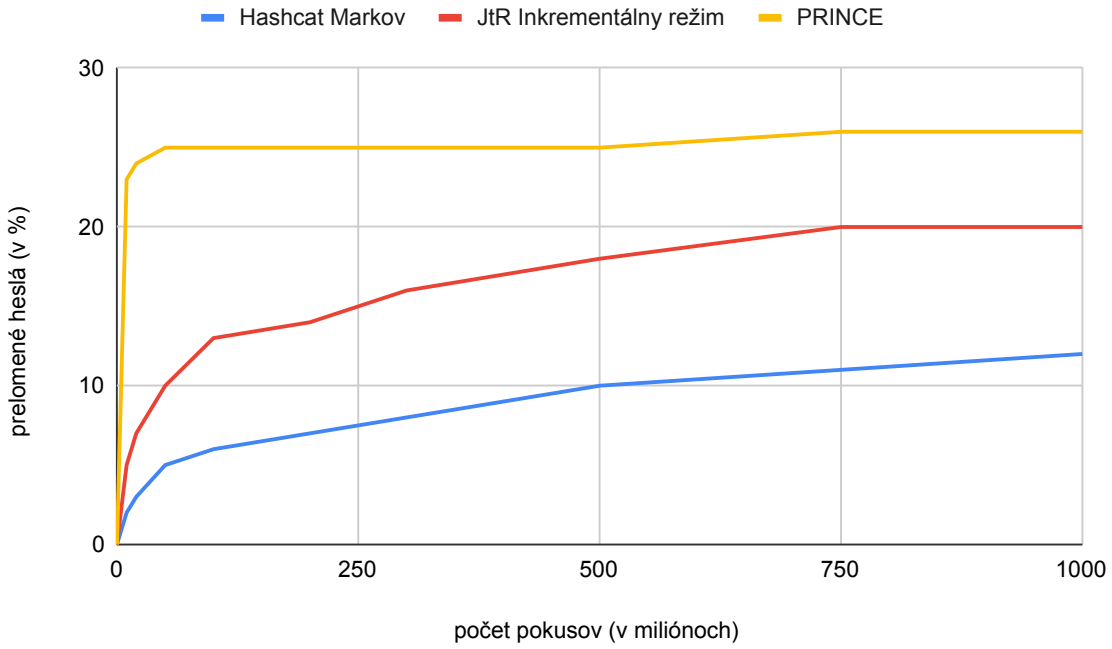
  Purple Rain útok [23] je zameraný na dynamické vytváranie nekonečného množstva kombinácií hesiel. Slovník, s ktorým sa má útok vykonať, sa najskôr náhodne premieša a následne je použitý ako vstupný slovník pre nástroj princeprocessor, ktorý generuje kandidátne heslá, ktoré sa následne lámu pomocou nástroja hashcat. Hashcat počas lámania generuje náhodné mutačné pravidlá, pomocou ktorých sa vytvárajú z každého hesla ďalšie nové heslá, ktoré sa tiež lámu.

4.5 Porovnanie s ostatnými útokmi

Útok pomocou algoritmu PRINCE bol pred pár rokmi porovnávaný s rôznymi ostatnými typmi útokov na rôznych dátových sadách [31]. Obsahom tejto podkapitoly je popis a analýza týchto experimentov, ktoré poukazujú na silné a slabé stránky PRINCE.

4.5.1 PRINCE, Hashcat Markov režim a John The Ripper v inkrementálnom režime

Vstupným slovníkom pre PRINCE bolo prvých 100 000 najpopulárnejších hesiel zo slovníka *rockyou*. Pre nástroj hashcat bol vygenerovaný štatistický súbor pre celý slovník *rockyou* s limitom 16. U nástroja John The Ripper bol použitý predvolený inkrementálny režim so znakovou sadou „All“. Cieľom útokov bol starý zoznam MySpace.

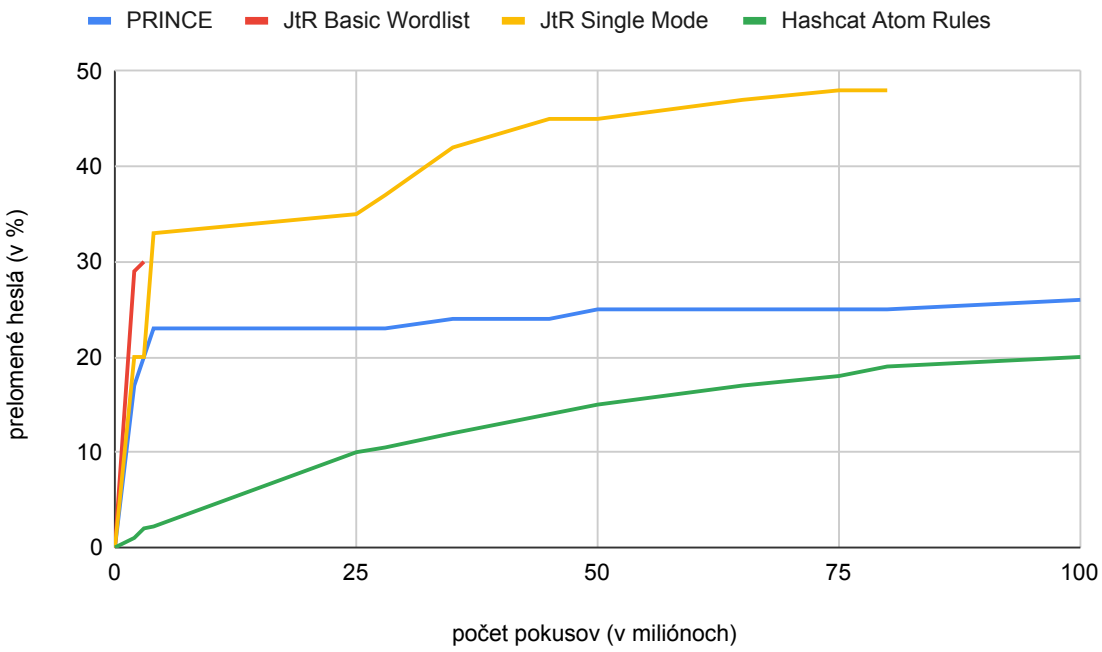


Obrázok 4.2: Výsledky experimentu s PRINCE, Hashcat Markov režimom a John The Ripper v inkrementálnom režime. Prevzaté a upravené zo stránky [31].

Graf na obrázku 4.2 ukazuje, že PRINCE bol zo začiatku veľmi efektívny, no postupne strácal na efektivite. Vysvetlením je fakt, že PRINCE na začiatku použil väčšinu najviac populárnych slov zo slovníka *rockyou* a nadobudol charakter normálneho slovníkového útoku. Taktiež vidíme, že koncu behu začal inkrementálny režim dobiehať PRINCE a dosahovať podobné výsledky.

4.5.2 PRINCE a slovníkové útoky

Experiment prebiehal v rovnakej konfigurácii ako predošlý experiment, no v tomto experimente boli testované slovníkové útoky. V prípade nástroja John The Ripper bola použitá predvolená sada pravidiel a pokročilejšia sada „Simple“. U nástroja hashcatu bola použitá sada pravidiel „Atom“. U všetkých slovníkových útokoch bol použitý slovník obsahujúci 100 000 najpopulárnejších hesiel zo slovníka *rockyou*.

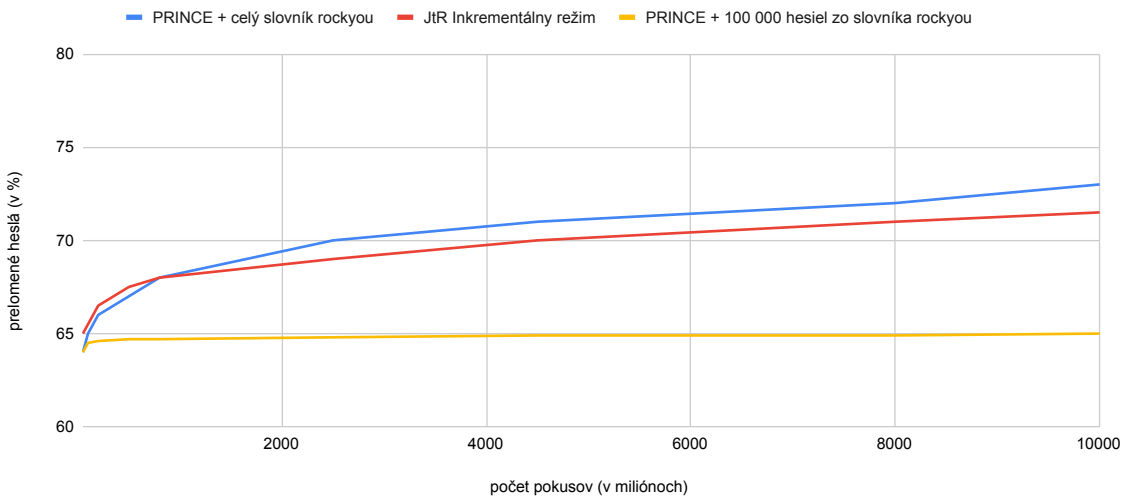


Obrázok 4.3: Výsledky experimentu s PRINCE a slovníkovými útokmi. Prevzaté a upravené zo stránky [31].

Výsledky experimentu znázornené na obrázku 4.3 jasne poukazujú na dominanciu slovníkových útokov. Všetky slovníkové útoky boli oveľa lepšie ako PRINCE. Tieto výsledky nie sú prekvapivé, keďže ich sada pravidiel bola ručne vytváraná, narozdiel od PRINCE, kde si PRINCE sám generoval pravidlá automaticky za behu. Tento experiment poukázal na to, že pre malé úlohy je vhodné použiť klasický slovníkový útok namiesto PRINCE.

4.5.3 PRINCE a zoznam slov z John The Ripper + inkrementálny režim

V tomto experimente sa porovnávala efektivita PRINCE s rôznymi slovníkmi. V prvom prípade bol použitý slovník *rockyou*, ktorý bol zmenšený na prvých 100 000 položiek a v druhom prípade bol použitý celý slovník. Taktiež bolo vykonané lámanie hesiel pomocou inkrementálneho útoku hrubou silou.



Obrázok 4.4: Výsledky experimentu s PRINCE a John The Ripper v inkrementálnom režime. Prevzaté a upravené zo stránky [31].

Ako vidíme na obrázku 4.4, pri použití slovníku *rockyou* so 100 000 položkam sú výsledky útoku PRINCE dosť slabé. Pri použití celého slovníka sú výsledky najlepšie a útok PRINCE dokázal poraziť aj inkrementálny útok hrubou silou. Tento experiment poukázal na fakt, že efektívnosť útoku pomocou PRINCE výrazne závisí od použitého vstupného slovníka. Experiment ukázal silu útoku PRINCE pri použití celého slovníka, ale otázkou je, prečo boli výsledky také rozdielne pri použití menšieho a celého slovníka. Autor experimentu Matt Weir sa domnieva, že v tomto prípade dochádza k emulácii dlhšieho slovníkového útoku, no na úplné potvrdenie tejto teórie je potrebné vykonať ďalšie experimenty.

Kapitola 5

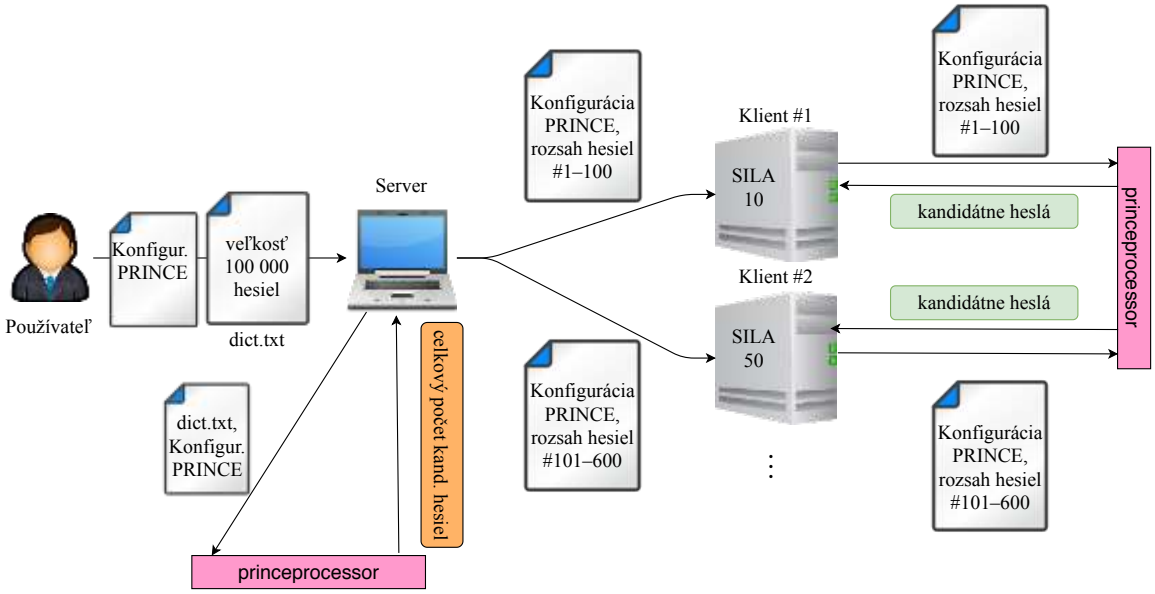
Návrh rozšírenia systému Fitcrack

Obsahom tejto kapitoly je návrh realizácie distribuovaného útoku založeného na algoritme PRINCE a jeho integrácie do systému Fitcrack. V podkapitolách popisujem zmeny v častiach systému, ktoré je nutné upraviť alebo rozšíriť z dôvodu pridania tohto nového typu útoku.

Návrh distribuovaného útoku PRINCE

Po analýze možností, ktoré ponúka nástroj princeprocessor, som sa rozhodol, že distribuovanosť útoku bude založená na prepínačoch `--skip` a `--limit`, ktoré tento nástroj podporuje.

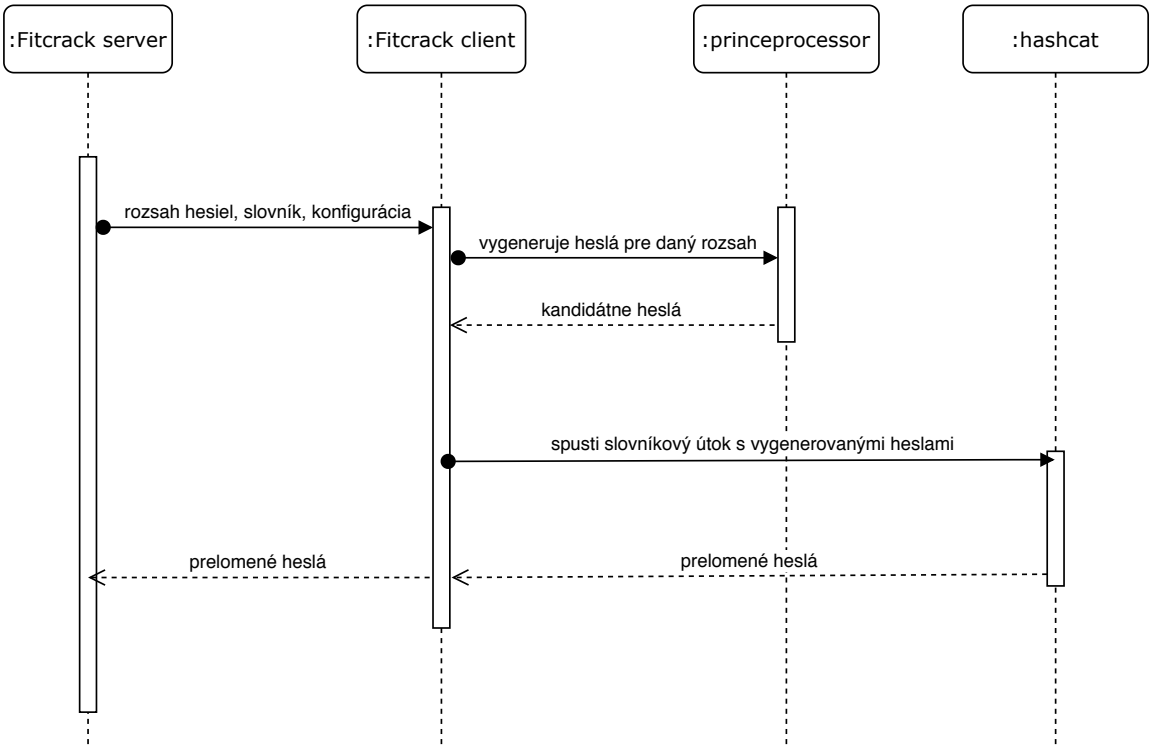
Jedno riešenie distribúcie by mohlo byť založené na myšlienke, že server určí rozsah podmnožiny množiny možných hesiel pre klienta a tento rozsah mu pomocou nástroja princeprocessor vygeneruje a pošle. Problém tohto riešenia je spomalenie servera a celého útoku kvôli sekvenčnému generovaniu možných hesiel pre klientov na strane servera. Klienti by museli čakať kým sa najskôr vygeneruje a pošle súbor s možnými heslami, ktoré majú spracovávať, a až po prijatí by klienti mohli začať lámať heslá. Problematická by mohla byť aj veľkosť týchto vygenerovaných súborov, ktoré by sa prenášali sieťou. U druhého možného riešenia útoku, ktoré sa javí ako rýchlejšie a efektívnejšie, by server jednotlivým klientom posielal iba informáciu o určitom rozsahu, ktorý bude určovať podmnožinu množiny všetkých možných hesiel vygenerovaných nástrojom princeprocessor, ktorú bude klient lámať. Návrh integrácie tohto riešenia distribúcie útoku PRINCE do systému Fitcrack je znázornený na obrázku 5.1.



Obrázok 5.1: Schéma navrhovanej distribúcie útoku PRINCE.

Klient si po prijatí úlohy spustí nástroj princeprocessor, aby mu vygeneroval podmnožinu možných hesiel pomocou algoritmu PRINCE, ktorú sa následne pokúsiť lámať pomocou nástroja hashcat v režime slovníkového útoku. Priebeh lámania hesiel bude klient oznamovať serveru, ktorému na konci oznámi výsledok lámania. Po získaní výsledku lámania od každého klienta je možné vytvoriť správu s konečným výsledkom lámania danej úlohy.

Na obrázku 5.2 je znázornený sekvenčný diagram, ktorý popisuje navrhovaný spôsob realizácie distribuovaného útoku pomocou algoritmu PRINCE. Fitcrack server pošle klientovi dáta o úlohe, ktoré obsahujú vstupný slovník, informácie o nastavení útoku a rozsah v rámci celej množiny možných hesiel, ktorú algoritmus PRINCE vygeneruje. Na strane klienta sa následne spustí program Runner so získanými informáciami o nastavení útoku a rozsahu. Program Runner sa postará o celý priebeh útoku na strane klienta, kde spúšťa nástroj hashcat na lámanie hesiel a nástroj princeprocessor so vstupným slovníkom a rozsahom hesiel, ktorý mu pridelil Fitcrack server.

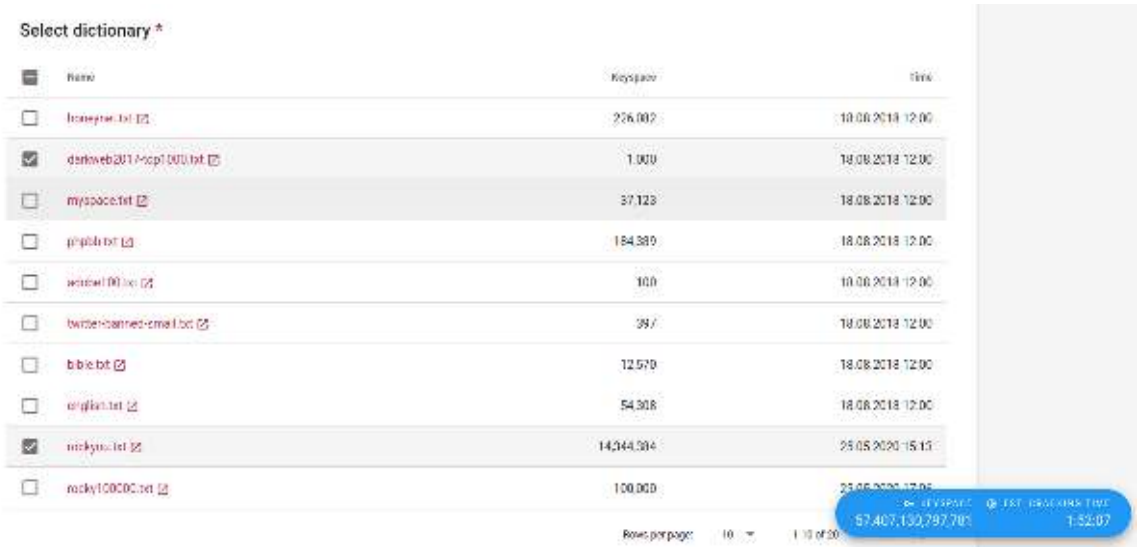


Obrázok 5.2: Sekvenčný popisujúci distribúciu útoku PRINCE medzi klientov.

Možným riešením útoku by bolo najskôr vygenerovať možné heslá do súboru pomocou nástroja princeprocessor a následne spustiť slovníkový útok pomocou nástroja hashcat s týmto vygenerovaným súborom ako so vstupným slovníkom. Toto riešenie nie je efektívne kvôli zbytočnému čakaniu na vygenerovanie celého súboru. Lepším a efektívnejším riešením je priebežné generovanie možných hesiel, ktoré sa môžu hneď lámať pomocou nástroja hashcat. Nástroj princeprocessor začne generovať heslá pomocou algoritmu PRINCE, ktoré budú presmerované na vstup nástroja hashcat, ktorý ich začne lámať. Informácie o priebehu lámania budú odosielané serveru pomocou správ *Trickle*. Po dokončení lámania si program Runner zistí, akým stavovým kódom skončilo lámanie a tento kód spolu s prípadnými získanými heslami odošle späť serveru.

5.1 Tvorba úlohy s útokom PRINCE

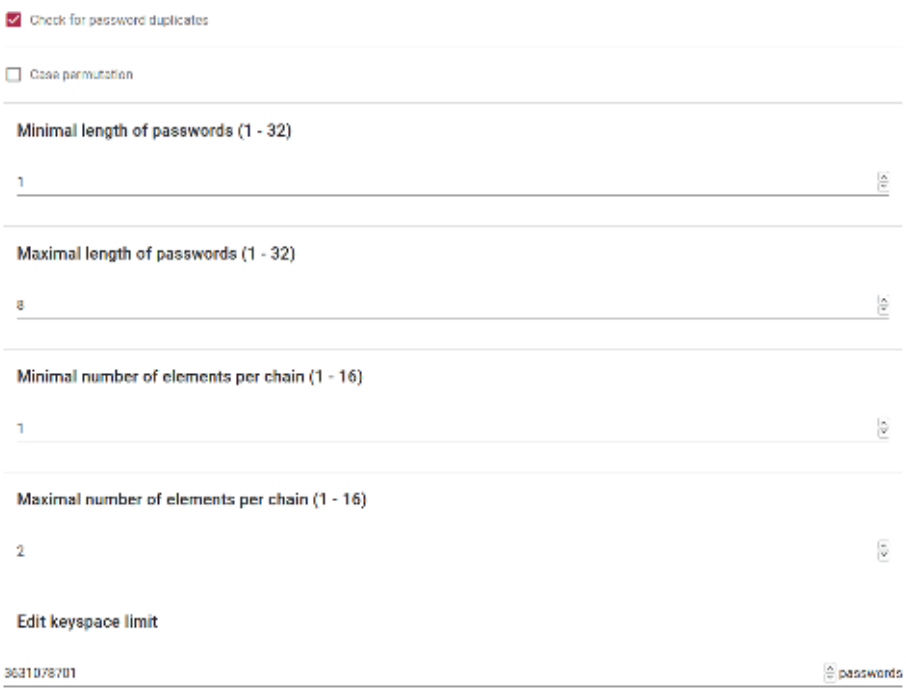
Súčasná ponuka útokov bude rozšírená o útok PRINCE a používateľ si bude môcť jednoducho nakonfigurovať útok v prehľadnom webovom rozhraní. V hornej časti konfigurácie útoku si používateľ vyberie slovník, ktorý bude použitý ako vstupný slovník v algoritme PRINCE. Systém bude podporovať voľbu viacerých slovníkov a interne vznikne jeden dočasný slovník, ktorý vznikne spojením viacerých vybraných slovníkov. Ukážka navrhovaného vzhľadu rozhrania na výber slovníkov je na obrázku 5.3.



Obrázok 5.3: Výber slovníka pre útok PRINCE.

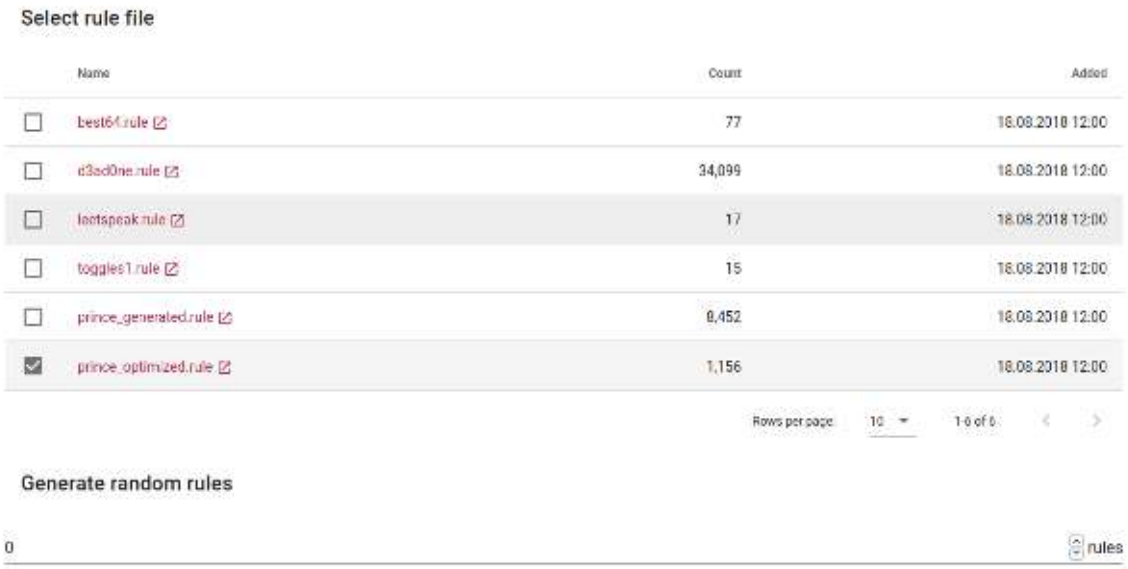
Po výbere slovníka bude možné upraviť konfiguračných možností útoku PRINCE, ktoré sa viažu priamo na algoritmus PRINCE. Návrh tejto časti grafického rozhrania je znázornený na obrázku 5.4. Používateľ si tu bude môcť vypnúť alebo zapnúť kontrolu duplicitných hesiel a permutáciu prvých písmen slov, nastaviť minimálnu a maximálnu dĺžku kandidátnych hesiel a minimálny a maximálny počet prvkov v reťazci. U každej konfiguračnej možnosti bude prebiehať kontrola správnosti zadaných údajov a v prípade nesprávnych vstupných údajov bude používateľ informovaný o danom probléme.

Systém na základe konfigurácie útoku bude používateľovi schopný zobraziť celkový počet kandidátnych hesiel. Po pridaní klientov, ktorí sa budú podieľať na výpočte úlohy, systém zobrazí aj odhadovaný čas trvania danej úlohy.



Obrázok 5.4: Konfiguračné možnosti útoku PRINCE.

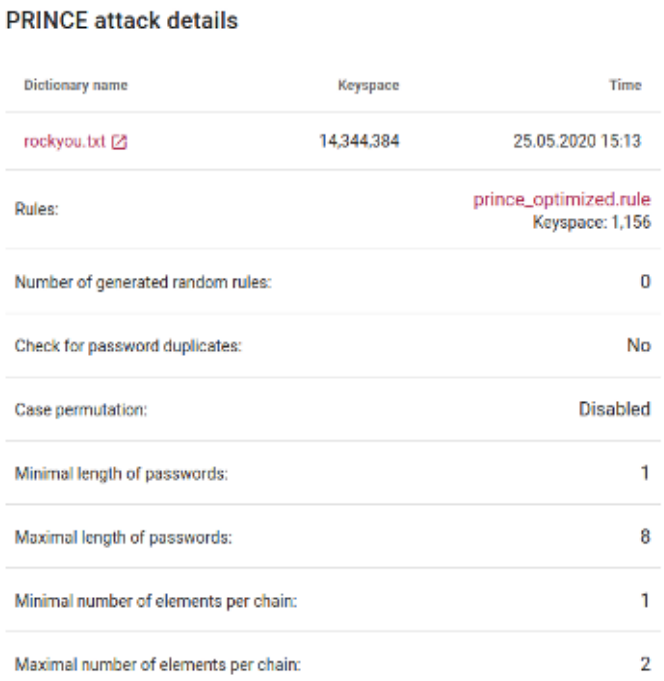
Používateľ v prípade záujmu bude môcť používať aj pravidlá, ktoré sa aplikujú na vstupné heslá v nástroji hashcat, čím sa vytvorí mnohonásobne viac hesiel, ktoré sa budú overovať. Bude možné si vybrať medzi automatickým náhodným generovaním pravidiel a súborom s pravidlami, ktorý si používateľ môže nahrať do systému Fitcrack. Návrh grafického rozhrania na výber použitia pravidiel je na obrázku 5.5.



Obrázok 5.5: Výber pravidiel pre útok PRINCE.

5.2 Informácie o úlohe s útokom PRINCE

Po vytvorení úlohy systém Fitcrack zobrazí stránku s informáciami o úlohe, o jej priebehu, prelomených hešoch, a pod. Stránku s informáciami o úlohe je taktiež možné otvoriť zo zoznamu všetkých úloh. Stránka je informačne bohatá, čo je silnou stránkou systému Fitcrack oproti iným systémom slúžiacim na lámanie hesiel. Obsahuje časť, ktorej obsah sa mení podľa použitého typu útoku. Pre útok PRINCE bude do tejto časti pridaná informácia o použitom slovníku / slovníkoch, pravidlách a o hodnotách nastavení, ktoré sú špecifické pre útok PRINCE. Ukážka navrhovaného vzhľadu a obsahu tejto časti je na obrázku 5.6.

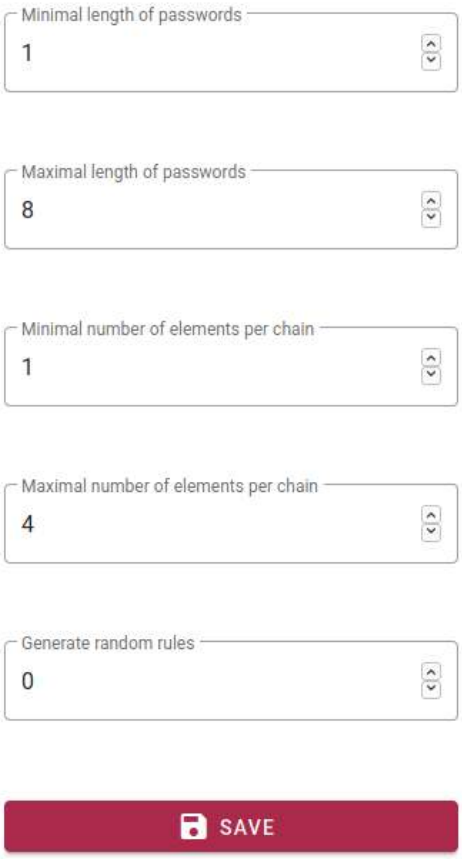


Obrázok 5.6: Informácie o úlohe špecifické pre útok PRINCE.

5.3 Editácia úlohy s útokom PRINCE

Počas vytvárania úlohy môže používateľ urobiť chybu pri konfigurovaní útoku, ktorá nemusí skončiť chybou, keďže sa môže jednať o platné hodnoty nastavení, alebo by už dokončenú úlohu chcel spustiť s novými hodnotami nastavení útoku. V oboch prípadoch je síce možné vytvorenie novej úlohy, no lepším riešením je pridanie podpory pre editáciu úlohy na stránku s informáciami o úlohe.

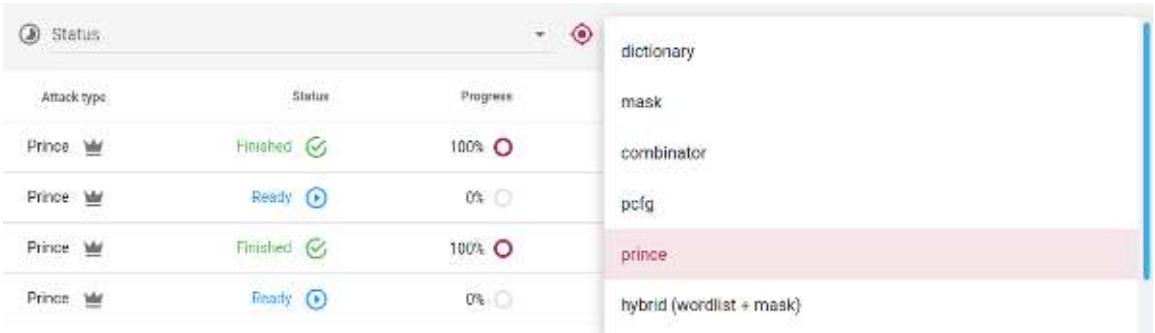
Návrh rozhrania na editovanie nastavení útoku PRINCE je zobrazený na obrázku 5.7. Okrem bežných položiek, ktoré je možné zmeniť u každého typu útoku, bude možné zmeniť aj položky, ktoré sú špecifické pre útok PRINCE. Po uložení zmien dôjde ku kontrole nových nastavení. Na chybné nastavenia bude používateľ upozornený a k žiadnym zmenám v nastavení úlohy nedôjde. Ak sú zadané údaje správne, celkový počet kandidátnych hesiel sa automaticky prepočíta a používateľ môže znova spustiť danú úlohu.



Obrázok 5.7: Editácia úlohy s útokom PRINCE.

5.4 Rozšírenie filtra v zozname úloh

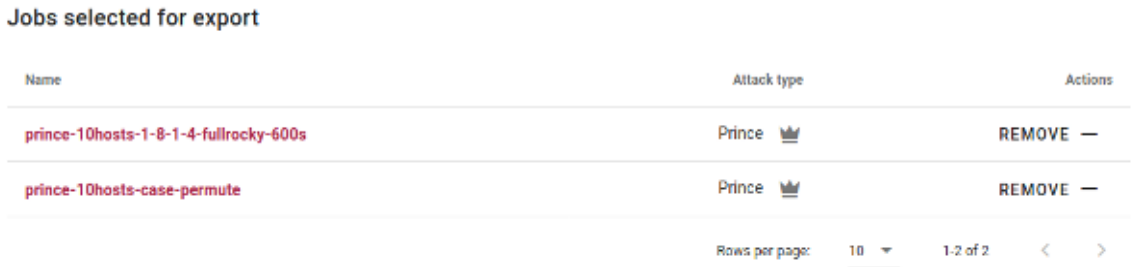
V súčasnosti je možné v zozname úloh filtrovať úlohy podľa existujúcich typov útokov, tj. slovníkového, kombinačného, hybridného (obe varianty) a útoku PCFG. Tento filter bude rozšírený o útok PRINCE aby si používateľ mohol zobraziť len úlohy s týmto útokom. Navrhované rozšírenie zoznamu typov útokov, podľa ktorých je možné filtrovať úlohy, je zobrazené na obrázku 5.8.



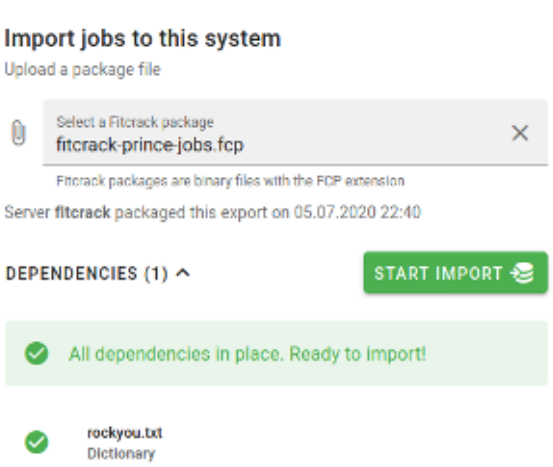
Obrázok 5.8: Rozšírenie filtra o útok PRINCE v zozname úloh.

5.5 Podpora útoku PRINCE pri exporte / importe úloh

Do systému Fitcrack bola v rámci bakalárskej práce [13] pridaná funkcionalita na exportovanie a importovanie nastavení úloh. Keďže útok PRINCE obsahuje vlastné konfiguračné možnosti, je nutné pridať podporu pre tento nový útok, aby systém vedel správne exportovať alebo importovať všetky konfiguračné možnosti u úloh s útokom PRINCE. Táto úprava bude spočívať v pridaní nových položiek špecifických pre PRINCE do zoznamu konfiguračných možností, ktoré sa majú exportovať / importovať.



Obrázok 5.9: Možnosť exportu úlohy s útokom PRINCE.



Obrázok 5.10: Možnosť importu úlohy s útokom PRINCE.

Na obrázku 5.9 je zobrazené rozhranie na export úloh, kde bude pridaná podpora pre útok PRINCE. Pri exporte úlohy s útokom PRINCE budú do zálohy uložená konfigurácia úlohy spolu s pravidlami. Slovník podobne ako aj v prípade iných útokov nebude do zálohy pridaný z kapacitných dôvodov, keďže používateľ daný slovník už môže používať pre iné úlohy a je zbytočné aby sa ukladali obrovské slovníky do každej zálohy. Pri importe úlohy s útokom PRINCE, ktorý je znázornený na obrázku 5.10, bude pridaná kontrola, či slovník, ktorý úloha požaduje, je dostupný v systéme. V prípade jeho nedostupnosti systém vyzve používateľa aby slovník do systému nahral ručne a následne sa úloha úspešne naimportuje do systému.

5.6 Distribúcia útoku PRINCE v komponente Generator

Distribuovanosť útoku bude spočívať v rozdelení úlohy na menšie časti medzi aktívnych klientov. Algoritmus 8 popisuje činnosť generátora pracovných jednotiek na serveri.

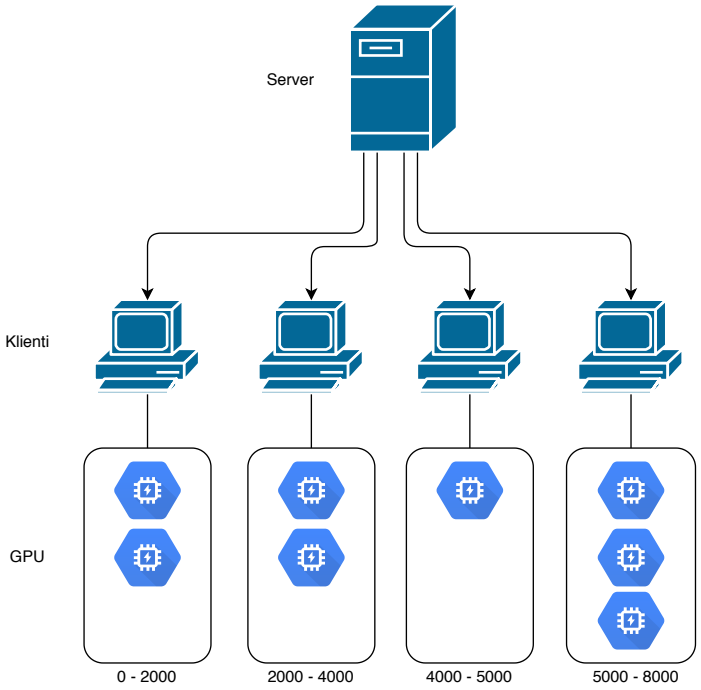
Algoritmus 8: Distribúcia úlohy s útokom PRINCE aktívnym klientom

Vstup: úloha (J), aktívni klienti (AC), slovník (D), konfigurácia PRINCE (CFGP), čas pre pracovnú jednotku (TWU)
Výstup: pracovné jednotky pre klientov

```
1  job_total_ks = get_prince_keyspace(D, CFGP);
2  job_ks_pos = 0;
3  while job_ks_pos < job_total_ks do
4      client = get_free_client(AC);
5      client_power = get_power(client);
6      passwords_in_wu = client_power * TWU;
7      if job_ks_pos + passwords_in_wu > job_total_ks then
8          passwords_in_wu = job_total_ks − job_ks_pos;
9      create_and_send_wu(J, client, job_ks_pos, passwords_in_wu);
10     job_ks_pos = job_ks_pos + passwords_in_wu;
```

Prvým krokom algoritmu je výpočet celkového množstva kandidátnych hesiel, ktoré sa budú skúšať pri počítaní úlohy (*job_total_keyspace*). Následne sa spúšťa hlavný distribučný cyklus, kde v každej iterácii sa vyberie voľný klient z množiny aktívnych klientov, ktoré používateľ vybral pre riešenie danej úlohy. Počet hesiel v pracovnej jednotke (*passwords_in_wu*) sa určí tak, že sa vynásobí sila klienta, tj. koľko hesiel vie vyskúšať za sekundu, časom vyhradeným pre jednu pracovnú jednotku. Aby sa nepresiahol celkový počet kandidátnych hesiel, je nutné túto situáciu ošetriť, a v prípade, že k nej dôjde musí byť počet hesiel v pracovnej jednotke patrične zmenšený. Na základe týchto informácii je možné vytvoriť pracovnú jednotku a odoslať ju klientovi (*create_and_send_wu*). Na záver iterácie sa zvýši celkový počet pridelených hesiel o počet hesiel v tejto pracovnej jednotke. Distribúcia pracovných jednotiek končí, keď počet pridelených hesiel klientom (*job_ks_pos*) sa rovná celkovému množstvu kandidátnych hesiel, čo znamená, že sa rozdelili všetky kandidátne heslá medzi klientov.

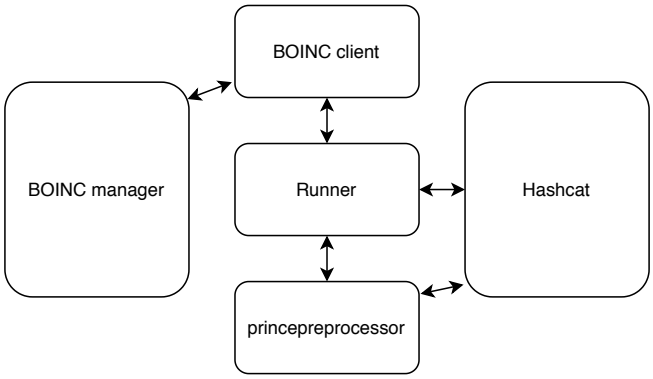
Obrázok 5.11 znázorňuje príklad rozdelenia celej množiny možných hesiel medzi klientov pomocou vyššie uvedeného algoritmu. Výkonnejšie zariadenia dostali väčší rozsah hesiel ako tie menej výkonné.



Obrázok 5.11: Príklad rozdelenia množiny možných hesiel o veľkosťou 8000 medzi 4 aktívnych klientov.

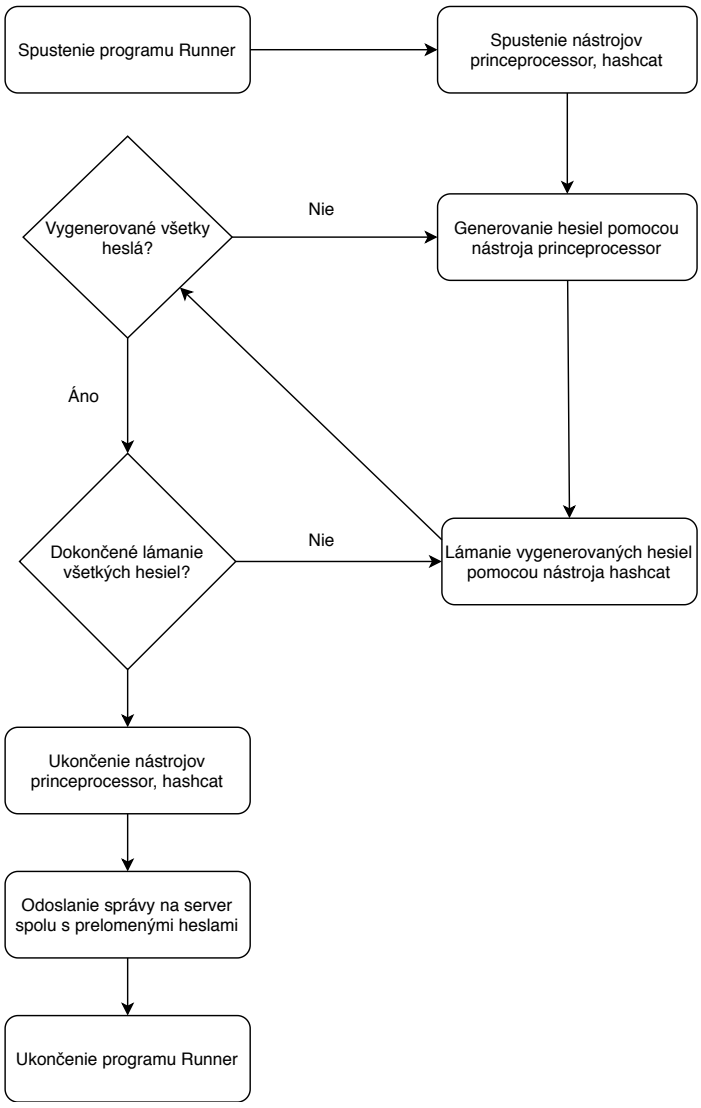
5.7 Podpora útoku PRINCE na strane Fitcrack klienta

Na obrázku 5.12 je znázornená navrhovaná architektúra Fitcrack klienta rozšírená o podporu útoku PRINCE. Bude pridaná nová komponenta - nástroj princeprocessor, ktorý bude slúžiť ako externý generátor možných hesiel implementujúci algoritmus PRINCE. V programe Runner je potrebné pridať podporu pre vytváranie tohto nového útoku. Externý generátor hesiel bude spravovaný programom Runner a jeho výstup bude presmerovaný na vstup nástroja hashcat, ktorý bude vygenerované heslá priebežne lámať.



Obrázok 5.12: Architektúra Fitcrack klienta rozšírená o PRINCE útok

Na obrázku 5.13 je znázornený vývojový diagram, ktorý popisuje navrhnutý spôsob realizácie útoku PRINCE v programe Runner. Po spustení programu Runner sa spustia aj nástroje hashcat a princeprocessor. Priebežne sa budú generovať heslá pomocou nástroja princeprocessor a tieto heslá sa budú postupne lámať pomocou nástroja hashcat. Správy o priebehu lámania budú odosielané na Fitcrack server. Ak sa vygenerovali všetky možné heslá a nástroj hashcat ich už tiež spracoval, dochádza k ukončeniu nástrojov hashcat a princeprocessor. Program Runner vytvorí správu s výsledkom lámania a odošle ju spolu s prípadnými získanými heslami naspäť na server.



Obrázok 5.13: Vývojový diagram fungovania útoku PRINCE v programe Runner.

Kapitola 6

Implementácia rozšírenia systému Fitcrack

Táto kapitola sa zaoberá implementáciou rozšírenia systému Fitcrack, ktoré umožní realizáciu distribuovaného útoku pomocou algoritmu PRINCE. Za účelom rozšírenia systému Fitcrack o podporu útoku pomocou tohto algoritmu bolo nutné vykonať úpravy vo viacerých častiach systému. Obsahom tejto podkapitoly je implementačný popis vykonaných zmien. Novému útoku bolo pridelené identifikačné číslo 8 v rámci systému Fitcrack. Implementovaný útok PRINCE bol začlenený do Github repozitára projektu Fitcrack[1] a je dostupný v systéme Fitcrack od verzie 2.2.0[2], ktorá bola publikovaná v júni 2020. V prílohe B sa nachádza zoznam všetkých pridaných alebo upravených súborov spolu s popisom úprav.

6.1 Tvorba úlohy s útokom PRINCE

V súbore `webadmin/fitcrackFE/src/components/job/addJobView.vue` sa nachádza implementácia webového rozhrania pre tvorbu novej úlohy, ktoré v sebe zahŕňa časť špecifickú pre každý typ útoku. Pre útok PRINCE sa táto časť rozhrania nachádza v súbore `webadmin/fitcrackFE/src/components/job/attacks/prince.vue`, kde je definované rozloženie stránky na konfiguráciu útoku PRINCE. Popis konfiguračných možností útoku PRINCE sa nachádza v tabuľke 6.1.

názov	popis
case_permute	permutácia prvých písmen v slove
check_duplicates	kontrola duplicitných hesiel
min_password_len	minimálna dĺžka kandidátnych hesiel
max_password_len	maximálna dĺžka kandidátnych hesiel
min_elem_in_chain	minimálny počet prvkov v reťazci
max_elem_in_chain	maximálny počet prvkov v reťazci
generate_random_rules	počet náhodných pravidiel

Tabuľka 6.1: Konfiguračné možnosti útoku PRINCE.

[1] https://github.com/nesfit/fitcrack
[2] https://fitcrack.fit.vutbr.cz/download/

U každého vstupného poľa vo webovom rozhraní je uvedené, aký je rozsah platných hodnôt. Systém Fitcrack pri každej zmene hodnoty konfigurácie útoku spúšťa kontrolu platnosti vstupných údajov, ktorá je implementovaná vo funkcii *checkValid*. Pri zistení problému je naň používateľ upozornený informačným panelom v pravej dolnej časti obrazovky. Ak sú zadané údaje správne, webový klient pošle konfiguráciu úlohy na koncový bod `/job/crackingTime`, aby od servera zistil odhadovaný čas trvania úlohy pri danej konfigurácii. Server na základe konfigurácie úlohy musí najskôr vypočítať celkový počet kandidátnych hesiel. Na tento účel bola implementovaná funkcia *compute_prince_keyspace*, ktorá prevedie konfiguráciu úlohy na argumenty pre nástroj princeprocessor a následne ho spustí s prepínačom *--keyspace*. Výstupom nástroja je číslo, ktoré reprezentuje celkový počet kandidátnych hesiel, ktoré sa vrámci tejto úlohy budú skúšať. Na základe tejto hodnoty a sily klientov, ktorí sú priradení k úlohe, sa vypočítava odhadovaný čas trvania úlohy.

Ak je používateľ spokojný s konfiguráciou útoku, môže vytvoriť novú úlohu. Webový klient odošle požiadavku na vytvorenie novej úlohy na server. Na koncovom bode `/job` server prijme konfiguráciu úlohy. Následne vytvorí záznam o novej úlohe v databáze. Aby bolo možné uložiť všetky konfiguračné možnosti útoku PRINCE, bolo nutné rozšíriť tabuľku *fc_job* o nové stĺpce, ktoré sú znázornené vo výpise 6.1.

```
...
‘case_permute‘ tinyint(1) unsigned NOT NULL DEFAULT ’0’,
‘check_duplicates‘ tinyint(1) unsigned NOT NULL DEFAULT ’1’,
‘min_password_len‘ int(10) unsigned NOT NULL DEFAULT ’1’,
‘max_password_len‘ int(10) unsigned NOT NULL DEFAULT ’8’,
‘min_elem_in_chain‘ int(10) unsigned NOT NULL DEFAULT ’1’,
‘max_elem_in_chain‘ int(10) unsigned NOT NULL DEFAULT ’0’,
‘generate_random_rules‘ int(10) unsigned NOT NULL DEFAULT ’0’,
```

Výpis 6.1: Nové stĺpce v tabuľke fc_job

Nové stĺpce som doplnil aj do databázového modelu reprezentujúceho úlohu - trieda *FcJob*. Model sa v SQLAlchemy implementuje ako odvodená trieda od triedy *Base*. Výpis 6.2 popisuje nové položky spolu s ich dátovými typmi a predvolenými hodnotami. Stĺpce tabuľky v modeli popisujú inštancie triedy *Column*, kde ich pomenované argumenty zodpovedajú názvom stĺpcov v MySQL. Práca s modelom potom prebieha v jednotlivých metódach na koncových bodoch, ktoré implementujú požadované akcie.

```
class FcJob(Base):
    __tablename__ = ’fc_job’
    ...
    case_permute = Column(Integer, nullable=False,
                          server_default=text("’0’"))
    check_duplicates = Column(Integer, nullable=False,
                          server_default=text("’1’"))
    min_password_len = Column(Integer, nullable=False,
                          server_default=text("’1’"))
    max_password_len = Column(Integer, nullable=False,
                          server_default=text("’8’"))
    min_elem_in_chain = Column(Integer, nullable=False,
```

```
                    server_default=text("'1'"))
max_elem_in_chain = Column(Integer, nullable=False,
                    server_default=text("'8'"))
generate_random_rules = Column(Integer, nullable=False,
                    server_default=text("'0'"))
...
```

Výpis 6.2: Nové položky v databázový modele úlohy

U každého typu útoku sa do stĺpca *hc_keyspace* v tabuľke *fc_job* ukladá celkový počet kandidátnych hesiel. V prípade útoku PRINCE sa tu uloží hodnota získaná z nástroja princeprocessor. Ak sa nepoužívajú pravidlá, do stĺpca *keyspace* v tabuľke *fc_job* sa uloží hodnota *hc_keyspace*, inak sa do *keyspace* uloží hodnota *hc_keyspace* vynásobená počtom použitých pravidiel typicky - počet riadkov v súbore s pravidlami.

6.2 Informácie o úlohe s útokom PRINCE

Do súčasného webového rozhrania s informáciami o úlohe bola pridaná podpora pre zobrazenie hodnôt konfiguračných možností špecifických pre útok PRINCE. Zobrazenie informácii o úlohe sa nachádza v `webadmin/fitcrackFE/src/components/jobDetail/jobInfo.vue`. Funkciu *attackDetailComponent* v tomto súbore som upravil tak, aby v prípade úlohy s útokom PRINCE zobrazila webovú komponentu *princeDetail*, ktorá je implementovaná v súbore `webadmin/fitcrackFE/src/components/jobDetail/attacks/prince.vue`. Jedná sa o šablónu, kde je definované rozloženie jednotlivých prvkov grafického rozhrania. Vyplnením tejto šablóny údajmi priamo z úlohy vzniká časť stránky s informáciami o úlohe, ktorá je pre každý útok navrhnutá a implementovaná individuálne.

6.3 Editácia úlohy s útokom PRINCE

Existujúce webové rozhranie určené na editáciu úlohy, ktoré je implementované v súbore `webadmin/fitcrackFE/src/components/jobDetail/jobEditor.vue` bolo rozšírené o nové vstupné textové polia, kde používateľ môže zmeniť konfiguráciu útoku PRINCE. Po otvorení editora úlohy sa načítajú aktuálne hodnoty nastavení úlohy. Zmenené hodnoty je potrebné uložiť pomocou tlačidla Uložiť. Webový klient pošle novú konfiguráciu na koncový bod `/job/<id úlohy>`. Na strane servera prebieha kontrola platnosti novej konfigurácie. V prípade problému server odosiela webovému klientovi správu s informáciou o zistenej chybe. V opačnom prípade sa kontroluje, či došlo k zmene v konfiguračných možnostiach, ktoré ovplyvňujú celkový počet kandidátnych hesiel. Ak áno, server pomocou nástroja princeprocessor prepočíta túto hodnotu na základe novej konfigurácie útoku v úlohe a uloží ju do databázy. Ak nevznikol žiadny problém pri výpočte tejto hodnoty, je celá nová konfigurácia uložená do databázy. Po opätovnom spustení úlohy sa už budú používať nové hodnoty nastavení.

6.4 Rozšírenie filtra v zozname úloh

Do zoznamu typov útokov, ktorý je uložený v premennej *jobs_types* nachádzajúcej sa v súbore `webadmin/fitcrackFE/src/components/job/jobsView.vue`, bola pridaná položka

„prince“ pre útok PRINCE. V serverovej časti bol upravený koncový bod `/job` aby previedol reťazec „prince“ z filtrovacej požiadavky na kód útoku 8. Následne sa z databáze načítajú iba úlohy s útokom PRINCE a tento zoznam úloh server posiela webovej aplikácii vo formáte JSON. Po prijatí tohto zoznamu sa vo webovom klientovi zobrazia len úlohy s útokom PRINCE v zozname úloh.

6.5 Podpora útoku PRINCE pri exporte / importe úloh

Pridanie tejto podpory predstavovalo úpravu v serverovej časti, a to konkrétne v súbore `webadmin/fitcrackAPI/src/src/api/fitcrack/endpoints/serverInfo/transfer.py`. Zoznam *JOB_EXPORTABLE_COLUMNS* obsahuje stĺpce, ktoré sa majú exportovať. Do tohto zoznamu som pridal nové položky, ktoré súvisia s útokom PRINCE:

- *min_password_len*,
- *max_password_len*,
- *min_elem_in_chain*,
- *max_elem_in_chain*,
- *case_permute*,
- *check_duplicates*,
- *generate_random_rules*.

Popis týchto položiek je uvedený v tabuľke 6.1.

6.6 Distribúcia útoku PRINCE v komponente Generator

Nasledovné úpravy boli prevedené v serverovej komponente Generator.

Nová trieda CAttackPrince

Trieda implementuje virtuálnu metódu *makeWorkunit*, ktorá slúži na vytvorenie jednej pracovnej jednotky s daným typom útoku. Metóda sa skladá z nasledovných logických častí:

- **Generovanie pracovnej jednotky**

  Na základe sily klienta, ktorému bude pracovná jednotka pridelená, a na základe času vyhradeného na pracovnú jednotku sa vypočíta počet hesiel, ktoré sa majú spracovať v danej pracovnej jednotke. Následne sa z databáze získa hodnota reprezentujúca celkový počet kandidátnych hesiel u danej úlohy a hodnota reprezentujúca pozíciu vrámci úlohy, tj. počet už rozdelených hesiel medzi klientov. Ak by súčet počtu už rozdelených hesiel a počtu hesiel v aktuálnej pracovnej jednotke presiahol celkový počet kandidátnych hesiel, počet hesiel v pracovnej jednotke sa automaticky zmenší. Následne sa vygeneruje pracovná jednotka a zvýši sa počet rozdelených hesiel o počet hesiel v práve vygenerovanej pracovnej jednotke.

- **Vytvorenie konfiguračného súboru**

  Na základe konfigurácie útoku PRINCE uloženej v databáze sa vytvorí textový konfiguračný súbor, ktorý bude obsahovať kópiu týchto informácii pre potreby klienta. V tomto kroku je najzaujímavejšie generovanie konfiguračných položiek *skip_from_start* a *dict_hc_keyspace*. Hodnota *skip_from_start* reprezentujúce pozíciu vrámci úlohy a udáva, koľko hesiel sa má preskočiť pri generovaní pomocou nástroja princeprocessor. Hodnota *dict_hc_keyspace* udáva počet hesiel, ktoré si klient má vygenerovať a ktoré bude následne lámať.

- **Vytvorenie súboru s hešami**

  Systém vytvorí súbor pre klienta, ktorý bude obsahovať heše z databázy, ktoré sa majú vrámci úlohy lámať.

- **Vytvorenie slovníka pre úlohu**

  Pri prvej pracovnej jednotke sa vytvára dočasný slovník pre úlohu. Ak používateľ vybral iba jeden slovník, obsah sa iba prekopíruje. V prípade, že vybral viacero slovníkov, obsahy týchto slovníkov sa spoja do tohto dočasného slovníka. U ďalších pracovných jednotiek sa pracuje s dočasným slovníkom a nedochádza k neustálemu kopírovaniu hesiel medzi slovníkmi. Po odoslaní poslednej pracovnej jednotky sa dočasný slovník maže.

- **Vytvorenie súboru s pravidlami**

  Ak používateľ si u úlohy nastavil, že chce použiť pravidlá, súbor s pravidlami sa pridá do pracovnej jednotky.

- **Registrácia pracovnej jednotky v systéme BOINC**

  V systéme BOINC vznikne nová výpočtová úloha odpovedajúca vytvorenej pracovnej jednotke. Pomocou šablóny *prince_in*, resp. *prince_rules_in*, sa špecifikujú súbory, ktoré sa majú preniesť na klienta. V šablóne je u slovníka a u súboru s pravidlami nastavený príznak *sticky*, ktorý informuje systém BOINC o tom, že tieto súbory sa medzi pracovnými jednotkami nemenia a stačí ich preniesť na klienta iba raz v rámci danej úlohy.

Úprava triedy SimpleGenerator

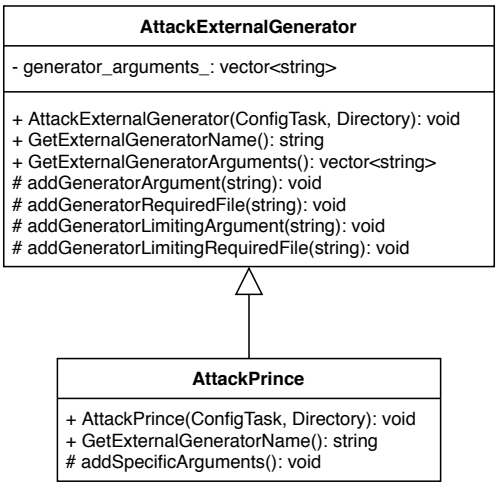
Následne je nutné vytvoriť útok PRINCE (inštancia triedy CAttackPrince) v metóde *CreateAttack* triedy *SimpleGenerator*. V tejto metóde sa podľa kódu útoku vytvorí odpovedajúci útok, pre útok PRINCE sa overuje, či kód útoku v úlohe je 8.

6.7 Podpora útoku PRINCE na strane Fitcrack klienta

Pred začatím implementácie rozšírenia bolo nutné previesť refaktorizáciu kódu a upraviť niektoré rozhrania, aby boli univerzálnejšie a umožňovali jednoduchú integráciu nových útokov. Obsahom tejto podkapitoly je popis úprav v programe Runner potrebných pre integráciu nového typu útoku. Okrem implementácie nových tried bolo potrebné upraviť aj niektoré existujúce metódy a pridať do nich podporu pre útok pomocou algoritmu PRINCE.

Nová trieda AttackPrince

Novo pridaná trieda, ktorá zapuzdruje vytváranie útoku pomocou algoritmu PRINCE. Diagram tejto triedy je znázornený na obrázku 6.1. Dedí od triedy `AttackExternalGenerator` a volá jej konštruktor. Na základe konfiguračných dát a informácii o aktuálnom priečinku pripravuje argumenty (*addSpecificArguments*), s ktorými bude neskôr spustený nástroj princeprocessor. Mapovanie konfiguračných možností na argumenty tohto nástroja je znázornené v tabuľke 6.2. V aktuálnom priečinku sa hľadá súbor *dict1*, ktorý bude použitý ako vstupný slovník pre PRINCE. Z konfiguračného súboru sa načítajú informácie o pracovnej jednotke. Konfiguračné možnosti útoku PRINCE sa prevedú na argumenty, s ktorými sa následne spustí nástroj princeprocessor. V triede je taktiež implementácia virtuálnej funkcie *getExternalGeneratorName*, ktorá je používaná na získanie názvu externého generátora - „princeprocessor“.



Obrázok 6.1: Nová trieda AttackPrince a jej umiestnenie v triednej hierarchii.

názov	prepínač nástroja princeprocessor
max_password_len	*--pw-max*
min_password_len	*--pw-min*
max_elem_in_chain	*--elem-cnt-max*
min_elem_in_chain	*--elem-cnt-min*
case_permute	*--case-permute*
check_duplicates	*--dupe-check-disable*
skip_from_start	*--skip*
dict_hc_keyspace	*--limit*

Tabuľka 6.2: Mapovanie konfiguračných možnosti útoku PRINCE na argumenty nástroja princeprocessor.

Úpravy v triede TaskComputeBase

V metóde *initialize* prebieha spúšťanie externých generátorov (v súčasnosti len *pcfg-manager*), takže kód metódy bol rozšírený aby v prípade PRINCE vytvoril proces pre nástroj princeprocessor. Systém Fitcrack integruje staticky skompilované binárne verzie princeprocessor pre operačné systémy Linux a Windows. Statická metóda *create*) v triede *Process* má za úlohu vyhľadať správny binárny súbor podľa typu operačného systému a následne vytvoriť nový proces.

Následne bolo potrebné prepojiť vstup nástroja hashcat na výstup nástroja princeprocessor. Ako bolo spomenuté v návrhu, jednoduchým riešením by bolo najskôr vygenerovať možné heslá do súboru a následne spustiť slovníkový útok pomocou nástroja *hashcat* s týmto vygenerovaným súborom, ako so vstupným slovníkom. Toto riešenie nie je efektívne z dôvodu, že zatiaľ čo sa generujú nové a nové heslá, je možné priebežne lámať tie, ktoré sú už vygenerované. Z tohto dôvodu je vhodnejším riešením tohto problému prepojiť nástroje princeprocessor a hashcat cez rúry (*pipes*). Potrebná infraštruktúra na toto prepojenie už bola v programe Runner implementovaná, čiže nutnou úpravou bolo povolenie prepojenia výstupu generátora na vstup nástroju hashcat cez rúry aj pri útoku pomocou algoritmu PRINCE. Poslednou úpravou bolo rozšírenie metódy `startComputation`, aby v prípade útoku pomocou algoritmu PRINCE sa spustil nástroj princeprocessor so správnymi argumentami.

Úpravy v triede ConfigTask

Metódu *initSupported* v triede *ConfigTask* bolo potrebné upraviť, aby si systém do zoznamu podporovaných konfiguračných možností pridal možnosti, ktoré sú špecifické pre útok PRINCE. Tieto nové možnosti sú znázornené v tabuľke 6.3 spolu s dátovými typmi. Bez tejto úpravy by program Runner hlásil, že vstupný konfiguračný súbor je poškodený a obsahuje neznáme konfiguračné možnosti.

názov	popis	dátový typ
max_password_len	maximálna dĺžka kandidátnych hesiel	UInt
min_password_len	minimálna dĺžka kandidátnych hesiel	UInt
max_elem_in_chain	maximálny počet prvkov v reťazci	UInt
min_elem_in_chain	minimálny počet prvkov v reťazci	UInt
case_permute	permutácia prvých písmen v slove	UInt
check_duplicates	kontrola duplicitných hesiel	UInt
skip_from_start	pozícia, od ktorej začať generovať heslá	BigUInt
generate_random_rules	počet náhodných pravidiel	UInt

Tabuľka 6.3: Konfiguračné možnosti útoku PRINCE a ich dátové typy.

6.8 Testovanie implementácie

Na nové či upravené časti systému Fitcrack som navrhol a implementoval testy, ktoré majú za úlohu zaručiť, že ďalšie zmeny v systéme Fitcrack v budúcnu nespôsobia nefunkčnosť útoku PRINCE. Obsahom tejto podkapitoly je popis nových testov pre útok PRINCE a ich použitie.

6.8.1 Integračné testy

Systém Fitcrack v súčasnosti je testovaný pomocou integračných testov, kde u každého typu útoku je uvedených niekoľko konfigurácií úloh a hľadané heslá vo forme hešov. U každého hesla / hešu je ďalej uvedené, či sa má nájsť alebo nie. Tieto testy nie sú automatizované a je nutné ručne vytvoriť úlohy podľa testovanej konfigurácie útoku a s danými hešami. Kontrola výsledku testu prebieha taktiež ručne, porovnaním výstupu testu s tabuľkou, kde sú uvedené očakávané výsledky daného testu.

Pre útok PRINCE som vytvoril niekoľko konfigurácií úloh, ktoré sú uvedené v tabuľke 6.4. Pre každú konfiguráciu som vybral niekoľko hesiel vo forme SHA-1 (*Secure Hash Algorithm 1*) hešov, ktoré sa pri správnej funkčnosti útoku PRINCE majú alebo nesmú nájsť. Tieto heslá spolu s očakávaným výsledkom testu danej konfigurácie je možné nájsť v tabuľkách 6.5 až 6.7. Ostatné možnosti, ktoré je možné u úlohy nastaviť, boli ponechané na predvolených hodnotách.

konfigurácia	1.	2.	3.
slovník	honeynet.txt	honeynet.txt	adobe100.txt
pravidlá	nie	nie	toggles1.rule
minimálna dĺžka hesiel	4	6	4
maximálna dĺžka hesiel	7	7	7
minimálny počet prvkov v reťazci	1	2	1
maximálny počet prvkov v reťazci	2	2	2
permutácia prvého písmena každého slova	nie	áno	nie

Tabuľka 6.4: Testované konfigurácie úloh s útokom PRINCE.

son	john	heslo	oracleoracleoracle	oracle	sunsunsunsunsun	sunjohn
	✓	✓		✓		✓

Tabuľka 6.5: Vstupné heslá a očakávaný výsledok ich lámania pri konfigurácii č. 1.

Test č. 1 sa zameriava na overenie funkčnosti konfiguračných možností týkajúcich sa obmedzenia dĺžky kandidátnych hesiel a počtu prvkov v reťazci, tj. v kandidátnom hesle. Očakávané výsledky pre tento test sú uvedené v tabuľke 6.5. Heslá *john*, *heslo*, *oracle* a *sunjohn* sa nachádzajú vo vstupnom slovníku, ich dĺžka patrí do povoleného rozsahu a sú zložené s 1 alebo 2 prvkov, tj. slov zo vstupného slovníka, a preto je u nich očakávaným výsledkom, že sa majú nájsť. Heslá *oracleoracleoracle* a *sunsunsunsunsun* sa nájsť nesmú, keďže ich dĺžka je väčšia ako 7, čo je maximálna dĺžka hesiel nastavená v tejto konfigurácii.

SunSun	Sunjohn
✓	✓

Tabuľka 6.6: Vstupné heslá a očakávaný výsledok ich lámania pri konfigurácii č. 2.

V teste č. 2 sa kladie hlavný dôraz na overenie funkčnosti konfiguračnej možnosti na povolenie alebo zakázanie permutácie prvého písmena u každého slova zo vstupného slovníka. Očakávané výsledky pre tento test sú uvedené v tabuľke 6.6. Pri správnej funkčnosti testovanej konfiguračnej možnosti sa heslá *SunSun* a *Sunjohn* nájsť majú, keďže slová *sun* a *john* sa nachádzajú vo vstupnom slovníku, a následnou permutáciou prvého písmena týchto slov

vznikli nové slová *Sun* a *John*. Spojením slov *Sun* a *Sun*, resp. *Sun* a *john*, vzniknú hľadané heslá.

testAbc	teSTabc	TESTABC	aBctest
✓			✓

Tabuľka 6.7: Vstupné heslá a očakávaný výsledok ich lámania pri konfigurácii č. 3.

Test č. 3 overuje funkčnosť útoku PRINCE s pravidlami pre hashcat. Súbor s pravidlami *toggles1.rule* obsahuje pravidlá, ktoré z každého kandidátne hesla vygenerovaného pomocou algoritmu PRINCE vytvoria ďalšie tak, že zmenia veľkosť práve jedného znaku, tj. malé písmeno sa zmení na veľké alebo opačne. Slová *test* a *abc* sa nachádzajú vo vstupnom slovníku a na základe konfigurácie budú algoritmom PRINCE vygenerované heslá *test*, *testabc* a *abctest*. Následne po aplikácii pravidiel v nástroji hashcat tesne pred procesom ich lámania vznikli ďalšie heslá, ako napríklad *Testabc*, *tEstabc* a iné. V tabuľke 6.7 so vstupnými heslami sú uvedené heslá *teSTabc* a *TESTABC*, ktoré sa avšak nájsť nesmú, keďže použité pravidlá nemohli vytvoriť tieto heslá. Heslá *testAbc* a *aBctest* sa nájsť majú, keďže vznikli zmenou veľkosti práve jedného písmena v heslách *testabc* a *abctest*.

6.8.2 Testy klientského programu Runner

Systém Fitcrack obsahuje základné testy pre rôzne časti systému [5] a jednou z týchto častí je aj klientský program Runner. Testy pre program Runner je možné spustiť pomocou príkazu `python3 -m unittest test_runner.TestRunner` v adresári *automaticke-testy*, viď príloha A. K súčasnej sade testov som pridal ďalšie testy viažúce sa na útok PRINCE. Cieľom týchto testov je overiť, či program Runner spúšťa nástroj hashcat so správnymi argumentami, ktoré sú určené podľa konfiguračného súboru, ktorý je súčasťou každej pracovnej jednotky. Na základe navrhnutého riešenia, kde výstup nástroja princeprocessor sa presmerováva na vstup nástroja hashcat, je potrebné v testoch overiť, že pri útoku PRINCE Runner spustí hashcat v režime slovníkového útoku. Okrem overenia správnosti argumentov sa overujú aj výstupy a návratové kódy programu Runner.

Okrem týchto základných testov som navrhol ďalšie a špecifickejšie testy pre útok PRINCE, ktoré sú implementované v skripte `runner/testground/prince_tester.py`. Účelom tohto testovacieho skriptu je overenie funkčnosti programu Runner z pohľadu načítania konfiguračného súboru s útokom PRINCE a následnej kontroly, či akcie spustené týmto programom majú požadované výstupy. Vo funkcii `run_prince_tests` sa nachádza niekoľko testov, ktoré overujú funkčnosť konfiguračných možností algoritmu PRINCE. Každý test sa skladá z niekoľkých fáz: vytvorenie vstupného slovníka, vytvorenie súboru s MD5 hešmi pre hľadané heslá, vytvorenie konfiguračného súboru obsahujúceho nastavenia pre algoritmus PRINCE, zavolania programu Runner a následnej kontroly, či výstup lámania obsahuje heslá, ktoré sa hľadajú.

6.8.3 Testy aplikačného rozhrania

Existujúce testy aplikačného rozhrania [5], ktoré testujú rôzne koncové body serverovej časti, som rozšíril o nové testy, ktoré overujú správne spracovanie útoku PRINCE na serveri. Testy API je možné spustiť pomocou príkazu `python3 -m unittest test_api.py` v adresári *automaticke-testy*, viď príloha A. Cieľom nových testov sú koncové body pre vytvorenie, úpravu a zmazanie úloh. Pridal som tri testy, kde prvý test reprezentuje zá-

kladne použitie útoku, v druhom teste je použitá možnosť na obmedzenie celkového počtu kandidátnych hesiel a v tretom teste boli použité aj pravidlá.

V každom teste sa vytvorí požiadavka na vytvorenie úlohy s útokom PRINCE s jej určitou konfiguráciou. Následne sa táto požiadavka pošle na serverovú časť, ktorý ju spracuje a vykoná rôzne akcie. Pri vytváraní úlohy prebieha výpočet celkového počtu kandidátnych hesiel na základe konfigurácie úlohy, ktorý sa spolu s inými informáciami o úlohe uloží do databázy. V testoch sa overuje, či vypočítaný počet kandidátnych hesiel sa zhoduje s hodnotou, ktorú očakávame. V prípade použitia konfiguračnej možnosti na ručné obmedzenie celkového počtu kandidátnych hesiel je nutné, aby celkový počet kandidátnych hesiel v databáze sa zhodoval s týmto ručne nastaveným limitom.

Ďalej sa kontroluje zhodu konfigurácie úlohy z požiadavky s tou, ktorá je uložená v databáze. Ak všetky overenia uspejú, posiela sa nová požiadavka na úpravu úlohy s inou konfiguráciou. Následne prebieha kontrola, či nová konfigurácia bola správne uložená do databázy. Dôležitou kontrolou je taktiež kontrola prepočtu celkového počtu kandidátnych hesiel v databáze na základe novej konfigurácie. Poslednou časťou testu je overenie, či je po odoslaní požiadavku o zmazanie úlohy táto úloha skutočne vymazaná.

Kapitola 7

Experimenty

Obsahom tejto kapitoly je popis navrhnutých experimentov, namerané výsledky a ich vyhodnotenie. Po dokončení a otestovaní implementovaného riešenia pre distribuovaný útok PRINCE bola navrhnutá sada experimentov, v ktorej sa experimenty primárne zameriavajú na kľúčové faktory relevantné pre útoky na lámanie hesiel, medzi ktoré patrí efektivita, doba, réžia a škálovateľnosť výpočtov. Hodnoty týchto faktorov boli získané zo systému Fitcrack, kde u každej úlohy sa tieto informácie zobrazujú. Útok PRINCE som taktiež porovnával s inými typmi útokov, a to konkrétne so slovníkovým a kombinačným útokom. Posledná časť experimentov sa venovala porovnaniu implementácií útoku PRINCE medzi systémami Fitcrack a Hashtopolis.

U každého experimentu sú uvedené konkrétne hodnoty nastavení, s ktorými bol experiment vykonaný. U ostatných konfiguračných možností, ktoré nie sú u experimentu explicitne spomenuté, boli použité predvolené hodnoty. Experimenty boli vykonávané na počítačoch v laboratóriu C304 na FIT VUT v Brne. Hardvérová konfigurácia všetkých počítačov bola rovnaká a je uvedená v tabuľke 7.1. Systém Fitcrack vo verzii 2.2.0 bol nainštalovaný na inom školskom počítači, ktorého hardvérová konfigurácia sa nachádza v tabuľke 7.2. Tento počítač slúžil ako Fitcrack server.

Operačný systém	Windows 7
CPU	Intel Core i5-3570K, 3,4 GHz
GPU	Nvidia GTX 1050 Ti
RAM	DDR3, 8 GB

Tabuľka 7.1: Hardvérová konfigurácia počítačov použitých na experimenty.

Operačný systém	Ubuntu 18.04.4 LTS
CPU	AMD Ryzen 5 2600X, 3,6 GHz
GPU	Nvidia RTX 2080, GTX 1050 Ti
RAM	DDR4, 16 GB

Tabuľka 7.2: Hardvérová konfigurácia servera použitého na experimenty.

V experimentoch som prevažne pracoval so slovníkom *rockyou.txt*, ktorý obsahuje 14 341 564 unikátnych hesiel. Tieto heslá patria k 32 603 388 používateľským účtom zo stránky rockyou.com. V prípadoch, kde z časových alebo pamäťových dôvodov nebolo možné použiť

tento slovník, som použil slovník *adobe100.txt*. Tento slovník obsahuje 100 používateľských hesiel, ktoré unikli spoločnosti Adobe.

7.1 Experimenty s konfiguračnými možnosťami PRINCE

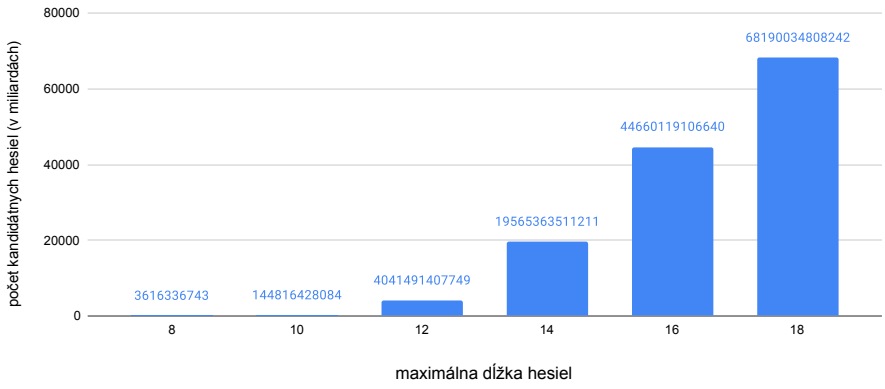
V experimentoch som sa zameral na analýzu vplyvu zmeny konfiguračných možností algoritmu PRINCE na veľkosť množiny kandidátnych hesiel, od ktorej sa ďalej odvíja čas potrebný na výpočet úlohy. Cieľom útoku boli SHA-1 heše získané pre 100 najpoužívanejších hesiel v službe Linkedin. U každého experimentu v tejto podkapitole je uvedené, koľko by trval danej výpočet v nedistribuovanom a distribuovanom prostredí. Distribuované prostredie obsahovalo desať klientov.

7.1.1 Vplyv zmeny maximálnej dĺžky hesiel

V tabuľke 7.3 sú uvedené hodnoty nastavení, ktoré boli použité pri tomto experimente. Jedinou meniacou sa hodnotou bola maximálna dĺžka hesiel.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
minimálny počet prvkov v reťazci	1
minimálny počet prvkov v reťazci	2
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.3: Hodnoty použitých nastavení v experimente.



Obrázok 7.1: Vplyv zmeny maximálnej dĺžky hesiel na množstvo kandidátnych hesiel.

Na grafe 7.1 je možné vidieť výsledok tohto experimentu, kde pri zvyšujúcej sa maximálnej dĺžke hesiel postupne rastie množstvo kandidátnych hesiel vygenerovaných algoritmom PRINCE. Tabuľka 7.4 obsahuje dobu trvania úlohy pri danej maximálnej dĺžke hesiel s jed-

ným a s desiatimi klientami. Od maximálnej dĺžky hesiel 12 je v tabuľke uvedený odhad predpokladaného času potrebného na výpočet úlohy, ktorý bol získaný zo systému Fitcrack.

maximálna dĺžka hesiel	doba výpočtu s 1 klientom	doba výpočtu s 10 klientami
8	4 minúty	1 minúta
10	3 hodiny, 9 minút	37 minút
12	3 dni, 16 hodín	9 hodín
14	19 dní	2 dni
16	40 dní	6 dni

Tabuľka 7.4: Vplyv zmeny maximálnej dĺžky hesiel na dobu výpočtu úlohy s jedným a s desiatimi klientami.

Získané výsledky so slovníkom *rockyou* ukazujú, že útok PRINCE v nedistribuovanom prostredí prestáva byť v praxi použiteľný pri maximálnej dĺžke hesiel nad 10 znakov, keďže časová náročnosť úlohy sa mení z pár hodín na niekoľko dní.

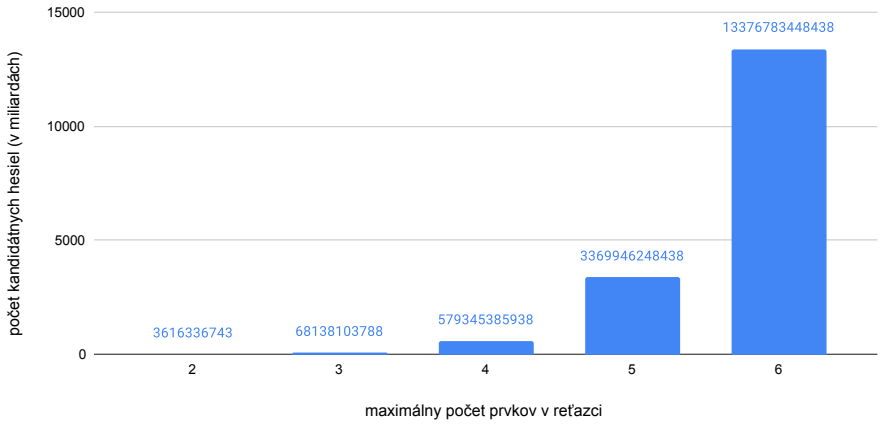
7.1.2 Vplyv zmeny maximálneho počtu prvkov v reťazci

V tabuľke 7.5 sú uvedené hodnoty nastavení, ktoré boli použité pri tomto experimente. Jedinou meniacou sa hodnotou bol maximálny počet prvkov v reťazci. Maximálna dĺžka hesiel bola zvolená 8, keďže väčšina hesiel má dĺžku do 8 znakov, čo ukazuje aj obrázok 4.1, ktorý zobrazuje distribúciu dĺžok hesiel v slovníku *rockyou*, ktorý obsahuje najčastejšie používané heslá.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.5: Hodnoty použitých nastavení v experimente.

Na grafe 7.2 je možné vidieť výsledok tohto experimentu, kde pri zvyšujúcom sa maximálnom počte prvkov v reťazci strmo rastie množstvo kandidátnych hesiel vyprodukovaných algoritmom PRINCE. Tabuľka 7.6 obsahuje dobu trvania úlohy s daným maximálnym počtom prvkov v reťazci s jedným a s desiatimi klientami.
Výsledky tohto experimentu ukazujú, že pri maximálne 4 prvkoch v reťazci a maximálnej dĺžke hesiel 8 je nedistribuovaný útok PRINCE použiteľný aj v praxi, keďže útok trvá len niekoľko hodín. V distribuovanom prostredí je možné použitie aj vyššieho počtu prvkov v reťazci. Na druhej strane, vyšší počet nemusí priniesť želané úspechy v lámaní, keďže väčšina hesiel používateľov je v praxi krátka a na ich prelomenie by postačil aj nižší počet prvkov v reťazci.



Obrázok 7.2: Vplyv zmeny maximálneho počtu prvkov v reťazci na množstvo kandidátnych hesiel.

max. počet prvkov v reťazci	doba výpočtu s 1 klientom	doba výpočtu s 10 klientami
2	4 minúty	1 minúta
3	1 hodina, 29 minút	7 minút
4	12 hodín, 39 minút	1 hodina, 23 minút
5	3 dni, 1 hodina	7 hodín, 28 minút

Tabuľka 7.6: Vplyv zmeny maximálneho počtu prvkov v reťazci na dobu výpočtu úlohy s jedným a s desiatimi klientami.

7.1.3 Vplyv permutácií prvých písmen slov

V tabuľke 7.7 sú uvedené hodnoty nastavení, ktoré boli použité pri tomto experimente. Jedinou meniacou sa hodnotou bola permutácia prvých písmen slov. Maximálna dĺžka hesiel bola zvolená 8 a maximálny počet prvkov v reťazci 4, čo sú hodnoty, ktoré sa na základe predchádzajúcich experimentov ukazujú ako vhodné pre nedistribuovaný útok PRINCE.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.7: Hodnoty použitých nastavení v experimente.

Výsledky pre tento experiment sú uvedené v tabuľkách 7.8 a 7.9. Povolenie tejto konfiguračnej možnosti u útoku PRINCE nie je vhodné v nedistribuovanom prostredí, keďže

permutácia prvých písmen	počet kandidátnych hesiel
bez permutácie	579 345 385 938
s permutáciou	4 983 723 470 383

Tabuľka 7.8: Vplyv permutácií prvých písmen slov na celkový počet kandidátnych hesiel.

permutácia prvých písmen	doba výpočtu s 1 klientom	doba výpočtu s 10 klientami
bez permutácie	12 hodín, 39 minút	1 hodina, 21 minút
s permutáciou	4 dni, 12 hodín	4 hodiny, 10 minút

Tabuľka 7.9: Vplyv permutácií prvých písmen slov na dobu výpočtu úlohy s jedným a s desiatimi klientami.

výrazne zvyšuje dobu výpočtu úlohy. V distribuovanom prostredí rozdiel nie je tak výrazný. V prípade, že vysoká spotreba energie pri desiatich klientoch nie je problémom, môže táto možnosť značne zvýšiť silu útoku, keďže sa generuje oveľa viac kandidátnych hesiel. U tohto experimentu sa bez povolenej permutácie prvých písmen podarilo prelomiť 26 SHA-1 heší, s povolenou permutáciou sa prelomilo 27 heší.

7.1.4 Experimenty s kontrolou duplicitných hesiel

Cieľom tohto experimentu bolo preskúmanie konfiguračnej možnosti útoku PRINCE, ktorá umožňuje povoliť alebo zakázať kontrolu duplicitných kandidátnych hesiel vo vstupnom slovníku pre PRINCE. Hodnoty nastavení útoku PRINCE, ktoré boli použité pre tento experiment a ktoré sa nemenili, sú uvedené v tabuľke 7.10.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.10: Hodnoty použitých nastavení v experimente.

kontrola duplicít	doba výpočtu s 1 klientom	doba výpočtu s 10 klientami
povolená	3 dni, 20 hodín	8 hodín, 47 minút
zakázaná	3 dni, 20 hodín	8 hodín, 44 minút

Tabuľka 7.11: Výsledky experimentu s povolenou / zakázanou kontrolou duplicitných hesiel s jedným a s desiatimi klientami so slovníkom bez duplicít.

V prvom experimente, ktorý neobsahuje žiadne duplicitné heslá, ako vstupný a pri danej konfigurácii útoku celkový počet kandidátnych hesiel bol 579 345 385 938. Následne som spustil túto úlohu bez a s kontrolou duplicitných hesiel. Namerané výsledky sú uvedené

v tabuľke 7.11 a poukazujú na to, že réžia spôsobená touto kontrolou je takmer zanedbateľná z pohľadu efektivity a doby lámania a nespôsobuje žiadny výkonnostný problém u úloh, kde vstupný slovník obsahuje unikátne heslá.

V druhom experimente som sa zameral na vplyv duplicitných hesiel na efektivitu a dobu lámania. Výskyt duplicitných hesiel vo všeobecnosti nemusí byť až tak nereálny, najmä ak sa pre lámanie použije viac slovníkov, ktoré sa spoja do jedného vstupného slovníka pre PRINCE. Pre tento experiment som vytvoril slovník *rockyouwithdups.txt*, ktorý obsahoval 10 % duplicitných hesiel. U prvej úlohy som kontrolu duplicitných hesiel. Počet kandidátnych hesiel bol 579 345 385 938, čo je rovnaký počet ako v predchádzajúcom experimente a duplicitné heslá boli ignorované. U druhej úlohy som taktiež použil slovník *rockyouwithdups.txt*, no kontrola duplicitných hesiel u tejto úlohy ostala vypnutá a celkový počet kandidátnych hesiel bol až 1 700 349 609 369. Výsledky z týchto úloh sú znázornené v tabuľke 7.12, kde je možné vidieť značný rozdiel u doby lámania medzi týmito dvoma úlohami. Experiment preukázal, že ak existuje šanca, že vstupný slovník obsahuje duplicitné heslá, je povolenie tejto kontroly odporúčané a dokáže ušetriť veľa času a výpočtových zdrojov.

kontrola duplicít	doba výpočtu s 10 klientami
povolená	8 hodín, 49 minút
zakázaná	1 deň, 4 hodiny

Tabuľka 7.12: Výsledky experimentu s povolenou / zakázanou kontrolou duplicitných hesiel s jedným a s desiatimi klientami so slovníkom s duplicitami.

7.2 Experimenty skúmajúce škaľovatelnosť útoku

Pri lámaní hesiel je použitý hešovací algoritmus najvýraznejší faktor, ktorý ovplyvňuje výpočtovú náročnosť úlohy [30]. Z tohto dôvodu som sa rozhodol, že si vyberiem jedného zástupcu z pomalých a rýchlych hešovacích algoritmov a navrhnuté experimenty s útokom PRINCE vykonám dvakrát, raz pre pomalý a raz pre rýchly hešovací algoritmus. Sadu navrhnutých experimentov som postupne spustil s rôznym počtom klientov a sledoval som priebeh a distribúciu výpočtov. Okrem skúmania škálovateľnosti útokov som sa v experimentoch zameral aj na iné faktory s dôrazom na efektivitu a dobu výpočtu úloh.

Experimenty s pomalým hešovacím algoritmom

Experimenty s pomalým hešovacím algoritmom boli vykonávané s 1, 2 a 4 klientami. Ako zástupcu pomalých heší som si vybral populárny hešovací algoritmus bcrypt, ktorý obsahuje kryptografickú soľ. Cieľom útoku boli bcrypt heše získané pre 100 najpoužívanejších hesiel v službe Linkedin. Hodnoty nastavení útoku PRINCE boli nastavené podľa tabuľky 7.13 a výsledný počet kandidátnych hesiel vygenerovaných algoritmom PRINCE bol 10 000.
Získané výsledky experimentov sú zobrazené v tabuľke 7.14. V experimente s 2 klientami bolo 4600 hesiel pridelených prvému klientovi a 5400 druhému. V prípade so 4 klientami bolo prvým trom klientom pridelených 2700 hesiel a poslednému 1900. Výsledky ukazujú, že útok PRINCE je spravodlivo distribuovaný systémom Fitcrack medzi klientov. Zatiaľ čo pri 1 a 2 klientoch je efektivita útoku veľmi vysoká a réžia je takmer zanedbateľná, u experimentu so 4 klientami došlo k značnému poklesu efektivity. U všetkých troch experimentoch bol súčet časov všetkých pracovných jednotiek približne 9 hodín a 50 minút, čo nepoukazuje na problém v útoku na 4 klientoch. V grafe s priebehom úlohy, ktorý je zobrazený na obrázku

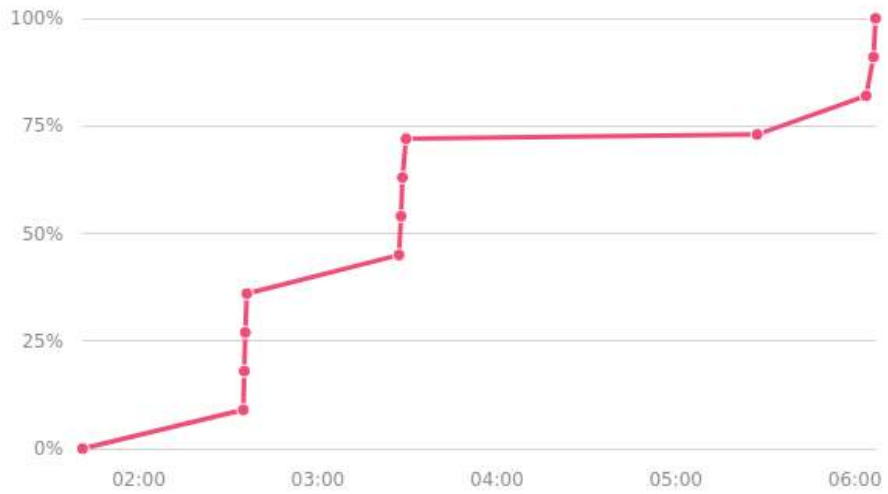
nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	24
minimálny počet prvkov v reťazci	2
maximálny počet prvkov v reťazci	2
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	adobe100.txt

Tabuľka 7.13: Hodnoty použitých nastavení v experimente.

metrika výpočtu	1 klient	2 klienti	4 klienti
efektivita	99 %	93 %	55 %
doba	9 hodín, 41 minút	5 hodín, 18 minút	4 hodiny, 25 minút

Tabuľka 7.14: Výsledky experimentov s pomalým hešovacím algoritmom s 1, 2 a 4 klientami

7.3, som si u experimentu s 4 klientami všimol časové obdobie o dĺžke približne 2 hodiny, kde nenastala žiadna zmena v stave úlohy. Na základe tohto zistenia usudzujem, že sa pravdepodobne vyskytli neznáme udalosti na klientoch (napr. antivírusová kontrola), ktoré tento experiment mohli ovplyvniť, a reálny prepad efektivity by bol nižší.



Obrázok 7.3: Priebeh úlohy v experimente so 4 klientami.

Experimenty s rýchlym hešovacím algoritmom

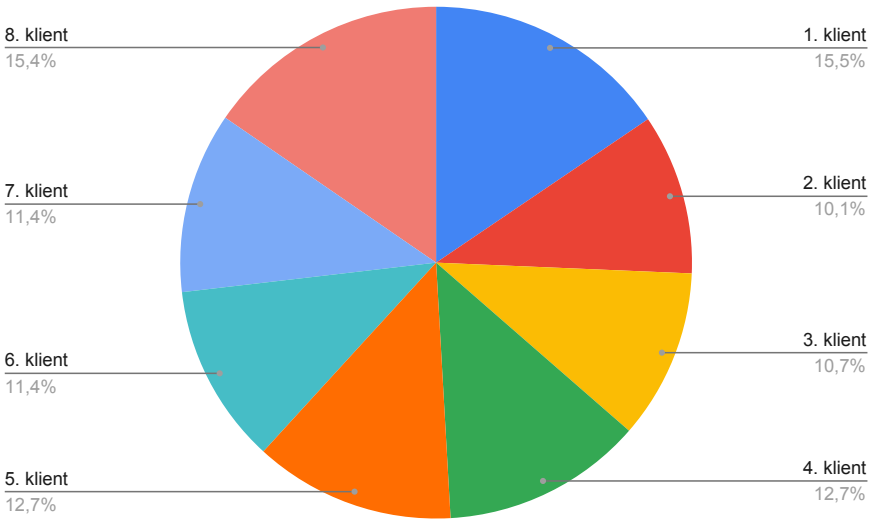
Experimenty s rýchlym hešovacím algoritmom boli vykonávané s 1, 2, 4 a 8 klientami. Ako zástupcu rýchlych heší som si vybral SHA-1. Hodnoty nastavení útoku PRINCE boli nastavené podľa tabuľky 7.15 a výsledný počet kandidátnych hesiel vygenerovaných algoritmom PRINCE bol 33 995 330 209.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	30
minimálny počet prvkov v reťazci	2
maximálny počet prvkov v reťazci	2
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	phpbb.txt

Tabuľka 7.15: Hodnoty použitých nastavení v experimente.

metrika výpočtu	1 klient	2 klienti	4 klienti	8 klientov
efektivita	99 %	94 %	90 %	74 %
doba	5 hod., 26 min.	2 hod., 44 min.	1 hod., 27 min.	54 min.

Tabuľka 7.16: Výsledky experimentov s rýchlym hešovacím algoritmom s 1, 2, 4 a 8 klientami



Obrázok 7.4: Distribúcia úlohy s útokom PRINCE medzi 8 klientov.

Namerané výsledky experimentov sú zobrazené v tabuľke 7.16. Efektivita distribuovaného útoku PRINCE je veľmi vysoká s minimom dodatočnej réžie a k výraznejšiemu poklesu dochádza až pri ôsmich klientoch. Na obrázku 7.4 je zobrazený graf, ktorý popisuje rozdelenie úlohy v experimente s ôsmimi klientami. Úloha bola distribuovaná rovnomerne a všetci klienti pracovali na približne rovnakej časti úlohy. Experimenty ukazujú, že implementované riešenie distribuovaného útoku PRINCE je výpočtovo veľmi efektívne.

7.3 Experimenty s rôznymi časmi pre pracovnú jednotku

U každého typu útoku v systéme Fitcrack je možné nastaviť čas vyhradený pre jednu pracovnú jednotku, ktorý ovplyvňuje počet hesiel v nej. V týchto experimentoch som sa zameral na analýzu vplyvu času pre pracovnú jednotku na celkovú efektivitu a dobu výpočtu lámacích úloh s útokom PRINCE. Experimenty prebiehali s desiatimi klientami.

Experimenty s pomalým hešovacím algoritmom

Cieľom útoku boli bcrypt heše získané pre 100 najpoužívanejších hesiel v službe Linkedin. Hodnoty použitých nastavení sú uvedené v tabuľke 7.17.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
pravidlá	nie
slovník	adobe100.txt

Tabuľka 7.17: Hodnoty použitých nastavení v experimente.

metrika výpočtu	600 sekúnd	1800 sekúnd	3600 sekúnd	7200 sekúnd
efektivita	99 %	98 %	97 %	98 %
doba	9 hod., 46 min.	9 hod., 44 min.	9 hod., 48 min.	9 hod., 49 min.

Tabuľka 7.18: Výsledky experimentov s rôznym časom pre jednu pracovnú jednotku.

Experimenty s rýchlym hešovacím algoritmom

Cieľom útoku boli SHA-1 heše získané pre 100 najpoužívanejších hesiel v službe Linkedin. Hodnoty použitých nastavení sú uvedené v tabuľke 7.19.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.19: Hodnoty použitých nastavení v experimente.

Namerané výsledky v tabuľkách 7.18 a 7.20 ukazujú, že vplyv veľkosti pracovnej jednotky na celkovú efektivitu a dobu výpočtu úlohy je minimálny. Pri príliš veľkom čase vyhradenom

metrika výpočtu	600 sekúnd	1800 sekúnd	3600 sekúnd	7200 sekúnd
efektivita	99 %	98 %	97 %	98 %
doba	8 hod., 45 min.	8 hod., 40 min.	8 hod., 47 min.	8 hod., 42 min.

Tabuľka 7.20: Výsledky experimentov s rôznym časom pre jednu pracovnú jednotku.

pre jednu pracovnú jednotku avšak hrozí, že vzniknú pracovné jednotky s veľkým počtom hesiel, ktoré sa nerozdelia medzi všetkých klientov a niektorí klienti ostanú nevyužití. Ak je vyhradený čas príliš malý, vytvára sa príliš veľa malých pracovných jednotiek, čo spôsobuje nárast réžie a pokles efektivity výpočtu, na čo upozorňuje aj systém Fitcrack pri vytváraní úlohy.

7.4 Porovnanie útoku PRINCE s inými typmi útokov

Ďalším cieľom navrhnutých experimentov bolo porovnanie distribuovaného útoku PRINCE s inými typmi útokov. Do experimentov som zaradil k útoku PRINCE aj slovníkový a kombinačný útok. V týchto experimentoch preto kladený dôraz na to, aby u každého experimentu bol počet kandidátnych hesiel medzi rôznymi útokmi identický alebo veľmi podobný. Aby sa táto podmienka dodržala, v prípade slovníkového útoku som si dopredu pripravil slovník, ktorý bol kombináciou vstupného slovníka pre útok PRINCE samého so sebou. V prípade kombinačného útoku bol ako ľavý a pravý slovník vybraný slovník, ktorý bol použitý ako vstupný slovník pre útok PRINCE. U útoku PRINCE bolo potrebné pre zaručenie tejto podmienky nastaviť minimálny a maximálny počet prvkov v reťazci na 2 a maximálnu dĺžku hesiel na prakticky najvyššiu možnú hodnotu, a to 30. Ostatné nastavenia boli ponechané na predvolených hodnotách. Podobne ako v predchádzajúcich experimentoch prezentovaných v podkapitole 7.2, aj u tohto porovnávania som bral do úvahy vplyv hešovacieho algoritmu na výpočtovú náročnosť úloh a všetky experimenty na porovnanie útokov medzi sebou som vykonal dvakrát - raz pre pomalý a raz pre rýchly hešovací algoritmus.

7.4.1 Experimenty s pomalým hešovacím algoritmom

Experimenty s pomalým hešovacím algoritmom boli vykonávané s 1, 2 a 4 klientami. Ako zástupcu pomalých heší som si vybral bcrypt a ako vstupný slovník bol použitý slovník *adobe100.txt*. Celkový počet kandidátnych hesiel u každého experimentu bol 10 000.

metrika výpočtu	PRINCE	kombinačný	slovníkový
efektivita	99 %	99 %	99 %
doba	9 hodín, 48 minút	20 hodín, 6 minút	13 hodín, 1 minúta

Tabuľka 7.21: Porovnanie PRINCE s kombinačným a slovníkovým útokom pri jednom klientovi

metrika výpočtu	PRINCE	kombinačný	slovníkový
efektivita	93 %	98 %	94 %
doba	5 hodín, 18 minút	10 hodín, 53 minút	5 hodín, 9 minút

Tabuľka 7.22: Porovnanie PRINCE s kombinačným a slovníkovým útokom pri dvoch klientoch

metrika výpočtu	PRINCE	kombinačný	slovníkový
efektivita	90 %	91 %	91 %
doba	2 hodiny, 43 minút	4 hodiny, 41 minút	2 hodiny, 41 minút

Tabuľka 7.23: Porovnanie PRINCE s kombinačným a slovníkovým útokom pri štyroch klientoch

V tabuľkách 7.25 až 7.28 sú uvedené výsledky z navrhnutých experimentov. Jedná sa o pomerne prekvapivé výsledky, a to najmä v prípade výsledkov u kombinačného útoku. Zaujímavým faktom je, že systém Fitcrack u každého z týchto experimentov na základe benchmarku vyhodnotil, že pre najefektívnejšie lámanie hesiel má pracovná jednotka obsahovať 900 hesiel. Toto rozhodnutie sa nejaví ako správne a optimálne, čo dokazujú aj namerané výsledky. Lepším rozhodnutím o počte hesiel v pracovnej jednotke by zrejme došlo k značnému zefektívneniu najmä kombinačného útoku v systéme Fitcrack. Zaujímavým výsledkom je aj to, že slovníkový útok bol o niekoľko hodín pomalší než útok PRINCE u úlohy s jedným klientom. Možným faktorom, ktorý mohol ovplyvniť dobu výpočtu, je usporiadanie hesiel v slovníku. Použitý slovník u slovníkového útoku nebol zoradený, a preto som experiment s jedným klientom zopakoval a tentoraz bol slovník zoradený podľa dĺžok hesiel od najkratších po najdlhšie. Výsledky tohto experimentu sú uvedené v tabuľke 7.24 a naznačujú, že usporiadanie slovníka je faktor, ktorý možne ovplyvniť efektivitu a dobu výpočtu úloh so slovníkovým útokom.

metrika výpočtu	PRINCE	slovníkový
efektivita	99 %	99 %
doba	9 hodín, 48 minút	9 hodín, 42 minút

Tabuľka 7.24: Porovnanie PRINCE so slovníkovým útokom pri jednom klientovi. Použitý slovník u slovníkového útoku bol zoradený podľa dĺžok hesiel od najkratších po najdlhšie.

7.4.2 Experimenty s rýchlym hešovacím algoritmom

Experimenty s rýchlym hešovacím algoritmom boli vykonávané s 1, 2, 4 a 8 klientami. Ako zástupcu rýchlych heší som si vybral SHA-1 a ako vstupný slovník bol použitý slovník, ktorý vznikol spojením slovníkov *phpbb.txt* a *honeynet.txt*. Celkový počet kandidátnych hesiel u každého experimentu bol skoro 168 miliárd. Keďže vytvorenie predpripraveného slovníka pre slovníkový útok nebolo z časových a hlavne kapacitných dôvodov reálne, porovnával som medzi sebou útok PRINCE a kombinačný útok. Alternatívou by bolo použiť menší slovník, no v tomto prípade sa systém Fitcrack u úlohy s ôsmimi pripojenými klientami rozhodol, že najlepšou voľbou je rozdeliť úlohu medzi 4 klientov, čo by bolo z pohľadu experimentov a analýzy výsledkov problémové.

metrika výpočtu	PRINCE	kombinačný
efektivita	99 %	53 %
doba	1 deň, 2 minúty	3 minúty, 46 sekúnd

Tabuľka 7.25: Porovnanie PRINCE s kombinačným útokom pri jednom klientovi

Získané výsledky z experimentov sú uvedené v tabuľkách 7.25 až 7.28. Ukazujú, že použitie útoku PRINCE s nastaveniami, ktorých výsledkom sú rovnaké kandidátne heslá ako

metrika výpočtu	PRINCE	kombinačný
efektivita	98 %	38 %
doba	11 hodín, 37 minút	2 minúty, 35 sekúnd

Tabuľka 7.26: Porovnanie PRINCE s kombinačným útokom pri dvoch klientoch

metrika výpočtu	PRINCE	kombinačný
efektivita	96 %	28 %
doba	6 hodín, 56 minút	2 minúty, 31 sekúnd

Tabuľka 7.27: Porovnanie PRINCE s kombinačným útokom pri štyroch klientoch

metrika výpočtu	PRINCE	kombinačný
efektivita	74 %	22 %
doba	3 hodiny, 38 minút	2 minúty, 28 sekúnd

Tabuľka 7.28: Porovnanie PRINCE s kombinačným útokom pri ôsmich klientoch

v prípade klasického kombinačného útoku, nie je vhodné a rozdiel v dobe výpočtu úlohy medzi týmito útokmi je veľmi veľký. Klasický kombinačný útok má preto stále svoje miesto medzi ostatnými typmi útokov a útok PRINCE ho úplne nenahradzuje. Experiment taktiež poukázal na praktické problémy u slovníkového útoku, kde slovník musí byť na začiatku dostupný na serveri a postupne sa jeho fragmenty rozdeľujú medzi klientov. V prípade generovaného slovníka, alebo slovníka, ktorý vznikol ako spojenie iných slovníkov, môže dôjsť ľahko k situácii, kedy výsledný slovník je príliš veľký a je problémové ho mať uložený na disku a následne s ním pracovať v rámci systému Fitcrack na serveri.

7.5 Porovnanie implementácií útoku PRINCE v systéme Fitcrack a Hashtopolis

V poslednej časti experimentov som porovnával implementácie útoku PRINCE medzi systémami Fitcrack a Hashtopolis. Verzia systému Hashtopolis použitá v experimentoch bola 0.12.0.

V prvom experimente som vytvoril malú úlohu, ktorá bola počítaná na 1 klientovi. V experimente som pomocou útoku PRINCE lámal SHA-1 heše získané pre 100 najpoužívanejších hesiel v službe Linkedin.

Pôvodným zámerom bolo u systému Hashtopolis vybrať benchmark typu *Runtime benchmark*, ktorý je založený na tom istom princípe ako implementovaný systém benchmarkov v systéme Fitcrack. Tu som však narazil na problém, keďže tento typ benchmarku nebol funkčný pre útok PRINCE - táto chyba bola vývojárom systému Hashtopolis nahlásená. Z tohto dôvodu som bol donútený pre experimenty použiť *Speed test* ako typ benchmarku spusteného pred samotným lámaním úlohy.

Konfigurácia prvej úlohy je uvedená v tabuľke 7.29. Výsledky prvého experimentu sú uvedené v tabuľke 7.30. Zatiaľ čo systém Fitcrack v informáciách o úlohe zobrazuje efektivitu výpočtu a aj predpokladaný čas trvania danej úlohy, systém Hashtopolis informáciu o efektivite neposkytuje a v odhade času trvania úlohy zobrazoval „—“. Výsledky ukazujú, že riešenia sú rovnako výkonné, čo sa očakávalo, keďže obe implementácie útoku PRINCE sú založené na rovnakej myšlienke distribúcie útoku. Vplyv algoritmu na adaptívne pláno-

nastavenie	hodnota
kontrola duplicitných hesiel	áno
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	6
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.29: Hodnoty použitých nastavení v prvom experimente.

metrika výpočtu	Fitcrack	Hashtopolis
efektivita	93 %	-
doba	29 minút, 22 sekúnd	29 minút, 25 sekúnd

Tabuľka 7.30: Porovnanie implementácii útoku PRINCE v systéme Fitcrack a Hashtopolis s jedným klientom na úlohe s jednou pracovnou jednotkou.

vanie pracovných jednotiek v systéme Fitcrack sa tu neprejavil, keďže celé lámanie úlohy spočívalo v spracovaní jednej pracovnej jednotky na jednom klientovi.

V druhom experimente som navrhol úlohu, kde je isté, že vznikne viac pracovných jednotiek. Tabuľka 7.31 obsahuje hodnoty použitých nastavení v tomto experimente, ktorý prebiehal v nedistribuovanom prostredí. V úlohe sa lámali tie isté SHA-1 heše ako v prvom experimente.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	9
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	2
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.31: Hodnoty použitých nastavení v druhom experimente.

metrika výpočtu	Fitcrack	Hashtopolis
efektivita	99 %	-
doba	4 hodiny, 9 minút	4 hodiny, 12 minút

Tabuľka 7.32: Porovnanie implementácií útoku PRINCE v systéme Fitcrack a Hashtopolis s jedným klientom na úlohe pozostávajúcej z viacerých pracovných jednotiek.

V tabuľke 7.32 sú uvedené výsledky druhého experimentu, ktoré ukazujú, že rozdiel medzi systémami je minimálny - útok PRINCE v systéme Fitcrack bol o pár minút rýchlejší. U tohto experimentu sa celá úloha rozdelila do viacerých pracovných jednotiek a za lepším výsledkom môže byť práve použitý algoritmu plánovania pracovných jednotiek, ktorý je viac prepracovanejší ako v systéme Hashtopolis a je aj predmetom dlhoročného výskumu.
V treťom experimente som použil bcrypt heše pre 100 najpoužívanejších hesiel v službe Linkedin, ktoré sa vo všeobecnosti lámu veľmi pomaly. Hodnoty nastavení úlohy sú uvedené v tabuľke 7.33.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	24
minimálny počet prvkov v reťazci	2
maximálny počet prvkov v reťazci	2
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	adobe100.txt

Tabuľka 7.33: Hodnoty použitých nastavení v treťom experimente.

metrika výpočtu	Fitcrack	Hashtopolis
efektivita	97 %	-
doba	9 hodín, 48 minút	6 hodín, 36 minút

Tabuľka 7.34: Porovnanie implementácií útoku PRINCE v systéme Fitcrack a Hashtopolis s jedným klientom na úlohe s pomalými bcrypt hešmi.

V tabuľke 7.34 sú uvedené výsledky tretieho experimentu. Systém Hashtopolis bol v tom experimente oveľa rýchlejší ako systém Fitcrack. Možným zdôvodnením môže byť fakt, že Fitcrack nastavil veľkosť každej pracovnej jednotky na 900, resp. 100 u poslednej jednotky.

Systém Fitcrack zistil, že u tejto úlohy sa láme jeden heš viac ako sekundu, čo zaokrúhlil na 1 z dôvodu reprezentácie výkonu klienta v celých číslach. Následne sa naplánovala minimálna možná pracovná jednotka, kde čas na ňu vyhradený bol vypočítaný ako 0,25 (predvolená hodnota koeficientu *ramp down*) * 3600 (čas pre jednu pracovnú jednotku), čo je 900 sekúnd. Na základe zisteného výkonu klienta, ktorý je jeden heš za sekundu, veľkosť pracovnej jednotky sa stanovila na 900 hesiel. Y informáciách o úlohe je možné zistiť, že sa vytvorilo až 12 pracovných jednotiek. Hashtopolis vytvoril len dve pracovné jednotky s 6832 a 3168 heslami.

Rozhodol som sa tejto jav bližšie preskúmať a zmenil som globálne nastavenie koeficientu *ramp down* na hodnotu 1. Experiment som následne zopakoval. Vytvorila sa iba jedna pracovná jednotka so všetkými 10 000 heslami a celá úloha trvala 5 hodín a 56 minút. Znížiť množstvo vygenerovaných pracovných jednotiek by bolo možné aj zvýšením minimálneho času vyhradeného na jednu pracovnú jednotku. Experiment priniesol zaujímave výsledky, ktoré sa síce neviažu na útok PRINCE ako taký, ale ukázal, že existuje priestor na dodatočné zlepšenia adaptívneho plánovacieho algoritmu používaného v systéme Fitcrack.

Štvrtý a zároveň posledný experiment bol zameraný na použitie útoku PRINCE v distribuovanom prostredí, ktoré obsahovalo desať klientov. Tabuľka 7.35 obsahuje hodnoty nastavení úlohy. Cieľom útoku boli SHA-1 heše získané pre 100 najpoužívanejších hesiel v službe Linkedin.

nastavenie	hodnota
kontrola duplicitných hesiel	nie
permutácia prvých písmen hesiel	nie
minimálna dĺžka hesiel	1
maximálna dĺžka hesiel	8
minimálny počet prvkov v reťazci	1
maximálny počet prvkov v reťazci	4
čas pre pracovnú jednotku	3600 sekúnd
pravidlá	nie
slovník	rockyou.txt

Tabuľka 7.35: Hodnoty použitých nastavení v štvrtom experimente.

metrika výpočtu	Fitcrack	Hashtopolis
efektivita	97 %	-
doba	8 hodín, 47 minút	8 hodín, 53 minút

Tabuľka 7.36: Porovnanie implementácií útoku PRINCE v systéme Fitcrack a Hashtopolis s desiatimi klientami.

Výsledky štvrtého experimentu sú uvedené tabuľke 7.36. Výsledky sú dosť podobné, s mierne lepším časom pre systém Fitcrack. Tento experiment poukazuje na fakt, že aj keď je distribúcia a realizácia útoku PRINCE medzi systémami veľmi podobná, riešenie v systéme Fitcrack sa javí byť ako mierne efektívnejšie a to najmä vďaka prepracovanejším algoritmom plánovania a generovania pracovných jednotiek. Ak vývojári systému Hashtopolis v budúcnosti opravia problém s benchmarkom typu *Runtime benchmark* u útoku PRINCE, mohlo by byť zaujímavé porovnať implementácie útoku medzi týmito dvoma systémami s použitím tohto typu benchmarku.

7.6 Zhodnotenie experimentov

Vykonané experimenty priniesli viacero zaujímavých zistení, ktoré budú zhrnuté v tejto podkapitole. Celkovo výsledky poukazujú na to, že navrhnuté a implementované riešenie realizácie distribuovaného útoku PRINCE je funkčné a efektívne.

V podkapitole 7.1 som skúmal vplyv rôznych konfiguračných možností útoku PRINCE na celkový počet kandidátnych hesiel a na dobu potrebnú na výpočet úlohy. Výsledky ukázali, že zvyšovaním počtu prvkov v reťazci výrazne narastá počet kandidátnych hesiel. Vyšší počet prvkov v reťazci nemusí byť veľmi užitočný, keďže distribúcia dĺžok hesiel je vo všeobecnosti známa a väčšina používateľských hesiel má dĺžku 6 až 8 znakov. Ďalej sa ukázalo, že kontrola duplicitných hesiel neprináša vysokú réžiu. V prípade, že používateľ vyberie viac vstupných slovníkov, je táto voľba veľmi efektívna a zamedzí generovaniu duplicitných hesiel. Povolenie permutácie prvých písmen slov výrazne navýšilo počet kandidátnych hesiel, čím sa zvýšila sila útoku.

Škálovateľnosť útoku som skúmal na sade experimentov v podkapitole 7.1.4. Experimenty som vykonal s rýchlym aj pomalým hešovacím algoritmom. Pridelením ďalších klientov na výpočet úlohy sa zvyšovala rýchlosť lámania hesiel. Navrhnutý spôsob distribúcie útoku vykazuje lineárnu škálovatelnosť.

Podkapitola 7.3 sa venuje vplyvu veľkosti pracovnej jednotky na efektivitu útoku a na dobu potrebnú na výpočet úlohy. Výsledky boli pre rôzne veľkosti pracovnej jednotky veľmi podobné. Extrémne veľkosti by avšak zvýšili množstvo réžie a znížili efektivitu distribúcie útoku medzi klientov.

V podkapitole 7.4 som porovnával implementovaný útok so slovníkovým a kombinačným útokom. Útok PRINCE mal najlepšie výsledky pri lámaní pomalých heší. Pri lámaní rýchlych heší bol kombinačný útok v skúmanom prípade s dvoma prvkami v reťazci najrýchlejší. V experimente sa taktiež ukázalo, že zatiaľ čo u kombinačného útoku a útoku PRINCE sa generujú a lámu heslá na klientoch, v prípade slovníkového útoku je nutné slovník dopredu nahrať na server. Veľkosť slovníka môže byť často veľmi veľká, resp. cieľový slovník nemusí byť možné ani vytvoriť či uložiť na server. Toto zistenie ukazuje, že kombinačný útok alebo útok PRINCE nie je možné úplne nahradiť dopredu vygenerovaným slovníkom a následným slovníkovým útokom s týmto slovníkom.

Implementácie útoku PRINCE v systéme Fitcrack a Hashtopolis som porovnával v podkapitole 7.5. Efektivita útoku v oboch implementáciách je veľmi vysoká. Útok v systéme Fitcrack bol pri lámaní rýchlych heší mierne rýchlejší v porovnaní so systémom Hashtopolis, čo môže byť spôsobené kvalitnejším plánovacím algoritmom, ktorý je v systéme Fitcrack adaptívny a priebežne počas výpočtu úlohy prispôsobuje veľkosti pracovných jednotiek podľa výpočtovej sily klientov. Pri lámaní pomalých heší bol systém Hashtopolis o dosť lepší, kde problém na strane systému Fitcrack nebol v samotnom útoku, ale v plánovacom algoritme, ktorý v danom experimente generoval príliš veľa pracovných jednotiek.

Kapitola 8

Záver

V tejto práci som sa venoval distribuovanému lámaniu hesiel, systému Fitcrack a algoritmu PRINCE. Na začiatku som predstavil platformu BOINC, ktorá umožňuje distribuovať náročné výpočtové úlohy medzi viacerých klientov a ich zariadenia. Ďalej som sa venoval systému Fitcrack, ktorý využíva technológiu BOINC na distribuované lámanie hesiel. Následne som sa zaoberal algoritmom PRINCE, ktorý je možné používať na pokročilé kombinačné útoky.

Cieľom práce bolo navrhnúť a implementovať distribuovaný útok PRINCE, ktorý bol následne integrovaný do systému Fitcrack. Na generovanie hesiel bol použitý nástroj princeprocessor, ktorý je referenčnou implementáciou algoritmu PRINCE. Vygenerované heslá sú následne presmerované do nástroja hashcat, ktorý slúži na lámanie hesiel. Distribúcia útoku je založená na možnostiach nástroja princeprocessor, ktorý umožňuje generovať podmnožinu množiny možných hesiel. Každému klientovi je pridelený určitý rozsah hesiel, ktoré si pomocou nástroja vygeneruje, a následne prebieha proces lámania hesiel. Prelomené heslá posiela klient na server a následne sa zobrazujú používateľovi vo webovom klientovi.

Nový distribuovaný útok bol plne integrovaný do systému Fitcrack. Pri tvorbe novej úlohy vo webovom klientovi si používateľ môže vybrať útok PRINCE a následne sa mu otvorí webové grafické rozhranie, ktoré umožňuje nastaviť konfiguráciu útoku presne podľa potrieb daného používateľa. Systém po zmene konfigurácie útoku automaticky prepočíta odhadový čas potrebný na výpočet úlohy. Na základe tohto odhadu môže používateľ upraviť konfiguráciu útoku. Po vytvorení novej úlohy s útokom PRINCE sa zobrazia všeobecné informácie o úlohe a zároveň aj hodnoty nastavení, ktoré sú špecifické pre tento typ útoku.

Na sade experimentov bol skúmaný vplyv rôznych konfiguračných možností na celkové množstvo kandidátnych hesiel. Permutácia prvých písmen slov výrazne zvyšuje množstvo generovaných hesiel a môže značne zvýšiť silu útoku. Kontrola duplicitných hesiel sa ukázala ako užitočná a v prípade použitia viacerých slovníkov naraz, kde je vysoká pravdepodobnosť existencie duplicitných hesiel v spojenom slovníku, je odporúčané ju používať. V ďalších experimentoch bola skúmaná škálovateľnosť implementovaného útoku. Výsledky experimentov ukazujú, že implementované riešenie je efektívne a vykazuje lineárnu škálovatelnosť.

Predmetom skúmania bol aj vplyv veľkosti pracovnej jednotky na efektivitu útoku. Tento vplyv bol zanedbateľný, keďže doba výpočtu úlohy pre rôzne veľkosti pracovnej jednotky bola veľmi podobná. Následne bol útok porovnaný so slovníkovým a kombinačným útokom. Výsledky lámania pomalých heší ukázali, že útok PRINCE bol najrýchlejší z tejto trojice útokov. Pri rýchlych hešiach bol kombinačný útok rýchlejší. Na záver boli porov-

nané implementácie tohto útoku medzi systémami Fitcrack a Hashtopolis. Efektivita oboch implementácií je vysoká a výsledné časy trvania úloh v experimentoch boli veľmi podobné.

Na túto prácu je možné nadviazať rôznymi vylepšeniami, ktoré by vylepšili silu a použitie útoku PRINCE v systéme Fitcrack. Podpora oddeľovacích znakov medzi slovami pri generovaní hesiel podľa algoritmu PRINCE by bola zaujímavým rozšírením nástroja princeprocessor. Pre skúsených používateľov systému Fitcrack by mohla byť užitočná nová možnosť na ručné zadanie argumentov pre nástroj princeprocessor pre získanie plnej kontroly nad generátorom kandidátnych hesiel. Ďalší výskum by sa mohol týkať aj nástroja PRINCE-LING, ktorý vytvára špeciálne slovníky na základe pravdepodobnostných bezkontextových gramatík. Vygenerované slovníky sú optimalizované na použitie v útoku PRINCE a mohli by zvýšiť jeho silu.

V experimentálnej časti bol odhalený problém s vykresľovaním grafov v systéme Fitcrack. Pri väčšom množstve dát je načítanie informácií o úlohe veľmi pomalé a pohyb na stránke nie je plynulý. Riešením by mohlo použitie inej knižnice na vykresľovanie grafov, ktorá nemá problém so spracovaním veľkého množstva dát.

Výsledky experimentov poukázali aj nízku efektivitu útoku pri pomalých hešiach, ktorá nie je spôsobená samotným útokom, ale adaptívnym plánovacím algoritmom. Plánovací algoritmus generoval príliš veľa pracovných jednotiek, ktoré spomaľovali výpočet úlohy nadbytočnou réžiou. Riešením problému by mohlo byť zvýšenie minimálneho času vyhradeného na spracovanie jednej pracovnej jednotky u pomalých heší.

Literatúra

[1] Alvarez, N. *The BOINC scheduling server protocol* [online]. [cit. 2019-11-23]. Dostupné z: https://boinc.berkeley.edu/trac/wiki/RpcProtocol.

[2] Anderson, D., Korpela, E. a Walton, R. High-performance task distribution for volunteer computing. In:. Január 2006. ISBN 0-7695-2448-6.

[3] Anderson, D. P. BOINC: A Platform for Volunteer Computing. *Journal of Grid Computing*. Nov 2019. Dostupné z: https://doi.org/10.1007/s10723-019-09497-9. ISSN 1572-9184.

[4] Chanda, K. Password Security: An Analysis of Password Strengths and Vulnerabilities. *International Journal of Computer Network and Information Security*. Júl 2016, roč. 8, s. 23–30.

[5] Chripko, J. *Automatizované testování systému Fitcrack*. Brno, CZ, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/20498/.

[6] Coisel, I., Sanchez, I. a Galbally, J. Divide, recombine and conquer: Syntactic patterns-reassembly algorithm applied to password guessing process. In:. Október 2017, s. 1–6.

[7] Coray, S. *Hashtopolis: A Hashcat wrapper for distributed hashcracking* [online]. [cit. 2020-02-17]. Dostupné z: https://github.com/s3inlc/hashtopolis.

[8] Coray, S. *Óðinn: Framework for Large-Scale Wordlist Analysis and Structure-Based Password Guessing*. Basel, Switzerland, 2019. Diplomá práce. Natural Science Faculty of the University of Basel. Dostupné z: https://dbis.dmi.unibas.ch/teaching/studentprojects/analysis-and-distribution-of-large-wordlists-for-password-recovery/.

[9] David P. Anderson, B. A. *A Runtime System for Volunteer Computing* [online]. [cit. 2019-11-29]. Dostupné z: https://boinc.berkeley.edu/boinc_papers/api/text.html.

[10] Fielding, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. 2000. Doctoral dissertation. University of California, Irvine. Dostupné z: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[11] Grinberg, M. *Flask Web Development: Developing Web Applications with Python*. 1st. O'Reilly Media, Inc., 2014. ISBN 1449372627.

[12] Hope, C. *BOINC* [online]. [cit. 2019-12-02]. Dostupné z: https://www.computerhope.com/jargon/b/boinc.htm.

[13] Horák, A. *Správa výpočetních úloh v systému Fitcrack*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22850/.

[14] Hranický, R., Matoušek, P., Ryšavý, O. a Veselý, V. Experimental Evaluation of Password Recovery in Encrypted Documents. In: *Proceedings of ICISSP 2016*. SciTePress - Science and Technology Publications, 2016, s. 299–306. Dostupné z: https://www.fit.vut.cz/research/publication/11052. ISBN 978-989-758-167-0.

[15] Hranický, R., Zobal, L., Ryšavý, O. a Kolář, D. Distributed password cracking with BOINC and hashcat. *Digital Investigation*. 2019, roč. 2019, č. 30, s. 161–172. Dostupné z: https://www.fit.vut.cz/research/publication/11961. ISSN 1742-2876.

[16] Hranický, R., Zobal, L., Večeřa, V. a Matoušek, P. Distributed Password Cracking in a Hybrid Environment. In: *Proceedings of SPI 2017*. University of Defence in Brno, 2017, s. 75–90. Dostupné z: https://www.fit.vut.cz/research/publication/11358. ISBN 978-80-7231-414-0.

[17] Hranický, R., Zobal, L., Večeřa, V. a Múčka, M. *Distribuce výpočtů pro nástroj hashcat*. 2018. 29 s. Dostupné z: https://www.fit.vut.cz/research/publication/11884.

[18] Hranický, R., Zobal, L., Večeřa, V., Múčka, M., Horák, A. et al. *The architecture of Fitcrack distributed password cracking system, version 2*. 2020. 85 s. Dostupné z: https://www.fit.vut.cz/research/publication/12300.

[19] Lundberg, T. *Comparison of Automated Password Guessing Strategies*. 2019. 70 s. Diplomová práca. Linköping University, Information Coding.

[20] McAtee, M. a Morris, L. CrackLord: Maximizing Computing Resources. In: *Black Hat USA*. August 2015, s. 7.

[21] Page, A. a Naughton, T. Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing. In:. Január 2005.

[22] Pelchen, C., Jaeger, D., Cheng, F. a Meinel, C. The (Persistent) Threat of Weak Passwords: Implementation of a Semi-automatic Password-Cracking Algorithm. In: Heng, S.-H. a Lopez, J., ed. *Information Security Practice and Experience*. Cham: Springer International Publishing, 2019, s. 464–475. ISBN 978-3-030-34339-2.

[23] Picolet, J. *Purple Rain Attack: Password Cracking With Random Generation* [online]. [cit. 2020-02-11]. Dostupné z: https://www.netmux.com/blog/purple-rain-attack.

[24] Picolet, J. *Hash Crack: Password Cracking Manual, version 3*. Netmux, 2019. ISBN 9781793458612.

[25] Pokorný Šimon. *Analytické zpracování metadat k lámání hesel*. Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21745/.

[26] Steube, J. *Hashcat - advanced password recovery* [online]. [cit. 2019-11-27]. Dostupné z: https://hashcat.net/.

[27] Steube, J. *Hashcat forums - Practical PRINCE* [online]. [cit. 2019-12-05]. Dostupné z: https://hashcat.net/forum/thread-3914.html.

[28] Steube, J. *Introducing the PRINCE Attack-Mode* [online]. [cit. 2019-12-05]. Dostupné z: https://video.adm.ntnu.no/pres/5494033a6f015.

[29] Steube, J. PRINCE: Modern Password Guessing Algorithms. In: *International Conference on Passwords (PASSWORDS'14).* December 2014.

[30] Večeřa, V. *Strategie distribuovaného lámání hesel.* Brno, CZ, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21556/.

[31] Weir, M. *Tool Deep Dive: PRINCE* [online]. [cit. 2019-12-05]. Dostupné z: https://reusablesec.blogspot.com/2014/12/tool-deep-dive-prince.html.

[32] You, E. *Introduction — Vue.js* [online]. [cit. 2020-01-20]. Dostupné z: https://vuejs.org/v2/guide/.

Príloha A

Obsah DVD

Na priloženom médiu v adresári *fitcrack* sú dostupné kompletné zdrojové kódy systému Fitcrack. Zdrojové súbory nástroja princeprocessor sa nachádzajú v adresári *princeprocessor*. Automatické testy systému Fitcrack rozšírené o testy pre útok PRINCE sú v adresári *automaticke-testy*. Slovníky a súbory s hešmi, ktoré boli použité v experimentoch, sa nachádzajú v adresári *experimenty-subory*. V adresári *doc* sa nachádza PDF verzia technickej správy spolu s jej zdrojovými súbormi v jazyku LaTeX:

```
DVD
├── fitcrack
├── princeprocessor
├── automaticke-testy
│   ├── test_api.py
│   └── test_runner.py
├── experimenty-subory
│   ├── slovniky
│   │   ├── rockyou.txt
│   │   ├── rockyouwithdups.txt
│   │   ├── adobe100.txt
│   │   ├── phpbb.txt
│   │   └── honeynet.txt
│   └── hash-subory
│       ├── linkedin-top100-sha1.hash
│       ├── linkedin-top100-bcrypt.hash
│       └── linkedin-top100-plain-passwords.txt
└── doc
    ├── DP.pdf
    └── zdrojové súbory technickej správy
```

Príloha B

Úpravy v systéme Fitcrack

Počas tvorby tejto práce som pridal alebo upravil rôzne súbory v systéme Fitcrack. Všetky moje úpravy v Github repozitári so systémom Fitcrack je možné nájsť na stránke https://github.com/nesfit/fitcrack/commits?author=davidbolvansky. Úpravy sa týkali súborov v nasledovných adresároch:

- `/webadmin/fitcrackFE` - Webový klient
- `/webadmin/fitcrackAPI` - Serverová časť
- `/server/sql` - Skripty SQL pre vytvorenie a nastavenie databáze
- `/server/src` - Zdrojové kódy generátora
- `/server/templates` - Šablóny popisujúce súbory súvisiace s pracovnými jednotkami
- `/runner` - Zdrojové kódy programu Runner

Úpravy vo webovom klientovi

Úpravy vo webovom klientovi sa týkali nasledovných súborov:

- `/store/job-form.js`

  Do tohto súboru boli pridané nové konfiguračné možnosti špecifické pre útok PRINCE do formulára na vytvorenie úlohy. Taktiež tu bola pridaná kontrola správnosti vstupných údajov pri vytváraní úlohy s útokom PRINCE. Do zoznamu typov útokov bol pridaný nový útok PRINCE.

- `/components/jobDetail/jobDetailView.vue`

  Úpravy v tomto súbore sa týkali pridania podpory pre zobrazenie špecifických konfiguračných možností u úloh s útokom PRINCE.

- `/components/jobDetail/attacks/prince.vue`

  Nový súbor, v ktorom je implementované webové grafické rozhranie, ktoré zobrazuje podrobnosti o úlohe s útokom PRINCE.

- `/components/jobDetail/jobEditor.vue`

  V tomto súbore bola implementovaná podpora pre editáciu konfiguračných možnosti u už vytvorenej úlohy s útokom PRINCE.

- `/components/job/jobsView.vue`

  Jedinou úpravou v tomto súbore bolo pridanie útoku PRINCE do zoznamu všetkých typov útokov, podľa ktorých je možné filtrovať úlohy vo webovom klientovi.

- `/components/job/addJobView.vue`

  V tomto súbore bola pridaná komponenta pre konfiguráciu útoku PRINCE pri vytváraní úlohy.

- `/components/job/attacks/prince.vue`

  Nový súbor, v ktorom je implementované webové grafické rozhranie na vytvorenie úlohy s útokom PRINCE. V súbore je taktiež implementovaná funkcionalita na kontrolu správnosti vstupných údajov pri vytváraní úlohy s útokom PRINCE. V prípade, že sa odhalí chyby vo vstupných údajov, zobrazí sa používatelovi správa o zistenej chybe.

- `/assets/scripts/iconMaps.js`

  Do súboru bolo pridanie mapovanie útoku PRINCE a odpovedajúcu ikonku pre tento útok, ktorá sa zobrazuje vo webovom klientovi.

Úpravy v serverovej časti

Úpravy v v serverovej časti sa týkali nasledovných súborov:

- `/princeprocessor/pp64.bin`

  Pridaný binárny súbor pre staticky skompilovaný nástroj princeprocessor.

- `/src/settings.py`

  Do súboru bola pridaná nová konštanta, ktorá obsahuje cestu k binárnemu súboru s nástrojom princeprocessor.

- `/src/src/api/fitcrack/lang.py`

  Pridanie mapovanie útoku PRINCE na kód útoku 8.

- `/src/src/api/fitcrack/endpoints/job/job.py`

  Úpravy v tomto súbore sa týkali pridania podpory pre editáciu úlohy s útokom PRINCE - uloženie nových položiek do databázy a výpočet novej veľkosti množiny možných hesiel pomocou nástroja princeprocessor.

- `/src/src/api/fitcrack/endpoints/job/functions.py`

  V súbore bola pridaná podpora pre odhad času trvania úlohy s útokom PRINCE na základe veľkosti množiny možných hesiel, ktorej veľkosť bola vypočítaná pomocou nástroja princeprocessor.

- `/src/src/api/fitcrack/attacks/processJob.py`

  V tomto súbore bolo implementované uloženie údajov úlohy s útokom PRINCE do databázy.

- `/src/src/api/fitcrack/attacks/functions.py`

  Do súboru bola pridaná nová funkcia *compute_prince_keyspace*, ktorá prevádza konfiguračné možnosti a ich hodnoty nastavené u úlohy na argumenty pre nástroj princeprocessor. Následne spúšťa tento nástroj s prevedenými argumentami za účelom získania veľkosti množiny možných hesiel pre konfiguráciu útoku PRINCE v danej úlohe.

- `/src/src/api/fitcrack/endpoints/serverInfo/transfer.py`

  Do zoznamu položiek, ktoré sa majú exportovať pri exporte úlohy, boli pridané položky špecifické pre útok PRINCE.

- `/src/src/database/models.py`

  V tomto súbore bola upravená trieda *FcJob*, do ktorej boli pridané konfiguračné možnosti pre útok PRINCE.

Úpravy v skriptoch SQL

Úpravy v skriptoch SQL sa týkali nasledovných súborov:

- `10_create_tables.sql`

  V skripte bola rozšírenia definícia tabuľky *fc_job* o nové stĺpce, ktoré odpovedajú novým konfiguračným možnostiam útoku PRINCE:

- `30_insert_data.sql`

  Pridaný príkaz na vytvorenie nového záznamu do tabuľky *fc_job* s ukážkovou úlohou *sample-prince-md5* s útokom PRINCE, ktorú si používateľ môže spustiť hneď po nainštalovaní systému Fitcrack.

Úpravy v generátore

Úpravy v generátore sa týkali nasledovných súborov:

- `/source/Config.cpp`

  Bola pridané nové konštanty s názvom šablón definujúcich odosielané súbory klientom pri úlohe s útokom PRINCE.

- `/headers/Config.h`

  Do súboru bola pridaná nová konštanta, ktorá definuje kód útoku pre útok PRINCE.

- `/source/AttackModes/AttackPrince.cpp`

  Nový súbor s implementáciou triedy *CAttackPrince*. Trieda implementuje vytváranie pracovných jednotiek u úloh s útokom PRINCE.

- `/headers/AttackModes/AttackPrince.h`

  Nový hlavičkový súbor s deklaráciou triedy *CAttackPrince*.

- `source/Generators/SimpleGenerator.cpp`

  V súbore bola implementované spracovanie útoku s kódom 8 a následné vytváranie pracovných jednotiek s útokom PRINCE.

- `source/Generators/AbstractGenerator.cpp`

  V súbore bola upravená metóda *getStickyFiles*, kde u úlohy s útokom PRINCE medzi *sticky* súbory patrí slovník a súbor s pravidlami.

- `source/work_generator.cpp`

  Úpravy v tomto súbore sa týkali registrácii nových šablón pre útok PRINCE.

Úpravy v šablónach

Úpravy v šablónach sa týkali nasledovných súborov:

- `prince_in`

  Nová šablóna definujúca odosielané súbory klientom pri úlohe s útokom PRINCE.

- `prince_rules_in`

  Nová šablóna definujúca odosielané súbory klientom pri úlohe s útokom PRINCE a použití pravidiel.

Úpravy v programe Runner

Úpravy v programe Runner sa týkali nasledovných súborov:

- `/src/ConfigTask.cpp`

  Do tohto súboru boli pridané konštanty, ktorých názov reprezentuje konfiguračné možnosti útoku PRINCE. V metóde *initSupported* bola pridaná podpora pre nové konfiguračné možnosti pre útok PRINCE.

- `/include/Attack.hpp`

  Útok PRINCE bol v tomto súbore pridaný do zoznamu podporovaných útokov.

- `/src/Attack.cpp`

  Do súboru bolo pridané spustenie útoku PRINCE pri odpovedajúcom kóde útoku v úlohe.

- `/src/AttackPrince.hpp`

  Nový s implementáciou triedy *AttackPrince*. Trieda implementuje mapovanie konfiguračných možnosti útoku PRINCE na argumenty pre nástroj hashcat a princeprocessor.

- `/include/AttackPrince.hpp`

  Nový hlavičkový súbor s deklaráciou triedy *AttackPrince*.

- `/src/TaskComputeBase.cpp`

  Pridané prepojenie výstupu nástroja princeprocessor na vstup nástroja hashcat.