VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA TECHNOLOGIÍ PRO DISTRIBUCI VÝPOČTU PŘI LÁMÁNÍ HESEL

ANALYSIS OF DISTRIBUTED COMPUTING TECHNOLOGIES FOR PASSWORD CRACKING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PATRIK MRÁZ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK HRANICKÝ**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Mráz Patrik, Bc.**
Program: Informační technologie Obor: Počítačové sítě a komunikace
Název: **Analýza technologií pro distribuci výpočtu při lámání hesel**
Analysis of Distributed Computing Technologies for Password Cracking
Kategorie: Paralelní a distribuované výpočty
Zadání:

1. Seznamte se s technologiemi pro distribuci výpočtů na více uzlů a odhadněte, které z nich mohou být použitelné pro výpočty typu GPGPU, konkrétně lámání hesel.
2. Navrhněte vhodné testovací prostředí a způsoby experimentálního ověření vaší hypotézy. Návrh může zahrnovat jak vlastní software, tak existující nástroje pro lámání hesel.
3. Implementujte navržený software a připravte testovací prostředí pro realizaci experimentů.
4. Realizujte navržené experimenty a zhodnoťte dosažené výsledky.

Literatura:

- Keonwoo Kim. "Distributed password cracking on GPU nodes". In: 2012 7th International Conference on Computing and Convergence Technology (ICCCT). Dec. 2012, s. 647-650.
- D. Apostal, K. Foerster, A. Chatterjee, and T. Desell. "Password recovery using MPI and CUDA". In: 2012 19th International Conference on High Performance Computing. Dec. 2012, s. 1-9.
- D. P. Anderson. "BOINC: a system for public-resource computing and storage". In: Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on. Nov. 2004, s. 4-10.
- Radek Hranický, Martin Holkovič, Petr Matoušek, and Ondřej Ryšavý. "On Efficiency of Distributed Password Recovery". In: The Journal of Digital Forensics, Security and Law 11.2 (2016), s. 79-96. issn: 1558-7215.
- Dále dle domluvy s vedoucím.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

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

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

Abstrakt

Cieľom tejto práce je analýza technológií pre distribúciu výpočtu pri lámaní hesiel. Distribúcia je nevyhnutná s ohľadom na celkovú dobu lámania, ktorá môže byť v niektorých prípadoch aj desiatky rokov. V úvodnej časti je priblížená problematika obecného lámania hesiel vrátane typov útokov a najpopulárnejších nástrojov. Následne sa venujeme paralelizácií na GPU a tiež nutnosti distribúcie výpočtu na viacero počítačov. Venujeme sa rôznym technológiam ako VirtualCL, BOINC, MPI a analyzujeme ich využiteľnosť v prípade lámania hesiel. Skúmame ich z hľadiska výkonnosti, efektivity, škálovateľnosti a prispôsobivosti pri vopred definovaných podmienkach. Súčasťou práce je tiež návrh a implementácia distribúcie lámania pomocou technológie MPI s nástrojom hashcat, ktorý sa prehlasuje za najrýchlejší lámač hesiel.

Abstract

The goal of this thesis is to analyze the technologies for distributed computing in password cracking. Distribution is a key factor regarding the total time of cracking the password which can sometimes take up to tens of years. In the introductory section we take a look at the general password cracking, types of attacks and the most popular tools. Next we address the GPU parallelization as well as the need of distributed computing on multiple computers. We look at all kinds of technologies, such as VirtualCL, BOINC, MPI and analyze their usability in password cracking. We examine each technology's performance, efficiency, scalability and adaptability when given pre-defined conditions. Part of this thesis is a design and implementation of distributed password cracking using MPI technology along with Hashcat, a self-proclaimed "World's fastest password cracker".

Kľúčové slová

MPI, BOINC, hashcat, John The Ripper, distribúcia, lámanie hesiel, obnova hesiel, VirtualCL

Keywords

MPI, BOINC, hashcat, John The Ripper, distribution, password cracking, password recovery, VirtualCL

Citácia

MRÁZ, Patrik. *Analýza technologií pro distribuci výpočtu při lámání hesel.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický

Analýza technologií pro distribuci výpočtu při lámání hesel

Prehlásenie

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

.....
Patrik Mráz
21. mája 2019

Poďakovanie

Touto cestou by som chcel poďakovať vedúcemu tejto diplomovej práce Ing. Radkovi Hranickému za jeho profesionálny prístup a odborné rady. V neposlednom rade ďakujem celej svojej rodine, ktorá ma od útleho detstva podporuje a stojí pri mne. Špeciálne ďakujem patrí mojej priateľke Deniske a našej dcérke Viktórii.

Obsah

Kapitola 1

Úvod

Dnešná doba sa nesie v znamení informačných technológií, ktoré nám uľahčujú fungovanie v mnohým ohľadoch. Niektoré úlohy vyžadujú vysokú výpočtovú silu a je takmer nemožné ich počítať na jednom zariadení. Ide napríklad o rozsiahle vedecké výpočty, predpovede počasia alebo forenzné účely k odhaľovaniu dôkazov.

Lámanie alebo obnova hesiel je užitočná nielen ako súčasť forenznej analýzy, kedy sa snažíme dešifrovať zaistené digitálne materiály, ktoré môžu viesť k dolapeniu páchateľov, ale aj ako súčasť bežného života. Pri strate hesla z webového informačného systému môžeme požiadať správcu systému o obnovenie starého (ak nie je šifrované), alebo nastavenie nového hesla. Túto možnosť nemáme pri heslách k šifrovaným súborom, pevným diskom, archívom a iným fyzicky uloženým dátam. Jediná možnosť je obnova hesla, ktorá je možná vďaka tomu, že poznáme algoritmus ktorým boli dáta zašifrované a tiež mechanizmus, akým si môžeme overiť správnosť zadaného hesla. Problém je, že pri aktuálnych algoritmoch je možností priveľa nato, aby bolo heslo v rozumnom čase obnovené na jednom počítači.

Pri lámaní hesiel sú všetky pokusy o dešifrovanie na sebe nezávislé, čo nám dovoľuje využiť masívne paralelizovanie medzi jadrami systému (CPU/GPU), ale aj medzi viacerými systémami v distribuovanom prostredí. Existuje mnoho komerčných aj open-source implementácií distribúcie výpočtov, ktoré sa líšia použitými technológiami, flexibilitou, spôsobom používania a mnohým ďalším.

Cieľom tejto práce je analýza a porovnanie technológií pre distribúciu výpočtu, ich kľúčových vlastností, výhod a nevýhod pri lámaní hesiel. Budeme skúmať ich efektivitu, spôsob komunikácie, nároky na réžiu, ale aj užívateľskú prívetivosť, možnosti dynamického pridávania a odoberania výpočtových uzlov atď.

V kapitole 2 tejto práce si priblížime princíp lámania hesiel a povieme si tiež v čom spočívajú jednotlivé typy útokov. Následne sa pozrieme na existujúce nástroje na lámanie hesiel a kapitolu zakončíme princípom distribuovaného výpočtu. V kapitole 3 sa budeme venovať existujúcim technológiám a nástrojom pre distribuovaný výpočet, ich vlastnostiam a vhodnosti pre distribuované lámanie hesiel. V kapitole 4 predstavíme návrh metodológie pre porovnanie týchto technológií s ohľadom na sledované ciele a okrem zvolených nástrojov spomenieme aj návrh vlastného riešenia pre distribúciu s využitím MPI a nástroja hashcat. Kapitola 5 je venovaná implementácií nášho nástroja a venuje sa architektúre nástroja, navrhnutému komunikačnému protokolu aj implementačným detailom. Popíšeme verziu s distribuovanou pamäťou aj implementačné rozdieli verzie so zdieľanou pamäťou. Kapitola 6 je venovaná experimentom. Popíšeme použité testovacie scenáre, vytvorené skripty na automatizáciu a zber nameraných dát, ale aj namerané hodnoty pre jednotlivé technológie vrátane ich porovnania a zhodnotenia.

Kapitola 2

Lámanie hesiel

Pod pojmom lámanie hesiel rozumieme situáciu, kedy heslo nepoznáme a snažíme sa ho získať. Pojem obnova hesiel je takmer identický, avšak občas sa interpretuje ako čiastočná znalosť hesla, teda poznáme použitú znakovú sadu, dĺžku alebo základ slova bez obmien [11].

Lámanie je založené na systematickom výbere z množiny možných hesiel a overovaní jeho správnosti. V tejto kapitole si priblížime základné princípy a prístupy ku generovaniu hesiel, ako aj spôsob overovania správnosti konkrétneho hesla. Uvedieme si tiež, aké sú možnosti akcelerácie lámania hesiel.

2.1 Šifrovanie a dešifrovanie súboru

Šifrovanie súboru alebo dokumentu prebieha väčšinou pomocou takzvanej hashovacej funkcie. Ide o matematickú funkciu, ktorá pre ľubovoľný vstup generuje výstup konštantnej dĺžky a vždy pre rovnaký vstup generuje rovnaký výstup. Aby sa predišlo útoku dúhovými tabuľkami[1], často sa pridáva do vstupu aj kryptografická soľ. V niektorých prípadoch sa pred porovnaním s overovacou hodnotou pričíta ďalšia hodnota, tzv. kryptografické korenie (ang. *pepper*). Dôvod je rovnaký, sťaženie útokov.

Hlavnou vlastnosťou hashovacej funkcie je jednosmernosť, teda že z hashu[2] nie je možné spätne získať heslo, ani pri znalosti hashovacieho algoritmu. Vďaka tejto vlastnosti môžeme zašifrovať súbor konkrétnym heslom a pripojiť jeho hash do tzv. metadát súboru, aby bolo možné pri dešifrovaní overiť správnosť hesla. (Kryptografické) metadáta sú dáta pripojené k samotnému súboru a obsahujú okrem hashu hesla aj informácie o použitom hashovacom algoritme, číslo verzie alebo bezpečnostnej revízie, kryptografickú soľ, atď. Pre väčšinu šifrovaných formátov sú metadáta to jediné, čo potrebujeme k overeniu správnosti hesla.

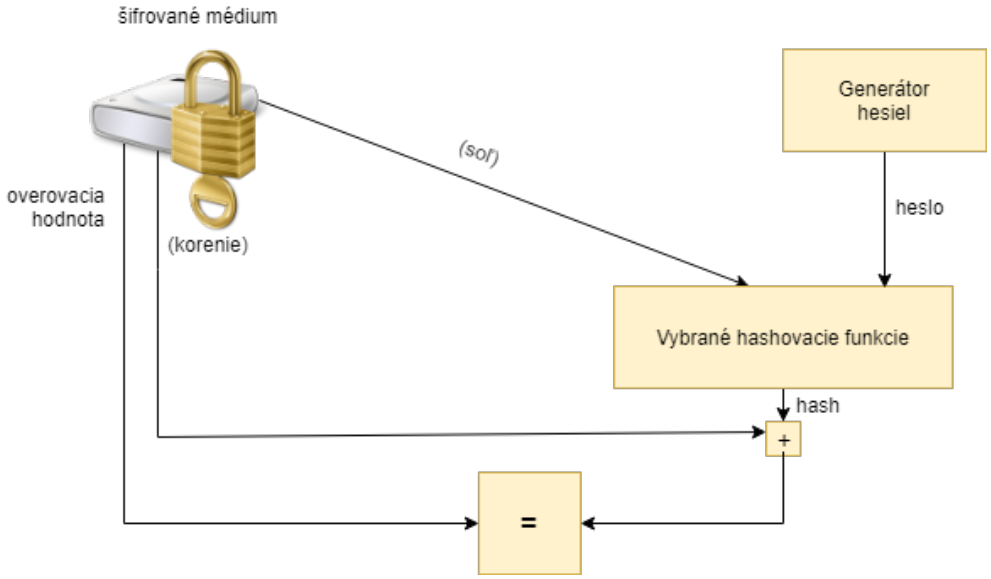
Extrakcia hashu hesla (metadát) môže byť pre väčšinu známych formátov automatizované. Pre používané nástroje na lámanie hesiel, ako napríklad Hashcat alebo John The Ripper, existujú voľne šíriteľné nástroje pre extrakciu metadát[3].

Po nájdení správneho hesla môžeme súbor jednoducho dešifrovať použitím aj pôvodného nástroja, ktorým bol súbor zašifrovaný [11, 12].

[1] Dúhové tabuľky (angl. rainbow tables) sú vopred vypočítané hashe často používaných hesiel. Ak sa hash hesla nachádza v dúhovej tabuľke, jednoducho sa z nej zistí heslo, z ktorého bol hash vypočítaný

[2] Hash – výstup hashovacej funkcie

[3] Napr. office2hashcat, pdf2hashcat, 7z2hashcat, rar2john, zip2john



Obr. 2.1: Lámanie hesla bez dešifrovania[11]

.

2.2 Základné typy útokov

Ako už bolo spomenuté, proces lámania hesiel je založený na systematickom výbere možných hesiel a pre každé heslo sa overí jeho správnosť. Tento stavový priestor všetkých možných hesiel často nazývame *priestor kľúča* alebo prebratým slovom *key space*. Lámanie hesla môže tým pádom skončiť dvoma scenármi, buď sa správne heslo v priebehu prehľadávania stavového priestoru možných hesiel nájde, alebo sa stavový priestor možných hesiel vyčerpá a heslo nebude nájdené.

Stavový priestor hesiel závisí od typu útoku, ktorým sa snažíme heslo zlomiť a vždy je generovaný *generátorom* hesiel. Medzi základné typy útokov patria:

- **Útok hrubou silou (brute-force attack)** - generuje heslá vopred definovanej dĺžky ako všetky kombinácie znakov predanej množiny znakov (*abecedy*). Nevýhoda tohto útoku je exponenciálny nárast stavového priestoru hesiel zvyšovaním dĺžky hesla, prípadne rozširovaním použitej znakovej sady. Šírka znakovej sady je uvedená v nasledujúcej tabuľke:

Znaková sada	Šírka
čísla	10
malé (ang.) písmená	26
čísla, malé a veľké písmená	62
čísla, malé a veľké písmená, špeciálne znaky	96

Tabuľka 2.1: Šírka použitej znakovej sady

Stavový priestor možných hesiel potom vypočítame ako:

$$\sum_{k=0}^{n} x^k$$

kde n je maximálna dĺžka hesla a x je šírka použitej znakovej sady. Napríklad, ak chceme útokom brute-force prelomiť heslo, o ktorom vieme, že má dĺžku maximálne 5 znakov a používa malé písmená a čísla, tak vygenerovaný stavový priestor bude mať veľkosť 62 193 780 ($32^1 + 36^2 + ... + 36^5 = 62\ 193\ 780$). Na príklade môžeme vidieť, že už pri relatívne jednoduchom hesle s malou znakovou sadou je stavový priestor hesiel pomerne rozsiahly [20].

- **Slovníkový útok** - generuje heslá čítaním zo slovníku, ktorý je definovaný textovým súborom, databázou, atď. Ide o základný typ útoku, v ktorom najčastejšie využívame uniknuté heslá z rôznych portálov. Medzi najznámejšie patrí napríklad *rockyou.txt* z rovnomenného portálu alebo *phpbb.txt* a pod.[4] Vytvoriť slovník je možné aj iným spôsobom ako získaním z reálnych (uniknutých) dát. Medzi nich patrí napríklad analýza existujúcich a následné vytváranie nových hesiel podľa pravdepodobností jednotlivých znakov (s využitím *Markovych reťazcov*). Ďalší zo spôsobov generovania slovníka je rozdelenie existujúcich hesiel na fragmenty. Tie môžeme potom ľubovoľne skladať a vytvárať tým nové heslá. V neposlednom rade môžeme využiť v dnešnej dobe veľmi populárne neurónové siete [21, 11].

- **Maskový útok** - je podobný brute-force útoku, avšak v maskovom útoku máme možnosť obmedziť generátor hesiel iba na konkrétne znaky na jednotlivých pozíciách pomocou takzvanej masky. Maskové útoky znižovaním šírky znakovej sady pre konkrétne pozície rapídne znižujú aj stavový priestor hesiel a tým aj dobu potrebnú na lámanie. Napríklad, ak sa snažíme prelomiť heslo *Pass91* útokom brute-force a vieme, že heslo obsahuje malé a veľké písmená a čísla, budeme potrebovať overiť 62^6 (56 800 235 584) možností. V prípade maskového útoku, ak vieme, že na prvej pozícií sa nachádza veľké písmeno, nasledujú tri malé písmená a heslo je zakončené dvoma číslami, tak sa stavový priestor zníži iba na 26 * 26 * 26 * 26 * 10 * 10 (45 697 600), čo je menej ako 1% pôvodného stavového priestoru brute-force útoku.

- **Hybridný útok** - je v podstate kombinácia slovníkového a maskového útoku. Jeho princíp spočíva v generovaní hesiel podľa slovníka ku ktorému pripojíme prefix resp. suffix definovaný maskou. Takýto vychádza z predpokladu, že ľudia často používajú heslá známych vzorov, ako napríklad meno + rok narodenia (*Michal985*, *Zuzana1992* atď.), ktoré môžeme pri útoku prelomiť kombináciou mena zo slovníku a roku narodenia z maskového generátoru. Populárne nástroje na lámanie hesiel podporujú dva typy hybridných útokov: heslo zo slovníka vľavo a heslo generované maskou vpravo a opačne. Veľkosť stavového priestoru hesiel hybridného útoku je potom prirodzene m x n, kde m je veľkosť slovníka kým n je počet hesiel generovaných maskou.

2.3 Existujúce nástroje

V tejto kapitole si uvedieme používané nástroje na lámanie hesiel, ich vlastnosti a možnosti použitia. Vymenujeme si tiež typy útokov, ktoré podporujú a s akými platformami sú kompatibilné.

[4] `https://wiki.skullsecurity.org/Passwords` - obsahuje uniknuté zoznamy hesiel, vhodné pre slovníkové útoky

2.3.1 Hashcat

Spomedzi nástrojov pre obnovu hesiel patrí medzi najrozšírenejšie nástroj hashcat, ktorý je voľne šíriteľný pod licenciou MIT so zverejnenými zdrojovými kódmi[5]. Podporuje stovky[6] formátov a beží na operačných systémoch Microsoft Windows, Linux aj Mac OS X. Má štandardnú podporu pre výpočet na CPU (aktuálne nazývaný *hashcat-legacy*, GPU (*oclHashcat*) a iných zariadeniach (napríklad *FPGA*) s podporou OpenCL. Medzi ďalšie užitočnosti tohto nástroja patrí *benchmark*, ktorý nám pred samotným lámaním poskytne informácie o rýchlosti na danom zariadení v jednotkách [h/s] (hash za sekundu). Dovoľuje tiež pozastaviť výpočet a následne v ňom pokračovať alebo aj automatizovanú kontrolu teploty a pod. Spomedzi nástrojov na lámanie hesiel vyniká predovšetkým rýchlosťou overovania hesiel, čo dosahuje vďaka vysoko optimalizovaným OpenCL kernelom, ktoré efektívne paralelizujú výpočet bez ohľadu na použité zariadenie [21].

Z pohľadu možných útokov, hashcat aktuálne podporuje:

- slovníkový útok (typ 0) - popísaný v 2.2,
- kombinačný útok (typ 1) - kombinuje heslá dvoch z dvoch slovníkov,
- maskový útok (typ 3) - popísaný v 2.2,
- hybridný útok - slovník + maska (typ 6), maska + slovník (typ 7) - popísaný v 2.2.

K ďalších rozšíreniam, ktorými hashcat vyniká, patria doplňujúce útoky, ktoré rozširujú slovníky o ďalšie potencionálne heslá. Ide hlavne o tieto typy útokov:

- *Rule-based* - útok založený na pravidlách. Ide o najkomplikovanejší typ útoku nástroja hashcat. Môžeme to chápať ako programovanie generátoru hesiel. Poskytuje rôzne funkcie na modifikácie, orezávanie, rozširovanie, prípadne preskakovanie hesiel zo slovníka. Často je založený na frekvenčnej analýze, pravdepodobnostnej gramatike, Markovských reťazcoch a pod.
- *Toggle-case* - útok založený na zámene veľkých písmen za malé a naopak. Aplikuje sa na každé písmeno každého hesla zo slovníka. V pôvodnej verzií (hashcat-legacy) existoval tento útok ako samostatný.

Bežné útoky môžu byť s doplňujúcimi útokmi kombinované a vytvárajú tak silný nástroj na lámanie hesiel s prirodzenými ľudskými vzormi. Nutno však podotknúť, že vyššie uvedené útoky založené na pravidlách nebudú fungovať na strojovo generovaných heslách [2, 11, 21].

2.3.2 John The Ripper

John The Ripper patrí medzi najstaršie, stále udržiavané a dostupné nástroje na lámanie hesiel. Ide o voľne šíriteľný program pod licenciou GPLv2[7]. Pôvodne bol vyvinutý iba pre Unixový operačný systém (na lámanie Unixových užívateľských hesiel), ale dnes má podporu pre všetky používané platformy (Microsoft Windows, Linux, Mac OS X).

Podporuje tri režimy: *single*, *wordlist* a *incremental*. *Single* a *wordlist* režimy skúšajú najskôr heslá, ktoré môžu byť pravdepodobnejšie správne. Očakávajú na vstupe slovník, ktorý je následne možné modifikovať prepisovacími pravidlami (ang. *mangle*). Tieto dva

[5] https://github.com/hashcat
[6] Viac na https://hashcat.net/wiki/doku.php?id=example_hashes
[7] General Public License version 2 - https://www.gnu.org/licenses/old-licenses/gpl-2.0.cs.html

režimy sa líšia v tom, že *single* je primárne určený na lámanie Unixových hesiel a vychádza z GECOS[8], na ktoré aplikuje všetky dostupné modifikácie. Ide pomerne o rýchly útok, ktorý môže trvať od niekoľko milisekúnd po jeden deň. *Wordlist* režim je obdobný s rozdielom, že vychádza z predaného slovníku. Užívateľ môže tiež zapnúť modifikácie a vybrať si tie, ktoré chce použiť k útoku. Následne, všetky modifikácie sa použijú na každý riadok (heslo) priloženého slovníka. *Incremental* režim je najsilnejší, samozrejme aj výpočtovo najnáročnejší a pri rozsiahlych úlohách nemusí v rozumnom čase skončiť a je potrebné ho predčasne prerušiť. Vyskúša všetky možné kombinácie (obdoba brute-force útoku popísanom v 2.2) a preto má spomedzi všetkých režimov najväčší stavový priestor hesiel. V konfiguračnom súbore očakáva parametre útoku, ako napríklad minimálna/maximálna dĺžka hesla, použitá znaková sada, atď[9].

Z pohľadu akcelerácie, John The Ripper má podporu paralelizácie na úrovni procesora a jadier prostredníctvom technológie OpenMP[10] a tiež aj na medziprocesorovej úrovni technológiou MPI [11][4, 17].

2.3.3 Cain & Abel

Nástroj Cain & Abel mal úspech hlavne medzi užívateľmi systému Windows, pretože to bola jediná platforma kde bežal, avšak dnes už je nepodporovaný a nevyvíjaný [20]. Bol užívateľsky prívetivý a bol integrovaný s inými nástrojmi, napríklad pre odpočúvanie sieťovej komunikácie (ang. *network packet sniffing*) a následnej extrakcie hashu. Bolo ním možné tiež vykonávať iné sieťové útoky, ako napríklad *ARP Poisoning* alebo *Man In The Middle* útok.

Napriek týmto výhodám mal Cain & Abel mnoho nedostatkov, medzi ktoré patrí napríklad slabá podpora pravidiel pri útokoch (obsahoval iba zámeny čísel za písmená, zámeny veľkých a malých písmen a pripájanie čísel na koniec hesla). V skratke, išlo o dobre navrhnutý nástroj (nielen) na lámanie hesiel, avšak v aktuálnom stave je vhodný iba na lámanie slabých hesiel [1, 20].

2.4 Možnosti distribúcie

Proces lámania hesiel si vzhľadom na svoju náročnosť vyžaduje mnoho výpočtovej sily. Trendu zvyšovania počtu jadier v procesoroch (namiesto zvyšovania frekvencie) sa prispôsobili aj vývojári nástrojov na lámanie hesiel a dnes je preto už samozrejmosťou paralelizácia výpočtu. Tento proces je možné efektívne paralelizovať vďaka tomu, že pozostáva z mnoho na sebe nezávislých čiastkových úloh. Zjednodušene povedané, za jednu jednotku času overíme namiesto jedného hesla n hesiel, kde n je počet jadier. Je teda prirodzené prenášať výpočet na zariadenia, ktoré majú čo najviac procesorových jadier.

Ako riešenie sa nám ponúka GPGPU[12], teda výpočet na grafických kartách (GPU), ktoré majú v dnešnej dobe stovky až tisícky jadier. V prípade karty GTX 1080 Ti je to 3584 jadier[13]. Porovnanie voči CPU je znázornené na obrázku 2.2.

[8] GECOS - Všeobecné informácie o užívateľovi. Viac na môže čitateľ dočítať na https://www.freebsd.org/cgi/man.cgi?query=passwd&sektion=5

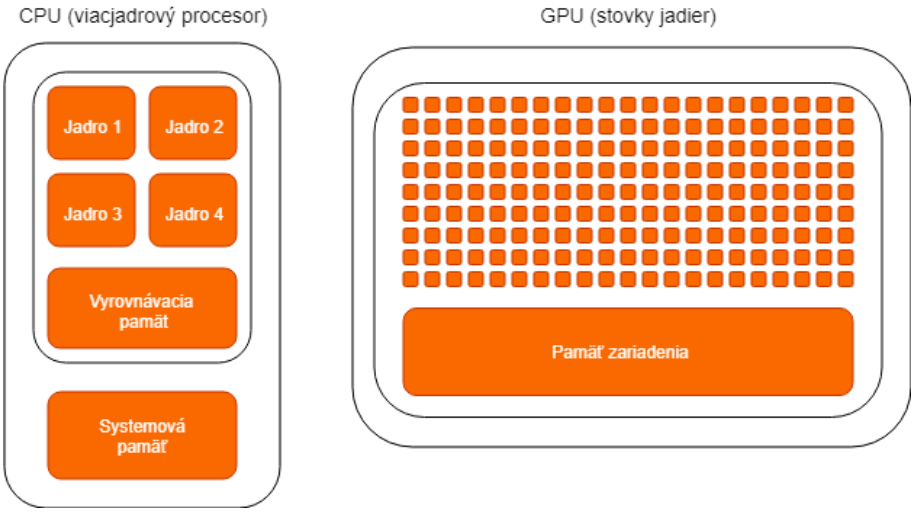
[9] Viac na https://www.openwall.com/john/doc/MODES.shtml

[10] Viac na https://www.openmp.org/

[11] MPI (ang. *Message Passing Interface*) - štandard pre komunikáciu medzi procesormi

[12] GPGPU (General-purpose computing on graphics processing units) - spôsob využitia paralelizácie na grafickej karte k výpočtu obecných algoritmov

[13] https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/



Obr. 2.2: Porovnanie CPU a GPU

Aj pri najvýkonnejších kartách sme však stále obmedzený počtom kariet v jednom počítači. Ako príklad môžeme uviesť lámanie hesla populárneho archívu WinZIP zašifrovaného symetrickým AES[14] algoritmom. Na výkonnom zariadení, zloženom z 8 x GPU Nvidia GTX 1080 Ti FE pri relatívne slabom hesle, ktoré obsahuje iba veľké a malé znaky a má dĺžku 8 znakov bude trvať lámanie (pri preskúmaní celého stavového priestoru) viac ako 74 dní. V prípade ak heslo obsahuje aj čísla, dostávame sa zhruba na 1 rok, čo je pre väčšinu účelov neprijateľné [14]. Tento čas sa dá znižovať nielen vylepšovaním samotných útokov a generátorov hesiel, ale aj zvyšovaním výpočtovej sily.

Ako jediné vhodné riešenie, prekonávajúce limity jedného počítača, je *distribuovaný výpočet*. Jeho základná myšlienka spočíva v spájaní výpočtovej sily viacerých fyzických zariadení do jedného celku. Väčšinou ide o zariadenia komunikujúce prostredníctvom TCP/IP protokolu zapojené v určitej sieťovej topológií (centralizovaná/decentralizovaná, hviezda, kruh, atď.).

Pri lámaní hesiel v distribuovanom prostredí sa spravidla využíva klient-server architektúra, v ktorej server zbiera dáta od klientov, rozdeľuje im úlohy a poskytuje rozhranie pre užívateľa. Klienti, ktorých môže byť ľubovoľný počet, počítajú čiastkové úlohy (ang. *chunks*) a o výsledkoch informujú server. Takto sa celý stavový priestor hesiel rozdelí na niekoľko úloh, ktoré sú predávané klientom.

Pri distribuovanom výpočte musíme brať do úvahy spotrebované zdroje nielen na samotné lámanie, ale aj na réžiu. Tam zaraďujeme dobu potrebnú na prijatie úlohy s jej parametrami, odosielanie výsledkov serveru, čakanie na úlohy, prenos prípadného slovníka potrebného k útoku, ale aj beh benchmarku pre zistenie rýchlosti konkrétneho klienta.

[14]Viac na https://cs.wikipedia.org/wiki/Advanced_Encryption_Standard

Kapitola 3

Technológie pre distribuovaný výpočet

3.1 VirtualCL

VirtualCL (skrátene VCL) je wrapper nad frameworkom OpenCL, ktorý dovoľuje nemodifikovaným aplikáciám aby využívali OpenCL zariadenia v clusteri (zoskupenie prepojených zariadení) akoby boli všetky lokálne v jednom zariadení. Medzi jeho základné vlastnosti patrí, že:

- dokáže pracovať s OpenCL zariadeniami (CPU, GPU, atď.) všetkých značiek,
- podporuje ho takmer každú OpenCL 1.1 (a 1.0) aplikáciu,
- umožňuje transparentný výber z dostupných OpenCL zariadenia,
- podporuje beh viacerých aplikácií v rovnakom clusteri,
- hostiteľský uzol nemusí podporovať OpenCL.

VirtualCL komunikuje medzi hostiteľským a vzdialenými uzlami štandardnými TCP/IP schránkami. Za limitujúci faktor pri vzdialenom behu OpenCL aplikácií môžeme jednoznačne označiť latenciu siete. V tabuľke 3.1 sú porovnané rýchlosti bežného OpenCL s lokálnym a vzdialeným VirtualCL pre niektoré SHOC (skr. *Scalable HeterOgeneous Computing*) testy [8]. Výsledný čas bol priemer 5 meraní a použité zariadenia na výpočtových uzloch boli grafické karty NVIDIA GeForce GTX 480 (Fermi)

Aplikácia	OpenCL čas [s]	VirtualCL čas [s]	
		Lokálny	Vzdialený
BusSpeedDownload	0,89	0,88	0,88
DeviceMemory	31,44	56,78	243,81
Triad	6,01	11,83	53,37
S3D	32,39	32,68	33,17
MD	14,08	13,66	13,80

Tabuľka 3.1: SHOC Benchmark - porovnanie

Zo SHOC testov bolo jasne vidieť, že réžia na VirtualCL nie je zanedbateľná. Testy ako napríklad *MD*, *S3D* alebo *BusSpeedDownload* sú ideálne pre VirtualCL, pretože majú

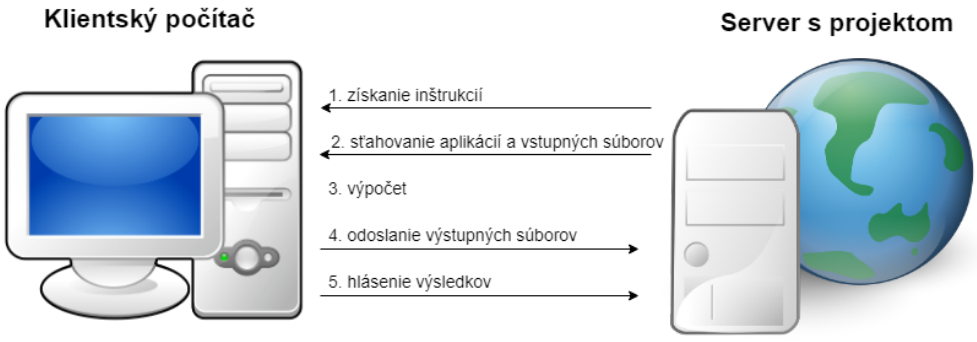
relatívne malé vstupné dáta (ktoré treba prenášať) a dlhé výpočtové kerneli. Naopak, testy ako *DeviceMemory* alebo *Triad* dosiahli veľmi slabé výsledky, pretože obsahovali väčší prenos dát a kratšie výpočtové kerneli. Špeciálne sa pozrime na test *DeviceMemory*, ktorý neobsahuje žiadny výpočet, iba vytvorí kópiu dát. Na tomto teste možno vidieť réžiu, ktorú pridalo VirtualCL. Lokálna verzia znamenala kopírovanie dát (cez UNIX-ové sockety) a vzdialená verzia tiež kopírovanie dát plus TCP/IP sieťový prenos [6].

Z pohľadu lámania hesiel VirtualCL nebude najvhodnejší kandidát, práve z dôvodu vysokej réžie na sieťový prenos dát. Táto réžia sa dá znížiť zapojením výpočtových uzlov (ideálne každý s viacerými OpenCL zariadeniami, rozsiahlou pamäťou a mnoho procesorovými jadrami) vysokorýchlostnými linkami s nízkou latenciou a tiež minimalizovaním prenosu dát a vykonávaním čo najväčších kernelov. Na druhú stranu, VirtualCL nepotrebuje žiadny nástroj ani úpravu aplikácií (napísaných pre OpenCL) a teda nie je potrebné riešiť problémy súvisiace s distribúciou výpočtu. Stačí jednoducho nainštalovať VirtualCL a nakonfigurovať sieť výpočtových uzlov.

3.2 BOINC

BOINC (skratka od *Berkeley Open Infrastructure for Network Computing*) je platforma vyvinutá výskumnou skupinou na Univerzite v Kalifornií v Berkeley pod vedením Davida Andersona[1]. Je založený na myšlienke, že väčšina počítačov nevyužíva svoj výpočtový výkon a podieľanie sa na rozsiahlych výskumných výpočtoch neovplyvní beh ani odozvu ostatných užívateľských aplikácií. Má zverejnené zdrojové kódy[2] a spadá pod licenciu LGPL, teda je ho možné použiť ako na voľne šíriteľný software, tak aj na komerčný software.

Architektúra systému BOINC je založená na klient-server modeli. Server zodpovedá za správu úloh, vytváranie čiastkových úloh, ich plánovanie atď. Klient reprezentuje výpočtový uzol, ktorý poskytuje svoju výpočtovú silu. V rámci jednej úlohy na riešenie (nazývanej ang. *project*) server vytvorí množinu čiastkových úloh (ang. *task*), ktoré riešia klienti samostatne a nezávisle na sebe. Jeden klient môže v rovnakom čase riešiť viac čiastkových úloh. Priebeh úlohy je zobrazený na obrázku 3.2.



Obr. 3.1: Riešenie úlohy cez BOINC

V prvom kroku odošle server klientovi inštrukcie k zadaniu úlohy. Zadanie pre každého klienta môže byť odlišné a závisí od jeho architektúry, operačného systému, hardwaru, atď.

[1] David Andersen je tiež riaditeľ najväčšieho projektu založenom na BOINC - *Seti@Home* (skrátene *Search for Extra-Terrestrial Intelligence*), ktorý bol spustený v roku 1999 a slúžil na distribúciu výpočtov medzi širšiu verejnosť na hľadanie mimozemských civilizácií

[2] https://github.com/BOINC/boinc

Následne sa automatizovane odošlú klientovi aplikácie potrebné k riešeniu úlohy. Akonáhle má klient všetky potrebné dáta a nástroje, začne sa výpočet, ktorý môže trvať od niekoľko minút až po niekoľko dní. Po dokončení úlohy sa výstup úlohy s reportom odošlú serveru a klient čaká na ďalšiu úlohu.

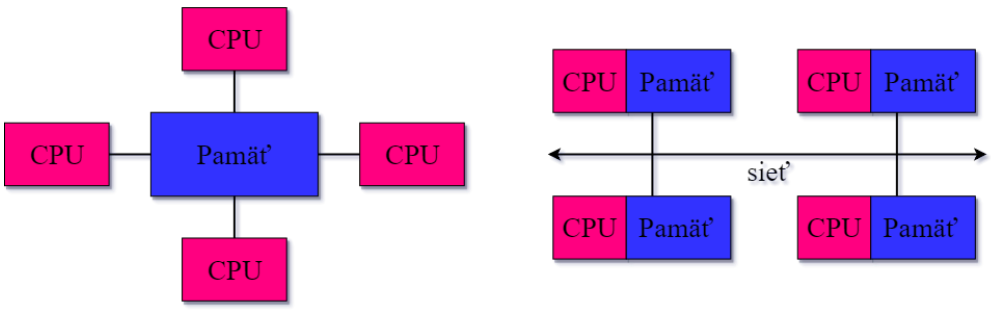
Medzi hlavné výhody platformy BOINC patrí, že klienti (výpočtové uzly) sa môžu dynamicky pripájať a odpájať z bežiacej úlohy. Môžu tiež rozhodovať koľko percent využitia CPU či GPU sú ochotní venovať výpočtu, koľko operačnej pamäte, diskového priestoru aj sieťových zdrojov (upload/download) [11, 12].

Fitcrack[3] je nástroj využívajúci platformu BOINC k distribuovanému lámaniu hesiel. Je voľne dostupný pod MIT licenciou a vyvíjaný výskumnou skupinou NES@FIT na Fakulte informačných technológií, Vysoké učení technické v Brně. K lámaniu na strane klientov využíva nástroj hashcat (popísaný v 2.3.1), čo zaručuje vysokú rýchlosť lámania na akomkoľvek OpenCL zariadení (CPU, GPU, FPGA, DSP či koprocesor). Okrem výhod platformy BOINC poskytuje Fitcrack aj automatizovanú extrakciu metadát a hashu zo súboru či voliteľné veľkosti čiastkových úloh (*chunk*). Obzvlášť za zmienku stojí progresívny prístup pri slovníkových útokoch, kedy sa namiesto prenosu celých (často rozsiahlych) slovníkov všetkým klientom prenáša iba alikvotná časť. To môže dramaticky znížiť réžiu, hlavne ak je klientov mnoho a útok je relatívne rýchly (v pomere k potrebnému času na prenos slovníka) [13].

3.3 MPI

MPI (skratka od *Message Passing Interface*) je špecifikácia protokolu pre zasielanie správ medzi procesormi, jadrami či vláknami. Samotné MPI nie je teda knižnica, ale skôr špecifikácia toho, čím by knižnica na zasielanie správ mala byť. Cieľom a motiváciou bolo navrhnúť rozhranie pre zasielanie správ, ktoré bude praktické, prenosné, efektívne a flexibilné. Aj napriek tomu, že MPI nie je súčasťou IEEE alebo ISO štandardov, stal sa akýmsi priemyselným štandardom pre vývoj aplikácií na HPC (ang. *high-performance computing*) platformy. MPI vznikol v roku 1992 a prešiel už viacerými úpravami až po najnovšiu verziu MPI-3.x (verzia MPI-4.0 sa aktuálne vyvíja a testuje).

Pôvodne bolo MPI navrhnuté pre architektúry s distribuovanou pamäťou (čo bolo populárne v 80-tych a začiatkom 90-tych rokov) a neskôr, ako sa trendy v architektúrach menili, boli pridané podpory aj pre architektúry so zdieľanou pamäťou a tiež hybridné architektúry.



Obr. 3.2: Zdieľaná vs. distribuovaná pamäť

[3] https://fitcrack.fit.vutbr.cz/

Existuje viacero voľne šíriteľných aj komerčných implementácií tohto štandardu, z ktorých za zmienku stoja OpenMPI[4], MPICH[5] (a z neho odvodený MVAPICH[6]) a IntelMPI[7].

OpenMPI je open-source implementácia MPI špecifikácie, ktorá vychádza z predchodcov FT-MPI, LA-MPI, LAM/MPI a PACX-MPI. Napriek relatívne krátkej histórií s veľkými cieľmi ide o stabilnú implementáciu, ktorá síce má rezervy v podobe slabých algoritmov kolektívnych operácií (napr. MPI_Reduce, MPI_Gather, MPI_Bcast, atď), ale zato má veľmi efektívne implementované jednoduché zasielanie správ, čo značí o dobrej štruktúre jadra implementácie. Vývoj je dynamický a aktuálne (od Novembra 2018) je možné aj experimentálne použiť verziu OpenMPI-4.0.

MPICH, niekedy známy aj pod názvom *MPICH2* je open-source implementácia MPI špecifikácie. Jej hlavným cieľom je maximálna prenositeľnosť, podobne ako prispôsobivosť chameleóna, z ktorého vychádzal aj názov jeho predchodcu MPICH1 (ang. *MPI Chameleon*). MPICH je úplne nová implementácia s ohľadom na roky skúseností z vývoju a údržby MPICH1. Z pohľadu výkonnosti je lepší ako OpenMPI a má tiež lepšie implementovanú prácu so zdieľanou pamäťou [18, 5].

IntelMPI je komerčná implementácia vychádzajúca z MPICH s podporou špecifikácie MPI-3. Ide rovnako o prenositeľnú knižnicu dodávanú s množstvom dodatočných nástrojov na analýzu kódu, profiling, atď. Implementáciu komunikácie okrem štandardného TCP/IP dopĺňa aj o ďalšie protokoly s nízkou latenciou, ako napr. Omni-path či iWarp [3].

Hotové nástroje na lámanie hesiel využívajúce MPI neexistujú. Nástroj John The Ripper (popísaný v 2.3.2) podporuje od verzie *1.7.7-jumbo-5* čiastočné rozšírenie pre MPI. Pre jeho využitie je potrebné zmeniť niektoré parametre pri preklade nástroja[8]. Experimentálne sa kombinácií John The Ripper a MPI venoval *Tyler Lubeck* v [17].

3.4 Protokol nad HTTP

Riešením pre komunikáciu v distribuovanom prostredí môže byť aj vlastný (prípadne existujúci) protokol nad protokolom HTTP [10]. Výhodou je možnosť pripájania výpočtových uzlov cez internet (mimo lokálnu sieť) aj z obmedzených sietí (ktoré môžu mať povolené iba základné typy protokolov, medzi ktoré HTTP určite patrí) a celková robustnosť.

Ako open-source reprezentanta tohto spôsobu distribúcie môžeme považovať nástroj *Hashtopolis*[9], ktorý je aj jedným z odporúčaných riešení hashcatu pre distribúciu výpočtu. Hashtopolis využíva pre zasielanie dát formát JSON[10], ktorý následne odošle protokolom HTTP. Súčasťou Hashtopolisu je aj inštalátor pre jednoduchšie nasadenie a webové rozhranie, pomocou ktorého je možné sledovať počet pripojených klientov vrátane ich podielu na riešení úlohy, rýchlosť lámania, celkový stav úlohy atď. Podporuje tiež distribúciu nástroja na lámanie (hashcat) klientom, vytváranie užívateľov a ich priraďovanie k úlohám, ale aj lámanie viacerých hashov (rovnakého typu) naraz.

[4] https://www.open-mpi.org/
[5] https://www.mpich.org/
[6] http://mvapich.cse.ohio-state.edu/
[7] https://software.intel.com/en-us/mpi-library
[8] Viac na https://openwall.info/wiki/john/parallelization
[9] https://github.com/s3inlc/hashtopolis
[10] Java Script Object Notation, viac na https://www.json.org/

Kapitola 4

Návrh metodológie pre porovnanie distribuovaných technológií

Táto kapitola sa bude venovať návrhu metodológii pre porovnanie distribuovaných technológií. Cieľom práce je zistiť ich vhodnosť na lámanie hesiel a ich výhody a nevýhody. To si vyžaduje vytvorenie testovacieho prostredia, voľbu konkrétnych nástrojov využívajúcich dané technológie, vytvorenie sieťovej topológie, určiť spôsob merania sledovaných hodnôt a pod.

4.1 Ciele porovnania

Aby sme dokázali technológie porovnať z hľadiska lámania hesiel, je nutné si vopred zadefinovať podľa akých kritérií ich budeme posudzovať.

4.1.1 Výkonnosť

Výkonnosťou myslíme celkový výkon distribuovaného systému meraný počtom overených hesiel (hashov) za sekundu. Zvyšujúcou sa výkonnosťou sa znižuje celkový čas potrebný na vykonanie útoku. Výkonnosť sa líši v závislosti od zložitosti použitého hashovacieho algoritmu a preto tento rozdiel nie je ovplyvniteľný distribúciou výpočtu.

4.1.2 Réžia

Na rozdiel od lámania hesiel na jednom počítači, pri distribuovanom lámaní musíme brať v úvahu i čas potrebný na distribúciu informácií o útoku (napr. typ útoku) a potrebných dát (napr. slovník, maska, hash lámaného hesla). Ďalej je potrebná priebežná komunikácia medzi riadiacim uzlom (serverom) a výpočtovými uzlami (klientami), ktorým sú odosielané čiastkové úlohy (časť stavového priestoru hesla). Tie môžu mať buď fixnú veľkosť alebo variabilnú (v závislosti na rýchlosti daného výpočtového uzlu). Variabilná veľkosť, ktorá minimalizuje čakanie na najpomalší uzol, si vyžaduje meranie rýchlosti (ang. *benchmark*) pred začatím samotného lámania. Tieto, a aj ďalšie faktory do značnej miery ovplyvňujú čas potrebný na lámanie hesiel v distribuovanom prostredí a preto réžiu distribúcie považujeme za kľúčový faktor. Meriame ju v percentuálnych bodoch ako pomer času stráveného distribúciou k celkovému času lámania

4.1.3 Efektivita

Efektivita je doplnok k réžií a vyjadruje pomer (strojového) času, počas ktorého výpočtové uzly lámali heslo k celkovému času lámania. Vypočítať ju môžeme ako:

$$Eff = \frac{\sum_{x=1}^{N} t_x}{N \times T_{fin}}$$

kde N je počet uzlov podieľajúcich sa na útoku, t_x je čas strávený lámaním pre uzol x a T_{fin} udáva celkový čas útoku [14].

Cieľom distribuovaného systému je dosahovať čo najvyššiu efektivitu a tým znižovať nielen čas potrebný na réžiu, ale aj trvanie celého útoku.

4.1.4 Škálovateľnosť

S počtom pribúdajúcich uzlov sa réžia zvyšuje, keďže je potrebné koordinovať a rozdeľovať keyspace pre viacero uzlov. Je tiež potrebné odoslať všetky potrebné dáta všetkým uzlom, čo v prípade napr. slovníkového útoku môže byť časovo náročné. Škálovateľnosťou vyjadrujeme ako sa mení réžia pri zmene počtu výpočtových uzlov.

4.1.5 Prispôsobivosť

V rámci tohto cieľa budeme sledovať a merať, ako sa distribuovaný systém dokáže vysporiadať s homogénnym (rovnako výkonné uzly) a heterogénnym (uzly rôzne výkonné) prostredím výpočtových uzlov. Adaptívny plánovač úloh by mal byť schopný rozdeľovať úlohy tak, aby sa minimalizovala doba čakania na pomalšie uzly (hlavne v závere útoku).

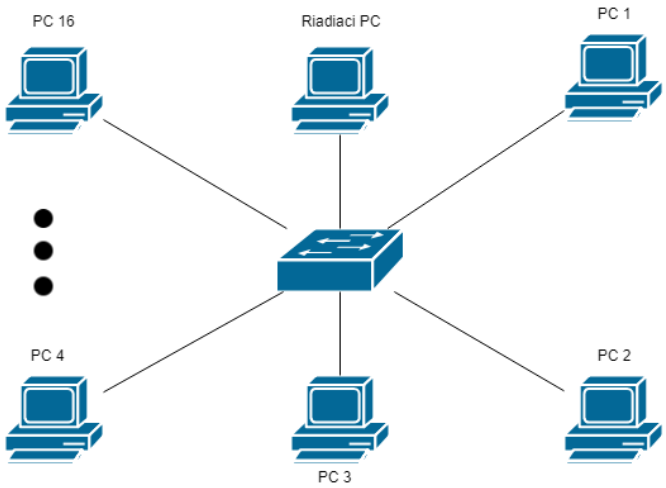
4.2 Návrh prostredia

Keďže cieľom práce je porovnanie systémov pre distribúciu, budeme sa snažiť riadiť pravidlom *ceteris paribus*, čo v našom prípade znamená, že pri sledovaní vplyvu distribuovaného systému musíme ostatné parametre (ovplyvňujúce výsledok) zachovať nezmenené. Iba tak bude možné merať reálny dopad distribúcie na celkový výsledok.

Merania prebehnú vo výskumnej laboratórií C304, kde sa nachádzajú osobné počítače a potrebná sieťová infraštruktúra. Každý z počítačov obsahuje:

- CPU - Intel Core i5-3570K,
- GPU - Nvidia GTX 1050 Ti,
- operačnú pamäť typu DDR3 - 8GB.

Pre experimenty bude použitá topológia hviezdy. Ako centrálny bod bude použitý prepínač *HP ProCurve 3400cl* a jednotlivé počítače budú ním prepojené sieťovým káblom s rýchlosťou 1 Gbps, viď 4.1.



Obr. 4.1: Návrh sieťovej topológie

4.3 Voľba nástrojov

Experimenty, aby boli zachované rovnaké podmienky, musia využívať rovnaký nástroj na obnovu hesiel. Ako najvhodnejší sa nám javí nástroj hashcat (popísaný v 2.3.1), ktorý vyhral 5 zo 7 ročníkov súťaže *Crack me if you can*[1]. Okrem vysokej rýchlosti lámania hesiel ponúka hashcat množstvo podporovaných formátov a rôzne typy útokov. V neposlednom rade má podporu pre všetky používané operačné systémy v 32-bitovej aj 64-bitovej verzii a dokáže pracovať na rôznych OpenCL zariadeniach, vrátane CPU, GPU, FPGA, DSP a ko-procesorov.

Pre mnoho technológií pre distribúciu už existuje distribuovaný nástroj na lámanie hesiel, ktorý je možné použiť pre experimentálne porovnania. Z kapitoly 3 sme pre jednotlivé technológie vybrali nasledovné nástroje:

- **VirtualCL** - nevyžaduje nástroj, keďže ide abstraktnú vrstvu nad OpenCL zariadeniami. VirtualCL bol prvý odporúčaný spôsob distribúcie hashcatom, avšak v tejto dobe už nie je udržiavaná kompatibilita a na možnú nefunkčnosť upozorňujú aj na ich webových stránkach s manuálom konfigurácie VirtualCL[2]. Napriek tomu a predpokladu, že ide o nevhodného kandidáta na lámanie hesiel, sme sa ho rozhodli zaradiť medzi analyzované nástroje. Pre vykonanie experimentov bude potrebné nakonfigurovať sieť počítačov a zvoliť si riadiaci (spúšťací) uzol. VirtualCL bolo pre hashcat dostupné pre operačný systém Linux v 64-bitovej verzii.

- **BOINC** - ako nástroj využívajúci technológiu BOINC sme zvolili **Fitcrack** (popísaný v 3.2). Pre vykonanie experimentov je potrebné nainštalovať na riadiaci uzol (nemusí byť výpočtový) Fitcrack server a na všetky výpočtové uzly BOINC client s voliteľným BOINC managerom. Fitcrack server obsahuje tiež webové rozhranie, ktorým je možné spúšťať útoky, vyberať klientov pre útok a monitorovať ich. Fitcrack je možné použiť na operačnom systéme MS Windows aj Linux v 64-bitovej verzii.

[1] https://contest.korelogic.com/
[2] https://hashcat.net/wiki/doku.php?id=vcl_cluster_howto

- **Vlastný protokol nad HTTP** - zvolený nástroj pre distribúciu lámania hesiel využívajúci vlastný protokol je **Hashtopolis** (popísaný v 3.4). Keďže ide o klient-server architektúru, je potrebné mať nainštalovaný hashtopolis server (ktorý poskytuje aj webové rozhranie) a na klientskych počítačoch hashtopolis client, ktorý následne treba zaregistrovať na požadovaný server. Hashtopolis client má podporu ako pre MS Windows (v podobe .NET binárneho súboru), tak aj pre Linux (v podobe Python skriptu).

- **MPI** - v minulosti boli implementované prototypy distribuovaných nástrojov na lámanie hesiel nad MPI, ale využívali iné (niektoré z nich vlastné) nástroje na lámanie (napr. *Wisecrack*[3] alebo *Wrathion*[16]). Vykazovali sľubné výsledky, avšak pre porovnanie s ostatnými technológiami bude potrebná vlastná implementácia distribúcie s využitím nástroja hashcat. V nasledujúcej kapitole 4.4 je popísaný návrh implementácie.

Experimenty sme sa rozhodli robiť na operačnom systéme Linux (konkrétne CentOS), pretože je dostupný v laboratóriu a podporujú ho všetky spomenuté distribuované nástroje. Okrem toho, nami implementované riešenie bude využívať MPI, ktoré ma výbornú podporu pre Linux.

4.4 Návrh distribuovaného nástroja nad MPI

S ohľadom na zachovanie rovnakých podmienok pri porovnávaní distribuovaných nástrojov je potrebné vytvoriť nástroj využívajúci technológiu MPI a nástroj na lámanie hashcat. Porovnania budú realizované na operačnom systéme Linux. Pre vývoj nástroja sme sa podľa informácií z 3.3 rozhodli využiť MPI implementáciu MPICH, pretože spomedzi voľne šíriteľných implementácií dosahuje pre naše účely najlepšie výsledky [18]. Ako implementačný jazyk sme sa rozhodli použiť Python s rozšírením `mpi4py`[4], ktoré poskytuje intuitívne a jednoduché rozhranie pre Python nad MPI [7]. Rovnako ako pri samotnom MPI, ide iba o rozhranie bez implementácie, preto ak sa v istom momente rozhodneme zmeniť MPICH za MVAPICH (napríklad kvôli prechodu z rozhrania Ethernet[5] na Infiniband[6]), stačí jednoducho zmeniť príslušné knižnice.

Aplikácia bude podporovať maskový útok (ktorým je v nástroji hashcat možné simulovať aj útok hrubou silou, popísané sú v 2.2) a slovníkový útok (popísaný v 2.2). Rozhranie bude odvodené z nástroja hashcat[7] s rozdielom, že iba pre podporované útoky, formáty a funkcie.

Ako príklad použitia môžeme uviesť:

- `mpiHashcat.py -a 0 -m 0 MD5_HASH dict.txt` pre slovníkový útok na hash typu MD5 [19] alebo

- `mpiHashcat.py -a 3 -m 100 SHA1_HASH ?u?l?l?l?d?d` pre maskový útok na hash typu SHA-1 [9].

Pre distribúciu potrebných dát (slovník, masky, prípadne binárne súbory) na výpočtové uzly sme navrhli dva prístupy, ktoré sú popísané v nasledovných sekciách.

[3] Viac na https://github.com/vikasnkumar/wisecracker

[4] Viac na https://mpi4py.readthedocs.io/en/stable/

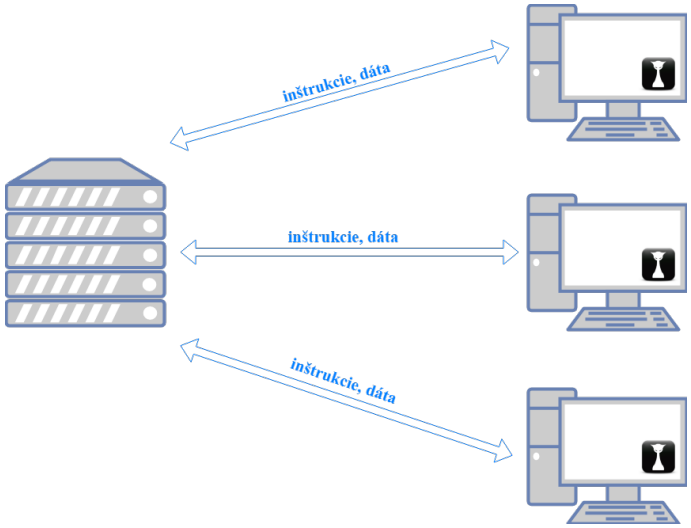
[5] http://www.ieee802.org/3/

[6] Štandard sieťového rozhrania s veľmi nízkou latenciou využívaný v HPC (*High-performance computing*)

[7] Argumenty príkazového riadku nástroja hashcat môže čitateľ nájsť na https://hashcat.net/wiki/doku.php?id=hashcat

4.4.1 Prístup bez zdieľanej pamäti

Prvý z prístupov využije pre distribúciu potrebných dát volania MPI (`MPI_Send` a `MPI_Recv`). Tieto dáta sa prenesú z riadiaceho uzlu na výpočtové uzly pred začiatkom výpočtu. Vyžaduje sa, aby bol nástroj na lámanie (hashcat) nainštalovaný a umiestnený na rovnakom mieste v každom výpočtovom uzle. Nevýhodou tohto riešenia je zvýšená počiatočná réžia na prenos dát, avšak následné operácie (napr. čítanie z lokálnej kópie slovníka) budú rýchle. Situácia je znázornená na obrázku 4.2.



Obr. 4.2: Návrh riešenia bez zdieľanej pamäte

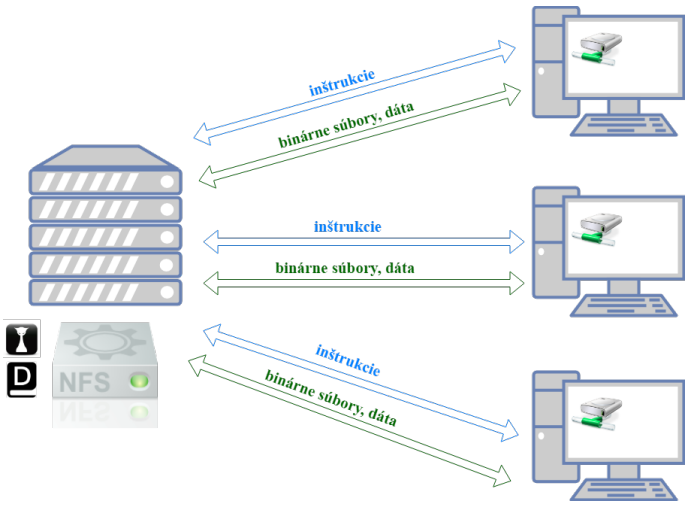
Na obrázku môžeme vidieť, že v tomto návrhu budú rovnakým komunikačným kanálom (MPI) prenášané aj dáta aj riadiace inštrukcie.

4.4.2 Prístup so zdieľanou pamäťou

Ako alternatíva k prvému prístupu je návrh so zdieľanou pamäťou. Implementácia zdieľanej pamäte bude riešená službou NFS[8] (*Network File System*), pomocou ktorej bude priečinok riadiaceho uzlu s dátami viditeľný výpočtovým uzlom v lokálnej sieti. Jednotlivé uzly ho namapujú ako sieťový priečinok službou `mount`[9]. Následne, pri samotnom lámaní budú všetky uzly pristupovať k tomuto zdieľanému priečinku kde nájdu dáta (slovník, nástroj hashcat) potrebné k útoku. Výhodou tohto riešenia je jednoduchšia konfigurácia výpočtového uzlu (nie je potrebné inštalovať nástroj hashcat). Nevýhodou a úzkym hrdlom pri slovníkových útokoch s väčšími slovníkmi môže byť prístup na pevný disk riadiaceho uzlu. Súčasný prístup viacerých uzlov na rôzne pozície toho istého súboru môže spôsobiť veľké latencie. Dopad prístupov na disk by sa čiastočne dal znížiť namapovaním zdieľaného priečinku na operačnú pamäť RAM, ktorá má oproti pevnému disku omnoho rýchlejšie náhodné prístupy do pamäti. Pri takomto vylepšení budeme však limitovaný veľkosťou operačnej pamäte, ktorá býva spravidla menšia ako veľkosť pamäte pevného disku. Výhodou tohto riešenia je, že pri kratších útokoch (správne heslo sa nachádza na začiatku slovníka) nebude potrebné

[8] Viac na https://searchenterprisedesktop.techtarget.com/definition/Network-File-System
[9] Viac na http://man7.org/linux/man-pages/man8/mount.8.html

prenášať celý slovník všetkým výpočtovým uzlom. Taktiež sa znižujú pamäťové nároky výpočtových uzlov (nie je potrebné ukladať lokálne kópie dát).



Obr. 4.3: Návrh riešenia so zdieľanou pamäťou

Na obrázku 4.3 sú znázornené dva komunikačné kanály, prvý (modrý) slúži na prenos inštrukcií a riadiacich informácií cez rozhranie MPI a druhý (zelený) slúži na prenos dát a binárnych súborov cez službu NFS.

Kapitola 5

Implementácia distribuovaného nástroja nad MPI

Nutnosť implementácie distribuovaného nástroja nad MPI vychádza z požiadaviek na porovnanie technológií, keďže MPI patrí k najrozšírenejším technológiám pre komunikáciu medzi procesmi a v súčasnosti neexistuje podľa našich vedomostí nástroj na lámanie hesiel (okrem spomenutého rozšírenia pre nástroj John The Ripper v kapitole 2.3.2), ktorý by túto technológiu využíval.

V tejto kapitole bude popísaná implementácia vlastného nástroja nad technológiou MPI s využitím nástroja na lámanie hesiel hashcat. Nástroj je implementovaný v dvoch variantoch, prvá využíva distribuovanú pamäť a druhá zdieľanú. Obe varianty majú rovnaké rozhranie príkazového riadku a oba nesú názov `mpiHashcat`. Užívateľ ich rozlíši umiestnením spúšťacieho skriptu.

5.1 Prerekvizity a závislosti

Nami implementovaný nástroj je naprogramovaný v programovacom jazyku `Python`[1] verzie 2.7, ktorá sa považuje za poslednú veľkú verziu rady 2.x s množstvom doplnkov uvedených vo verzií Python 3.1. Verzia 2.7 bola zvolená s ohľadom na požiadavky implementovanej aplikácie a optimalizáciu závislostí (Python 2.7 je dodávaný v mnoho linuxových operačných systémoch bez nutnosti doinštalovania).

Ako bolo uvedené v 3.3, MPI označuje štandard, nie konkrétnu implementáciu. My sme v našom nástroji využili implementáciu MPICH verzie 3.1.4[2], ktorá odstránila mnoho známych chýb v predchádzajúcich implementáciách a je kompatibilná s mnoho operačnými systémami a verziami jadra vrátane toho, ktorý budeme v našej práci využívať.

Pre prácu s MPI v programovacom jazyku Python sme sa rozhodli využiť balík `mpi4py`[3], ktorý poskytuje jednoduché a intuitívne rozhranie nad MPI. Obsahuje implementácie zasielania správ od procesu k procesu (ang. *point-to-point*) ako aj kolektívne funkcie. Užitočnou výhodou je možnosť zasielania a prijímania dátových typov jazyka Python (napr. `dict`) cez MPI bez potreby serializácie a deserializácie.

[1] https://www.python.org/
[2] https://www.mpich.org/2015/02/23/mpich-3-1-4-released/
[3] https://mpi4py.readthedocs.io

Využitie komunikácie nad MPI medzi rôznymi výpočtovými uzlami je podmienené službou `ssh`[4] bez hesla. Pre tento účel sa využila asymetrická kryptografia, konkrétne dvojica kľúčov typu `RSA`, z ktorej súkromný je na strane klienta a verejný na strane serveru.

Na logovanie v aplikácií využijeme balík `logging`. Ten ponúka možnosť logovania na rôznych úrovniach (`debug`, `info`, `warning`, ...) s výstupom na štandardný výstup (konzola) alebo do súboru.

Inštaláciu spomenutých závislostí sme vyriešili inštalačným skriptom `installC304.sh`, ktorý sa nachádza v repozitári spolu so zdrojovými kódmi riešenia v priečinku `scripts`. Inštalačný skript vykoná nasledovné kroky:

- stiahne, preloží a nainštaluje knižnicu MPICH 3.1.4,
- nainštaluje Python balík `mpi4py`,
- nainštaluje nástroj `p7zip`[5],
- stiahne a rozbalí nástroj hashcat a
- uloží súkromný a verejný kľúč pre službu `ssh`[6].

Inštalačný skript je prispôsobený na operačný systém CentOS[7], ktorý sa nachádza v laboratórií C304, kde budú prebiehať experimenty.

5.2 Rozhranie aplikácie

Nami implementovaný nástroj sa spúšťa spustiteľným súborom `mpiHashcat.py`, ktorý slúži ako vstupný bod celej aplikácie. Nástroj má, ako bolo uvedené v 4.4, rozhranie príkazového riadku odvodené z nástroja hashcat. Doplnené boli niektoré parametre ovplyvňujúce beh distribuovaného výpočtu. Všetky prepínače sú popísané v tabuľke 5.1.

Prepínač	Typ	Popis
-a	num	Typ útoku. Podporované sú maskové útoky (3) a slovníkové útoky (0)
-m	num	Typ hashu. Podporované sú všetky typy, ktoré podporuje nástroj hashcat
--chunk	num	Veľkosť jednej čiastkovej úlohy. Pre maskové útoky počet sekúnd, pre slovníkové počet hesiel zo slovníka
--report	str	Cesta k súboru s reportom útoku

Tabuľka 5.1: Prepínače nástroja mpiHashcat

Pre spustenie útoku s využitím MPI na viacerých uzloch je možné využiť skript `mpirun`[8], ktorý je súčasťou MPICH (a takmer každej inej) implementácie MPI. Alternatívne je možné tiež využiť príkaz `mpiexec`[9], ktorý má rovnaké rozhranie ako `mpirun`, avšak `mpiexec` je súčasťou štandardu MPI. Obe varianty odľahčujú konečného užívateľa od konfigurácie výpočtových uzlov, nastavenia architektúry procesora a pod. Príklad spustenia programu

[4] https://www.ietf.org/rfc/rfc4251.txt
[5] pre budúce rozbalenie archívu s nástrojom hashcat
[6] Súkromný kľúč je súčasťou skriptu, aby bolo možné použiť rovnakú dvojicu RSA kľúčov pre komunikáciu medzi ľubovolnými dvoma uzlami.
[7] https://www.centos.org/
[8] http://qcd.phys.cmu.edu/QCDcluster/mpi/mpirun_mpich.html
[9] https://www.mpich.org/static/docs/v3.1/www1/mpiexec.html

v linuxovom systéme na uzloch `h01`, `h02`, `h03` a `h04` by mohol vyzerať následovne:

```
mpirun --host h01,h02,h03,h04 <program> <argumenty>
```

Uvedomme si, že skript `mpirun` spustí `<program>` na každom z uzlov a preto je nutné, aby bol na každom uzly dostupný (napríklad aby cesta k programu bola umiestnená v systémovej premennej `PATH`). Riešenie môže byť tiež uloženie spustiteľného súboru na rovnakú adresu v každom uzly a v skripte `mpirun` použiť absolútnu cestu.

5.3 Architektúra

Nástroj `mpiHashcat` sa skladá z piatich modulov, ktoré medzi sebou komunikujú. V tejto kapitole krátko popíšeme každý z nich s tým, že implementačné detaily budú uvedené v ďalších sekciách.

mpiHashcat - vstupný bod aplikácie pre server aj výpočtové uzly. V tomto súbore sa definuje cesta k nástroju hashcat ako aj úroveň logovania celej aplikácie. Ďalej sa získa MPI *rank* (identifikátor uzlu) pre každý uzol, aby sa mohol beh aplikácie rozdeliť na server (rank 0) a N výpočtových uzlov (rank 1 až N).

mpiHashcatServer - implementuje metódy serveru, distribúcie úloh, zberu dát z výpočtových uzlov a zasielanie riadiacich správ. Pre každého klienta si ukladá (v triede `mpiHashcatClientStatus`) rýchlosť, čas posledného pridelenia úlohy a pod.

mpiHashcatClient - obsahuje implementácie klienta, teda výpočtového uzlu. Čaká na správy od serveru, ktoré interpretuje, vykoná a zasiela odpoveď späť serveru.

mpiHashcatClientStatus - drží informácie o konkrétnom výpočtovom uzle (čas spustenia, veľkosť poslednej úlohy, nameraná rýchlosť, ...) a implementuje metódy nad nimi (napr. výpočet veľkosti úlohy podľa rýchlosti).

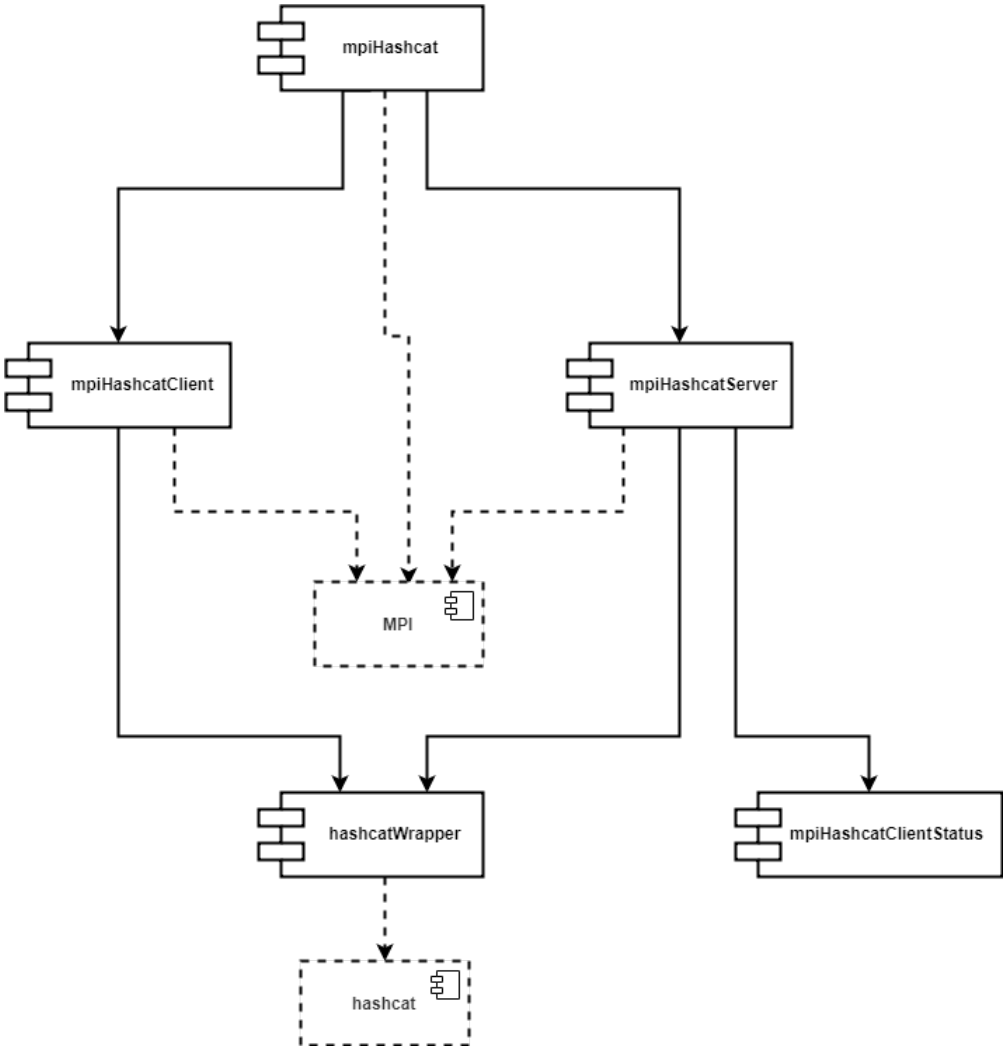
hashcatWrapper - trieda poskytujúca rozhranie nad nástrojom hashcat. Spúšťa útoky, nastavuje parametre útokov, preberá a parsuje výsledky.

Architektúra modulov nástroja mpiHashcat je znázornená na obrázku 5.1, vrátane závislostí na rozhraní MPI a nástroji hashcat.

5.4 Komunikácia a protokol

Komunikácia medzi serverom a výpočtovými uzlami prebieha cez štandardné rozhranie MPI. Využívajú sa funkcie `send` resp. `recv` pre odosielanie resp. prijímanie správy. Komunikácia prebieha výhradne od serveru k jednému klientovi alebo od jedného klienta k serveru.

Vlastný protokol pre prenos riadiacich informácií sa prenáša ako štandardný dátový typ jazyka Python `dict` (slovník). Obsah správy tvoria dvojice kľúč - hodnota (ang. *key - value*), kde "kľúč" je vždy sledovaný parameter a "hodnota" je hodnota daného parametru. Jediný parameter, ktorý nesie každá správa je `msgType`, teda typ správy. Podľa tohto kľúča sa klient resp. server rozhoduje, aké ďalšie parametre má v správe očakávať. Všetky použité riadiace správy sú popísané v nasledujúcom zozname.



Obr. 5.1: Architektúra nástroja mpiHashcat

- **benchmarkRequest** - žiadosť klienta o vykonanie merania rýchlosti nástrojom hashcat pre daný hash typ (`hashType`).

```
{
        "msgType" : "benchmarkRequest",
        "hashType": "num_type"
}
```

- **benchmarkResponse** - odpoveď klienta na `benchmarkRequest`. Obsahuje návratový kód nástroja hashcat (`retCode`) pre overenie správnosti merania a nameranú rýchlosť (`speed`) pre požadovaný hash type (`hashType`).

```
{
        "msgType" : "benchmarkResponse",
        "retCode" : X,
        "hashType": "num_type",
        "speed" : Y
```

```
}
```

- **maskAttackRequest** - žiadosť klienta o vykonanie maskového útoku. Obsahuje hash (`hash`) vrátane typu (`hashType`), masku (`mask`) a parametre čiastkovej úlohy `skip` a `limit`.

```
{
        "msgType" : "maskAttackRequest",
        "hashType": "num_type",
        "hash" : "hash_string",
        "mask" : "mask_string"
        "skip" : "num_skip_parameter",
        "limit" : "num_limit_parameter"
}
```

- **dictAttackRequest** - žiadosť klienta o vykonanie slovníkového útoku. Správa môže byť rozdelená na viac častí a posielaná oddelene. Obsahuje hash (`hash`) vrátane typu (`hashType`), čiastkový slovník uložený v reťazci, celkový počet častí správy (`partFrom`) a poradové číslo správy (`partNum`). Tento typ správy sa využíva iba vo verzii nástroja mpiHashcat s distribuovanou pamäťou.

```
{
        "msgType" : "dictAttackRequest",
        "hashType": "num_type",
        "hash" : "hash_string",
        "partNum" : X,
        "partFrom": Y,
        "dict" : "dict_string"
}
```

- **dictAttackRequest** - žiadosť klienta o vykonanie slovníkového útoku. Obsahuje hash (`hash`) vrátane typu (`hashType`), adresu k slovníku uloženého v zdieľanej pamäti (`dict`) a parametre čiastkovej úlohy `skip` a `limit`. Tento typ správy sa využíva iba vo verzii nástroja mpiHashcat so zdieľanou pamäťou.

```
{
        "msgType" : "dictAttackRequest",
        "hashType": "num_type",
        "hash" : "hash_string",
        "skip" : X,
        "limit" : Y,
        "dict" : "path_to_dict"
}
```

- **attackExhausted** - odpoveď klienta na ľubovoľný typ útoku. Čiastková úloha bola vykonaná a heslo nebolo nájdené. Súčasťou správy je aj návratový kód nástroja hashcat (`retCode`).

```
{
        "msgType" : "attackExhausted",
```

```
        "retCode" : X
}
```

- **attackFound** - odpoveď klienta na ľubovolný typ útoku. Čiastková úloha bola vykonaná a heslo (`passwd`) bolo nájdené. Súčasťou správy je aj návratový kód nástroja hashcat (`retCode`).

```
{
        "msgType" : "attackFound",
        "passwd" : "found_password",
        "retCode" : X
}
```

- **attackError** - odpoveď klienta na ľubovolný typ útoku. Čiastková úloha zlyhala a nástroj hashcat skončil s návratovým kódom (`retCode`).

```
{
        "msgType" : "attackError",
        "retCode" : X
}
```

- **attackAborted** - odpoveď klienta na ľubovolný typ útoku. Čiastková úloha bola prerušená a nástroj hashcat skončil s návratovým kódom (`retCode`).

```
{
        "msgType" : "attackAborted",
        "retCode" : X
}
```

- **appStopRequest** - žiadosť klienta o ukončenie procesu.

```
{
    "msgType" : "appStopRequest"
}
```

- **clientRuntimeRequest** - žiadosť klienta o zaslanie celkového času behu nástroja hashcat.

```
{
    "msgType" : "clientRuntimeRequest"
}
```

- **clientRuntimeResponse** - odpoveď klienta na `clientRuntimeRequest`. Správa obsahuje celkový čas behu nástroja hashcat na klientskom uzle (`time`).

```
{
    "msgType" : "clientRuntimeResponse",
    "time" : X
}
```

5.5 Implementácia priebehu útoku

Pri spustení útoku cez nástroj mpiHashcat na N uzloch sa vždy inicializuje jeden server a N-1 výpočtových uzlov, z ktorých každý má unikátny celočíselný identifikátor (rank - zo štandardu MPI). Pre minimalizáciu použitých počítačov môže byť jeden počítač aj server aj klient (stačí zadať dva krát jeho názov alebo adresu na začiatok parametru `--host` v skripte `mpirun`).

Pre spúšťanie nástroju hashcat využívame metódu `Popen` z modulu `subprocess`[10]. Tá spustí nový proces a vytvorí k nemu komunikačné kanály. Pomocou nich (konkrétne kanálu štandardného výstupu `stdout`) čítame výsledky behu nástroja hashcat, ktoré následne parsujeme a vytvárame správy z protokolu definovaného v 5.4. V triede `hashcatWrapper` sú tiež definované statické parametre nástroja hashcat, ktoré sú použité pri každom volaní. Patria tam:

- `--machine-readable` - pre jednoduchšie strojové spracovanie výstupu,
- `--quiet` - pre potlačenie informatívnych výpisov a
- `--potfile-disable` - pre možnosť opätovného lámania rovnákého hashu[11].

Táto možnosť statických parametrov je tiež vhodná pre nastavenie rôznych optimalizácií útoku (`--workload-profile`, `-O`, a pod.).

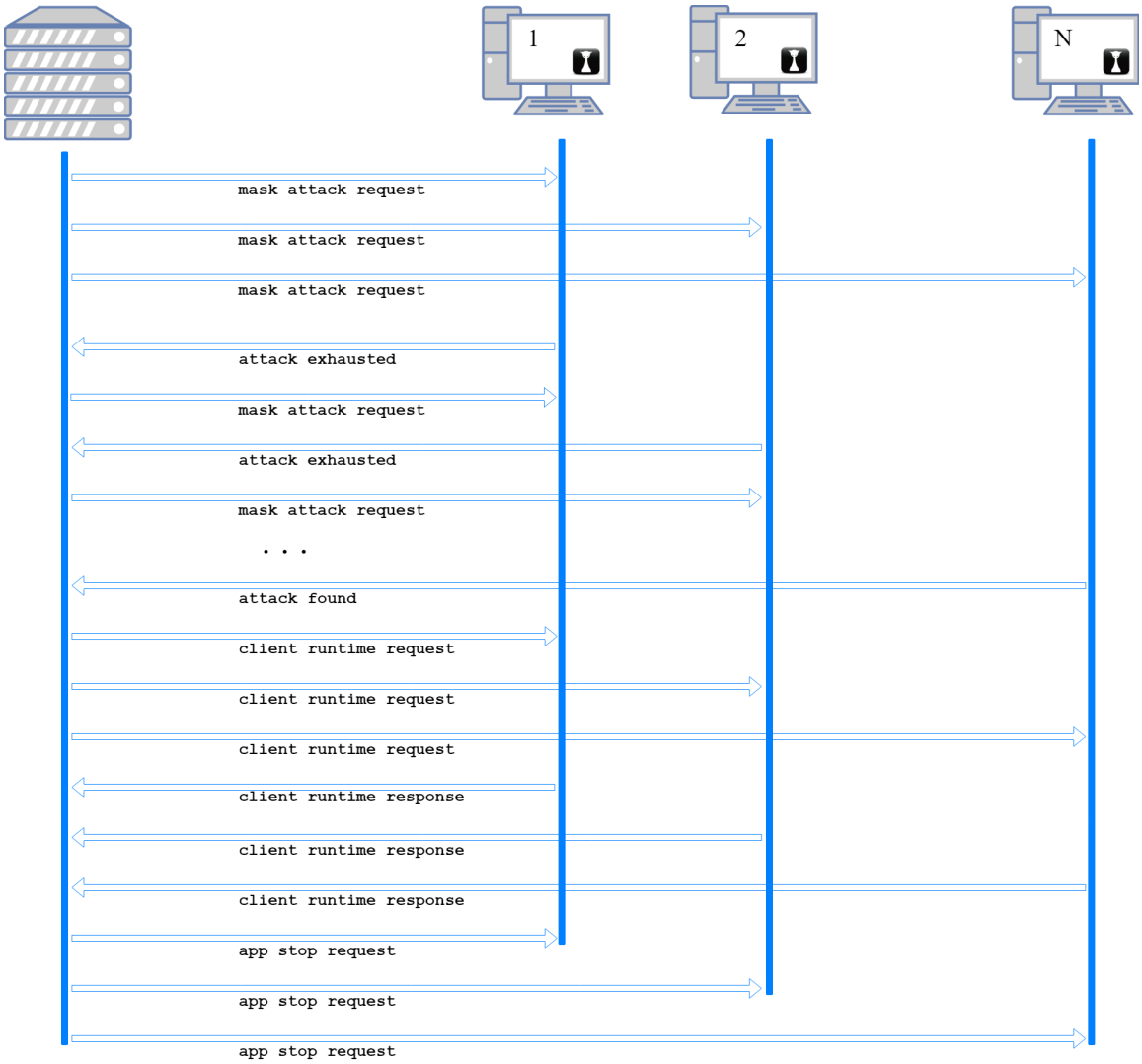
Server pri spustení načíta vstupné argumenty a overí ich správnosť. Následne pomocou nástroja hashcat zistí veľkosť stavového priestoru kľúča (keyspace) a túto informáciu si uloží. Pre každého klienta vytvorí inštanciu triedy `mpiHashcatClientStatus` a nastaví mu prednastavené hodnoty. Odošle všetkým klientom inicializačnú úlohu a po jej dokončení odmeria rýchlosť lámania v počte preskúmaných hesiel za sekundu. Pri vytváraní ďalšej úlohy pre klienta sa vychádza práve z tejto rýchlosti. Ak sme z poslednej úlohy zistili, že klient n má rýchlosť lámania 200 000 hesiel za sekundu a veľkosť čiastkovej úlohy je nastavená na 60 sekúnd, pridelí sa klientovi úloha o veľkosti 12 miliónov hesiel ($60 \times 200\,000$). Tento postup sa bude opakovať každou dokončenou úlohou a dosiahneme tým presnosť v zadávaní dĺžky čiastkových úloh. Pôvodne sme tento cieľ chceli dosiahnuť spustením nástroja hashcat na každom klientovi s parametrom `benchmark`, avšak jeho výsledky sa javili veľmi nepresné a skreslené. Táto funkcionalita je stále implementovaná (viď protokol v kapitole 5.4) vrátane spúšťania benchmarku a zbierania výsledkov na strane serveru. Ďalšia navrhnutá zmena bola zameniť benchmark za parameter `speed-only`, ktorý by mal namiesto reálneho lámania iba zmerať rýchlosť (prepínač využívaný aj nástrojom Hashtopolis), ale ani táto možnosť sa neosvedčila (veľké nepresnosti, hlavne pri slovníkových útokoch) a preto sme sa rozhodli implementovať meranie rýchlosti priamo v nástroji.

Po počiatočnej úlohe server prechádza do hlavného cyklu, v ktorom čaká na odpoveď od akéhokoľvek klienta a spracuje ju. Spracovanie sa líši podľa typu správy. Najčastejšie sa vyskytuje odpoveď `attackExhausted` a server pri takejto správe vygeneruje novú čiastkovú úlohu a odošle ju klientovi. V prípade ak príde odpoveď od klienta `attackFound`, server počká na dokončenie každého z klientov a beh ukončí. Pred ukončením si vyžiada od každého klienta dobu strávenú lámaním, sčíta ich pre všetkých klientov, zaznamená v súbore pre report a následne pošle požiadavku na ukončenie aplikácie. Aplikácia sa tiež ukončí, ak došlo k vyčerpaniu stavového priestoru hesla a teda všetky možnosti boli preskúmané s negatívnym výsledkom.

[10] https://docs.python.org/2.7/library/subprocess.html
[11] https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#what_is_a_potfile

Klientsky (výpočtový) uzol po počiatočnej inicializácií čaká v cykle na správy od serveru (teda od uzlu, ktorý má *rank* 0) a vykonáva ich. Aby bolo možné počítať čas behu nástroja hashcat na každom klientskom uzle individuálne, každý uzol meria celkový čas behu nástroja hashcat. Pred spustením prijatej úlohy sa spustí časovač, po jej dokončení zastaví a časový rozdiel sa pripočíta k celkovému času strávenému lámaním.

Komunikácia medzi serverom a N uzlami pri maskovom útoku s pozitívnym výsledkom (heslo bolo nájdené) je znázornená na obrázku 5.2.



Obr. 5.2: Ukážka komunikácie pri nájdenom hesle

Situácia je podobná v prípade slovníkového útoku. Vtedy sa namiesto veľkosti úlohy v sekundách berie veľkosť úlohy v počte hesiel. Server si pamätá posledný prístup do súboru so slovníkom a pomocou nástroja `islice` z balíka `itertools`[12] číta súboru požadovanej veľkosti, prevádza ju na reťazec a posiela ako súčasť žiadosti o slovníkový útok. Klientsky uzol túto správu ukladá do dočasného súboru a používa ho ako slovník čiastkovej úlohy. Vďaka tomu nie je potrebné používať parametre `skip` a `limit` ale jednoducho k čiastkovému

[12] https://docs.python.org/2.7/library/itertools.html

útoku použiť celý vytvorený slovník. V prípade, ak je časť slovníka príliš veľká na načítanie a odoslanie v jednej správe klientovi, sa správa fragmentuje na viac častí, z ktorých každá je odosielaná samostatne a klient ich postupne prijíma a ukladá do dočasného súboru. Tento prístup sme zvolili z dôvodu, že pri väčších čiastkových slovníkoch bol server príliš náročný na operačnú pamäť. Proces čítania a odosielania slovníka sme navyše ešte optimalizovali manuálnym uvoľňovaním pamäte príkazom `collect` zo štandardného modulu `gc`[13] (skratka z anglického *garbage collector*). Veľkosť samostatne prenášaného bloku je možné nastaviť v premennej `C_TRANSFER_LIMIT_PASSWORDS` v súbore `mpiHashcatServer.py`.

Ak je zadaný (nepovinný) parameter `--report cesta_k_suboru`, na konci úspešného aj neúspešného testu sa zapíše nový záznam do súboru vo formáte:

$$m; a; h; md; n; rt; tt$$

, kde:

- `m` je typ lámaného hashu,
- `a` je typ použitého útoku,
- `h` je hash hladaného hesla,
- `md` je maska v prípade maskového útoku a cesta k slovníku v prípade slovníkového útoku,
- `n` je počet výpočtových uzlov (server sa nepočíta),
- `rt` je reálny čas potrebný na útok a
- `tt` je celkový čas behu lámacieho nástroja na všetkých uzloch.

5.6 Rozšírenie nástroja o zdieľanú pamäť

Ako bolo uvedené v predchádzajúcej kapitole, pri slovníkových útokoch boli slovníky prenášané rozhraním MPI, čo si vyžadovalo réžiu na ich načítanie serverom, uloženie do dátového typu jazyka Python a odoslanie klientovi. Ten si slovník po prijatí uložil do dočasného súboru a cestu k nemu použil pri spúšťaní nástroja hashcat.

V tejto kapitole popíšeme implementáciu modifikácie nástroja mpiHashcat, ktorá pre prácu so slovníkmi využíva zdieľané úložisko. Ďalšou výhodou tohto riešenia je, že v zdieľanom úložisku môžu byť uložené aj používané nástroje a zdrojové súbory nástroja mpiHashcat. V prípade modifikácie je postačujúce zmeniť súbory iba v tomto úložisku a zmeny sa premietnu v každom uzle.

5.6.1 Konfigurácia uzlov

Nástroj mpiHashcat so zdieľanou pamäťou je implementovaný rovnako ako verzia s distribuovanou pamäťou v podpriečinku `mpi_shared`. Pre správny chod nástroja je potrebná dodatočná konfigurácia, konkrétne vytvorenie zdieľaného úložiska na servery. Pre tento účel sme využili službu `NFS` (skratka od anglického *Network File System*).

Na strane serveru sme pre zjednodušenie inštalácie zdieľaného priečinku vytvorili skript `installNfsServer.sh`, ktorý v prostredí laboratória C304 nainštaluje, nakonfiguruje a spustí službu NFS. Skript obsahuje príkazy:

[13] https://docs.python.org/2/library/gc.html

- `yum -y install nfs-utils` - pre inštalácu balíka `nfs-utils`,
- `systemctl enable nfs-server.service` - pre automatické spúšťanie NFS serveru po štarte,
- `systemctl start nfs-server.service` - pre okamžité spustenie NFS serveru,
- `echo '/opt/nfs *(rw,sync,no_subtree_check)' » /etc/exports` - pre sprístupnenie priečinku `/opt/nfs` klientom z akejkoľvek IP adresy a
- `exportfs -a` - pre exportovanie všetkých priečinkov zo súboru `exports`.

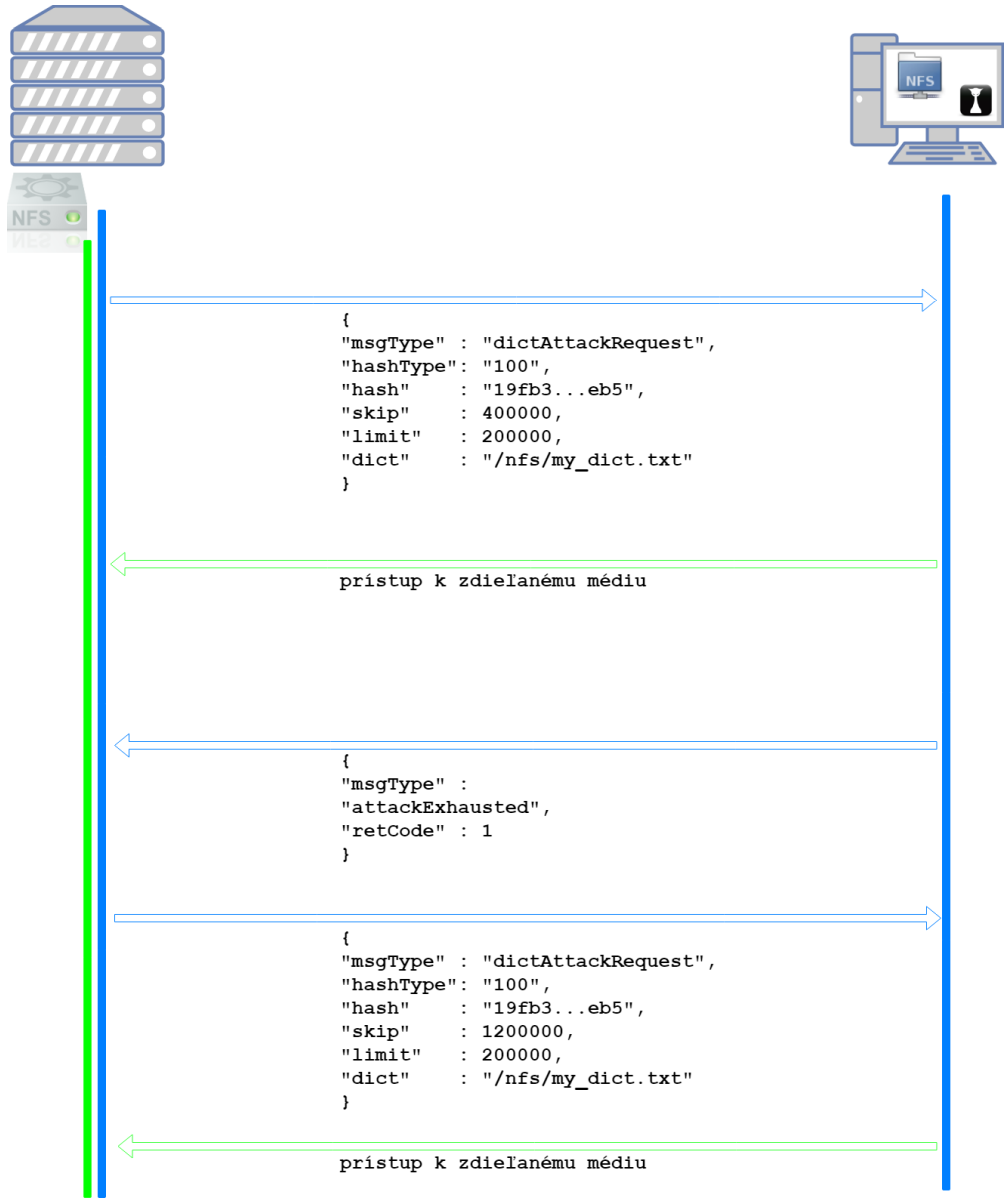
Na strane klientov už stačí príkazom `mount h17:/opt/nfs /opt/nfs` primontovať disk, kde `h17` je adresa serveru s NFS službou.

5.6.2 Rozdiely v implementácií

Nástroj mpiHashcat so zdieľanou pamäťou je implementačne veľmi podobný tomu s distribuovanou pamäťou. Pri maskovom útoku prebieha komunikácia medzi serverom a uzlami rovnako, rozdiel môže byť len v absolútnej ceste k nástroju hashcat. V prípade zdieľanej pamäte môžeme používať službu NFS k získaniu nástroja bez potreby inštalácie na každý výpočtový uzol.

Hlavná zmena nastáva pri slovníkovom útoku. Namiesto časti slovníku, ktorý je prenesený rozhraním MPI a lokálne uložený do súboru sa v tomto prípade prenáša iba cesta k slovníku s parametrami `skip` a `limit`, ktoré udávajú časť k preskúmaniu v rámci distribuovaného útoku. Slovník je uložený v zdieľanom úložisku, prístupný službou NFS a klienti k nemu pristupujú, akoby sa nachádzal na lokálnom disku.

Na obrázku 5.3 je znázornená komunikácia medzi serverom so službou NFS a jedným klientom, ktorému sú prideľované čiastkové úlohy v rámci distribuovaného maskového útoku na hash typu SHA-1.
Ako je z komunikácie zrejmé, využíva sa slovník `my_dict.txt` uložený v zdieľanom úložisku na adrese `/nfs/`.



Obr. 5.3: Ukážka komunikácie nástroja mpiHashcat so zdieľanou pamäťou

Kapitola 6

Experimenty

V tejto kapitola sa budeme venovať experimentovaniu s vybranými nástrojmi daných technológií za rovnakých podmienok. Popíšeme, aké testovacie sady sme zvolili pre experimentovanie a akým spôsobom budeme potrebné dáta získavať. Opísané budú aj implementované pomocné skripty na automatizáciu testovania a zberu dát.

6.1 Testovacie scenáre

Testovacie scenáre boli navrhnuté tak, aby bolo v rozumnom čase možné namerať čo najviac parametrov v rôznych scenároch útoku. Pre každú variantu sme merali ako celkový reálny čas, tak aj súčet časov všetkých uzlov, ktorý trávili lámaním. Pre maskový útok sme využili parametre z tabuľky 6.1.

Parameter	**Hodnoty**
Dĺžka hľadaného hesla (v počte malých písmen)	7, 8, 9
Počet uzlov	16, 8, 4, 2
Hľadaný hash typ	MD5, SHA-1, Whirlpool
Veľkosť čiastkovej úlohy v sekundách	60, 120, 600, 1800

Tabuľka 6.1: Parametre maskového útoku

Testovacie parametre by mali odzrkadľovať správanie technológie ako pri krátkych útokoch, tak aj pri útokoch dlhších. Útok môže trvať krátku dobu v prípade malého stavového priestoru kľúča ale aj v prípade väčšieho stavového priestoru s jednoduchým typom hashu (je jednoduchšie a menej strojovo náročné vypočítať jeho hash hodnotu). V prvom prípade môžeme očakávať menej riadiacich správ ako v druhom prípade, za rovnaký čas. V testovacích scenároch sa nachádzajú všetky kombinácie všetkých parametrov okrem typu `Whirlpool`. Ten sme zvolili ako reprezentanta veľmi náročného hashu, avšak po meraní prvej technológie sme sa rozhodli tento typ merať iba pre 16 uzlov. Pri 16-tich uzloch hesla dĺžky 9 znakov trval útok zhruba 1,6 hodiny. Predpokladáme, že pre 2 uzle by útok trval najmenej 12,8 hodiny. Potrebné je namerať takých útokov pre každú technológiu 4 (pre 4 rôzne dĺžky čiastkových úloh), čím sa dostávame za hranicu 10 celých dní iba pre jeden typ hashu, čo je pri využívaní laboratória C304 na experimenty neprípustné. Pre slovníkový útok sú parametre uvedené v tabuľke 6.2. Počty uzlov sú zachované z maskového útoku, rovnako aj typy hashu.

Parameter	Hodnoty
Počet uzlov	16, 8, 4, 2
Hľadaný hash typ	MD5, SHA-1, Whirlpool
Veľkosť čiastkovej úlohy v počte hesiel	500k, 1m, 3m, 6m
Použitý slovník	rockyou.txt, 6xrockyou.txt, 24xrockyou.txt

Tabuľka 6.2: Parametre slovníkového útoku

Veľkosť čiastkovej úlohy sme sa rozhodli implementovať v počte hesiel, nakoľko sme pri pôvodnej implementácií narazili na niekoľko problémov. Hlavný problém bol, že rýchlosť lámania pri slovníkových útokoch sa pohyboval vo vysokých číslach (rádovo milióny hesiel za sekundu) a pri rovnakom algoritme prideľovania úloh sa obsadzoval väčšinou jeden, alebo malá podmnožina uzlov. Pôvodný algoritmus spočíval v adaptívnom vypočítavaní veľkosti úlohy tak, že z poslednej úlohy sa vypočíta rýchlosť lámania za jednu sekundu a táto rýchlosť sa vynásobí počtom požadovanej úlohy. Výsledný počet hesiel čiastkovej úlohy (pri požadovanej dĺžke 600 alebo 1800 sekúnd) bol často väčší ako celkový počet hesiel v slovníku, preto sme sa rozhodli veľkosti pevne definovať počtom hesiel.

Veľkosti a počet hesiel v použitých slovníkoch sú uvedené v tabuľke 6.3. Vychádzali sme zo známeho slovníku `rockyou.txt`[1], ktorý sme konkatenovali za seba 6 krát (pri `6xrockyou.txt`), resp. 24 krát (pri `24xrockyou.txt`).

Slovník	Veľkosť	Počet hesiel
rockyou.txt	134 MB	14 344 391
6xrockyou.txt	801 MB	86 066 346
24xrockyou.txt	3,2 GB	344 265 384

Tabuľka 6.3: Použité slovníky

6.2 Automatizácia

V predchádzajúcej kapitole sme sa venovali testovacím scenárom, ktorých je približne 300 pre každú technológiu. Spúšťať a zaznamenávať všetky sledované parametre takémuto počtu testov by bolo časovo veľmi náročné a v niektorých prípadoch (napr. meranie celkového času pri nástroji Hashtopolis) aj nepresné. Z toho dôvodu sme sa snažili maximalizovať automatizáciu pre každý nástroj. V tejto kapitole budú popísané spôsoby automatizácie pre jednotlivé technológie vrátane pomocných skriptov, ktoré sú súčasťou repozitára s riešením.

Cieľom automatizácie, okrem zjednodušenia spúšťania, bolo čo najviac zjednotiť výstupy, aby následné spracovanie a porovnávanie bolo jednoduchšie a bez potreby modifikácie prípadne dopočítavania údajov. Rozhodli sme sa využiť formát CSV[2] so štruktúrou reportu z nástroja mpiHashcat (popísaná v 5.5).

V spúšťaných testoch sme zvolili taký hľadaný hash, ktorý sa nenachádza v príslušnej maske ani slovníku. Je to z dôvodu, aby sme dosiahli vyčerpanie celého stavového priestoru kľúča. V opačnom prípade by mohla nastať situácia, že rôzne technológie iným spôsobom priradia uzlom čiastkové úlohy. Minimalizujeme tak vplyv samotného nástroja (a jeho implementácie) na technológiu.

[1] https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt
[2] Čiarkou oddelené hodnoty, z anglického *Comma-Separated Values*

6.2.1 mpiHashcat

Pre hromadné spúšťanie maskových útokov nástrojom mpiHashcat (s distribuovanou pamäťou) sme vytvorili skript `distributedMaskBulkTests.sh` v unixovom jazyku `bash`[3]. V skripte sú definované absolútne cesty k spustiteľným súborom (`mpiHashcat.py`), k výstupom a k priečinku, kam sa uloží log ku každému spustenému testu. Pre slovníkové útoky je implementovaný skript `distributedDictBulkTests.sh` s rovnakou funkcionalitou.

Telo skriptu sa skladá zo štyroch zanorených cyklov tak, aby boli spustené všetky kombinácie. Iteruje sa cez všetky typy hashu, veľkosti čiastkových úloh, počty uzlov a dĺžky hľadaných hesiel. Výsledok testu aplikácia zapíše do súboru predaného s parametrom `--report` v očakávanom formáte CSV.

Pre sledovanie vyťaženia a teploty grafického čipu sme implementovali jednoduchý skript `gpuUtilizationMeasurement.sh`, ktorý prijíma ako jediný parameter cestu k výstupnému súboru. Do tohto súboru zaznamenáva každú sekundu aktuálny čas, teplotu grafického čipu, vyťaženie grafického procesoru a spotrebovanú pamäť na grafickom čipe. Tento skript je spúšťaný tesne pred spustením testu a ukončený tesne po ukončení testu. Získavame tak zaujímavé dáta o vyťažení GPU v čase.

6.2.2 Fitcrack

Nástroj Fitcrack poskytuje na strane serveru rozhranie *REST API*[4], ktorým je možné plnohodnotné ovládanie aplikácie, vrátane prihlasovania, vytváranie úloh a pod. Časť automatizácie bola vytvorená v spolupráci s Michalom Eisnerom v rámci jeho bakalárskej práce (konkrétne použité skripty `add_job_fitcrack.py` a `start_job_fitcrack.py`). Dávkové spúšťanie úloh bolo implementované v skriptoch:

- `fitcrack_bulk_tests_mask_md5_sha1.sh` - pre spustenie všetkých maskových útokov na hash typu MD5 a SHA1,
- `fitcrack_bulk_tests_mask_whirlpool.sh` - pre maskové útoky na 16 uzloch na hash typu Whirlpool,
- `fitcrack_bulk_tests_dict_md5_sha1.sh` - pre spustenie všetkých slovníkových útokov na hash typu MD5 a SHA a
- `fitcrack_bulk_tests_dict_whirlpool.sh` - pre spustenie slovníkových útokov na 16 uzloch na hash typu Whirlpool.

Rozdelenie vyplýva z potreby oddeliť typ Whirlpool, ktorý sme sa rozhodli testovať iba na 16-tich uzloch (viď 6.1).

Každý zo skriptov je rozdelený na dve časti. V prvej časti, podobne ako pri MPI, vo vnorených cykloch pridá cez Fitcrack API na server všetky požadované kombinácie útokov a následne ich volaním prebratého skriptu `fitcrack_start_job.py` spúšťa jeden po druhom.

Na zber výsledkov sme vytvorili skript `print_report_mask.py` resp. `print_report_dict.py`, ktorý cez API získa výsledky maskových resp. slovníkových útokov a v požadovanom formáte CSV ich vypíše na štandardný výstup.

[3] https://www.gnu.org/software/bash/

[4] REST API je aplikačné rozhranie využívajúce HTTP pre výmenu dát. Viac na https://searchmicroservices.techtarget.com/definition/RESTful-API

6.2.3 Hashtopolis

Nástroj Hashtopolis rozhranie API neposkytuje, takže jediným spôsobom automatizácie bol priamy prístup do jeho databázy. Prvý krok bolo sledovanie závislostí medzi tabuľkami hlavne z pohľadu vytvorenia maskového resp. slovníkového útoku a jeho spustenie. Vytvorené boli nasledovné skripty:

- `hashtopolis_add_dict_job.py` (pristupuje k databázovým tabuľkám `File`, `FileTask`, `Task`, `TaskWrapper`, `Agent`, `Assignment`) - vytvorí slovníkový útok podľa argumentov príkazového riadku, pridelí ho požadovanému počtu klientom a útok spustí.
- `hashtopolis_add_mask_job.py` (pristupuje k tabuľkám `Task, TaskWrapper, Agent, Assignment`) - rovnako ako predchádzajúci, vytvorí maskový útok a priradí ho klientom.
- `hashtopolis_get_cracking_time.py` (pristupuje k tabuľke `Chunk`) - podľa zadaného identifikátora (ID) úlohy sčíta časy lámania všetkých čiastkových úloh a výsledok vypíše na štandardný výstup.
- `hashtopolis_is_task_running.py` (pristupuje k tabuľkám `Chunk a Task`) - podľa priradených čiastkových úloh overí, či je úloha ukončená, alebo ešte prebieha.

Kombináciou týchto skriptov sme dosiahli rovnaké dávkové spúšťanie ako pri predchádzajúcich dvoch nástrojoch s rovnakým rozdelením, `hashtopolis_bulk_mask_md5sha1.sh` a `hashtopolis_bulk_mask_whirlpool.sh` pre automatizované spúšťanie maskových útokov a `hashtopolis_bulk_dict_md5sha1.sh` a `hashtopolis_bulk_dict_whirlpool.sh` pre slovníkové útoky.

Prvý z rozdielov oproti nástroju Fitcrack je meranie celkového reálneho času na útok. V nástroji Hashtopolis nie sú do celkového času započítané niektoré kroky útoku, ako napríklad prenos slovníka alebo beh úvodného benchmarku. Aby sme získali časový údaj porovnateľný s ostatnými technológiami, rozhodli sme sa merať celkový čas priamo v skripte.

Ďalším rozdielom je priraďovanie uzlov k úlohe. Kým v nástroji ficrack je možné jeden uzol priradiť viacerých úlohám a postupne ich spúšťať, v nástroji Hashtopolis môže byť uzol v rovnakom čase priradený maximálne jednej úlohe. Preto je nutné pri dávkovom spúšťaní vytvárať ďalšiu úlohu až po dokončení predchádzajúcej.

Súčasťou práce dávkového skriptu je aj zber dát z merania a jeho následné ukladanie v požadovanom formáte CSV. Výsledná štruktúra je rovnaká ako pri ostatných technológiách.

6.2.4 VirtualCL

Pri konfigurácií technológie VirtualCL sme narazili na niekoľko prekážok, z ktorých niektoré sa dali odstrániť a niektoré nie, preto sme VirtualCL vylúčili z technológií na experimenty. Využili sme postup z oficiálnej stránky nástroja hashcat[5] aj napriek tomu, že túto technológiu už nepodporujú a upozorňujú na fakt, že pravdepodobne to fungovať nebude[6]. Využiť sme museli novšiu verziu (1.25) VirtualCL, pretože odkaz na stiahnutie odporúčanej (1.22) bol neaktívny. Žiaľ, v novšej verzií nastali zmeny v štruktúre, kvôli ktorým postup inštalácie odporúčaný nástrojom hashcat nešiel vykonať. VirtualCL sme následne nainštalovali podľa odporúčaného návodu z oficiálnej stránky VirtualCL, avšak zariadenia v sieti

[5] https://hashcat.net/wiki/doku.php?id=vcl_cluster_howto_original
[6] https://hashcat.net/wiki/doku.php?id=vcl_cluster_howto

neboli viditeľné. Dôvodom boli nepodporované zariadenia NVidia. Po zostavení počítača s AMD grafickou kartou sa nám VirtualCL podarilo spojazdniť (OpenCL zariadenie bolo v sieti viditeľné), avšak nástroj hashcat ho nevedel využiť. Staršia verzia nástroja hashcat (`oclHashcat-plus-0.15`), ktorá bola použitá v demonštračných príkladoch na oficiálnej stránke už dostupná nie je a stiahnutá verzia z neoverených zdrojov ani pri veľkej snahe nefungovala.

6.3 Výsledky merania

V tejto kapitole budú predstavené výsledky meraní jednotlivých technológií. Každý z nich bol meraný automatizovane skriptami z kapitoly 6.2 a prezentované výsledky sú spracované z výstupných CSV súborov. Okrem priamo nameraných ukazovateľov sme dopočítavali celkovú réžiu pre všetky uzle, réžiu na jeden uzol, efektivitu a rýchlosť lámania všetkých uzlov. Vzhľadom na rozsah dopočítavaných údajov budeme v tejto kapitole uvádzať iba vybranú časť výsledkov. V tejto kapitole budeme používať nasledovné pojmy:

- **reálny čas** - skutočný čas potrebný na útok bez ohľadu na distribúciu,
- **strojový čas** - súčet časov klientov strávených lámaním,
- **efektivita** - pomer strojového času tráveného lámaním (podľa vzorca z 4.1.3),
- **réžia** - pomer strojového času tráveného inými aktivitami (napríklad prenosom slovníka alebo komunikáciou) ako lámaním (doplnok k efektivite),
- **celková rýchlosť** - spoločná rýchlosť lámania všetkých uzlov spolu (počítame ako stavový priestor hesla podelený reálnym časom) a
- **rýchlosť na 1 uzol** - priemerná rýchlosť lámania jedného uzlu (počítame ako celková rýchlosť podelená počtom uzlov).

Úplné výsledky pred spracovaním v tabuľkovej forme môže čítateľ nájsť v prílohe B tejto práce.

6.3.1 mpiHashcat

Technológia MPI, použitá v nástroji mpiHashcat, pôsobila flexibilne s rýchlou odozvou. Aj relatívne krátke maskové útoky boli efektívne distribuované.

Maskový útok

Nástroj mpiHashcat je implementovaný vo verzií so zdieľanou pamäťou a distribuovanou pamäťou. Keďže na maskový útok tento rozdiel nemá vplyv (pri maskovom útoku sa používajú iba riadiace správy, ktoré sú v oboch verziách zasielané cez rozhranie MPI), merania sme vykonali iba pre distribuovanú verziu.

V tabuľke 6.4 sú uvedené niektoré významné ukazovatele z maskového útoku. Lámanie hesla o dĺžke 7 znakov je príliš krátka úloha pre distribúciu na veľkom počte uzlov. Spomedzi meraných veľkostí čiastkových úloh bola najideálnejšia 120 sekundová úloha. Rozdiel v použití 2 uzlov a 16 uzlov bol iba niečo cez 1 sekundu. Z priemernej rýchlosti lámania jedného uzlu vidíme, že rýchlosť dramaticky klesla a efektivita sa pri 16 uzloch zastavila na slabých 36,3%.

Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť na 1 uzol [P/s]
7	2	120	11,74	19,90	84,70%	15,30%	341 935 397
7	4	120	11,70	26,44	56,51%	43,49%	171 665 595
7	8	120	11,59	38,42	41,45%	58,55%	86 636 158
7	16	120	**10,59**	61,51	36,30%	63,70%	**47 401 777**
8	2	60	88,51	162,42	91,75%	8,25%	1 179 711 842
8	4	60	78,50	178,33	56,80%	43,20%	665 085 760
8	8	60	35,97	207,20	72,01%	27,99%	725 714 375
8	16	60	**35,16**	257,62	45,80%	54,20%	371 232 191
8	2	120	83,61	159,17	95,18%	4,82%	1 248 777 411
8	4	120	81,04	169,51	52,29%	47,71%	644 218 870
8	8	120	81,01	194,90	30,07%	69,93%	322 206 107
8	16	120	**80,01**	248,38	19,40%	80,60%	163 123 631
9	2	60	1 906,66	3804,46	99,77%	0,23%	1 423 828 157
9	4	60	975,73	3813,44	97,71%	2,29%	1 391 145 663
9	8	60	502,38	3851,14	95,82%	4,18%	1 350 956 899
9	16	60	**266,65**	3915,40	91,77%	8,23%	1 272 613 287
9	2	1800	1 795,33	3582,50	**99,77%**	0,23%	1 512 122 370
9	4	1800	1 744,69	3591,26	51,46%	48,54%	778 005 495
9	8	1800	1 774,51	3632,59	25,59%	74,41%	382 465 080
9	16	1800	1 797,53	3684,18	12,81%	87,19%	188 783 977

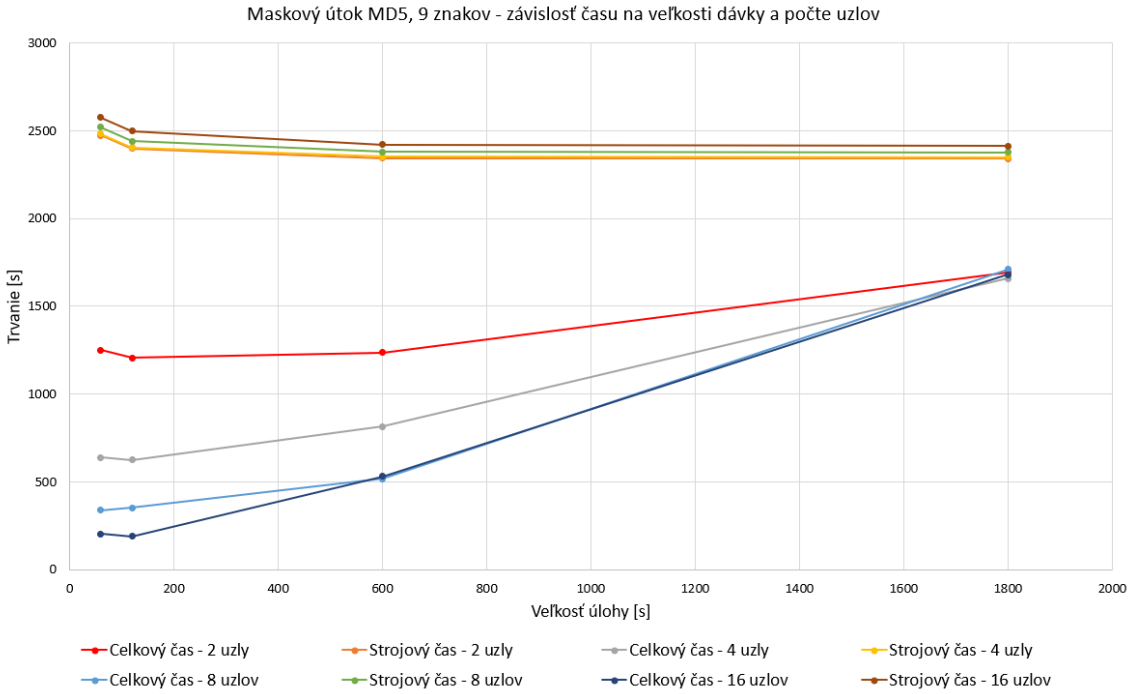
Tabuľka 6.4: Vybrané výsledky maskového útoku nástrojom mpiHashcat na hash SHA-1

Pri hesle dĺžky 8 znakov je síce najvyššia efektivita aj výkonnosť 2 uzlov pri 120 sekundových úlohách, avšak pri 16 uzloch bola úloha dokončená za menej ako polovicu pri 60 sekundových úlohách. Dôvodom je čakanie na jediný uzol, ktorému bola pridelená väčšia úloha v snahe priblížiť sa k 120 sekundám.

Situácia je podobná aj pri hesle dĺžky 9 znakov, čo je relatívne dlhá úloha. Pri 2 uzloch sa veľkosť úlohy 1800 sekúnd javí ako rozumná, pretože obom uzlom boli pridelené veľké úlohy a väčšinu času lámali oba uzly bez čakania. Problém nastáva v prípade ak veľkosť úlohy nie je dostatočná na to, aby vyťažila všetky dostupné uzly. Túto situáciu môžeme vidieť v poslednej časti tabuľky, kedy pri úlohe dĺžky 1800 sekúnd prakticky nezrýchľujeme výpočet pridávaním uzlov a rýchlosť lámania na 1 uzol nám radikálne klesá. Priemernú rýchlosť aj efektivitu nám značne znižujú uzly čakajúce na tie dva uzly, ktoré heslo reálne lámu.

Namerané hodnoty pre hash typu MD5 mali podobné vlastnosti ako SHA-1. Na obrázku 6.1 sú v grafe zaznamenané výsledky pre lámanie 9 znakového hesla.
Spodné 4 línie vyjadrujú reálny čas a horné 4 línie strojový čas (súčet všetkých klientov). Každá línia má záznam pre 4 rôzne veľkosti úloh. Ako je možné vidieť z grafu, pri 60 sekundových úlohách sa zvyšovaním počtu uzlov takmer lineárne skracuje aj čas potrebný na lámanie. Naopak, pri 1800 sekundových úlohách skončia takmer v rovnakom čase 2, 4, 8 aj 16 uzlov. Je to dôsledok rovnakého javu ako pri SHA-1.

Strojový čas je pri rôznom počte uzlov veľmi podobný s jemným navýšením pri vyšších počtoch uzlov (z dôvodu behu prvotnej "benchmark" úlohy na každom uzle).



Obr. 6.1: Maskový útok - graf lámania 9 znakového (MD5) hesla nástrojom mpiHashcat

Súčasťou testov je aj útok na hash typu Whirlpool (viď tabuľku 6.5), ktorý je časovo násobne zložitejší ako MD5 alebo SHA-1. Pri dĺžke hesla 9 znakov trvalo lámanie viac ako 1,5 hodiny a efektivita sa hýbala vo vysokých číslach. Znova, najhorší čas mala 1800 sekundová úloha z dôvodu čakania na najpomalší uzol. Najlepšie výsledky sme získali zo 120 sekundovej úlohy. ktorá mala réžiu iba na úrovni 0,30%. Podobne dobre vyšla aj 60 sekundová úloha, kde prírastok 0,17% môžeme pripísať dvojnásobnému počtu vymenených správ a réžií na spúšťanie nástroju hashcat.

Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť na 1 uzol [P/s]
9	16	60	5791,82	92229,85	99,53%	0,47%	58 590 261
9	16	120	5542,36	88414,68	**99,70%**	0,30%	61 227 311
9	16	600	5607,74	86319,83	96,21%	3,79%	60 513 486
9	16	1800	6061,91	86031,75	88,70%	11,30%	55 979 743

Tabuľka 6.5: Vybrané výsledky maskového útoku nástrojom mpiHashcat na hash Whirlpool

Slovníkový útok

Slovníkový útok bol meraný vo verzií s distribuovanou aj zdieľanou pamäťou. V tabuľke 6.6 sú uvedené vybrané výsledky lámania hash typu MD5 **distribuovanou verziou**. Z prvej časti tabuľky je zjavné, že pridávanie uzlov pri slovníkovom útoku má zmysel iba do určitého bodu. Rozdieľ medzi 8 a 16 uzlami je už záporný, teda rýchlosť lámania sa nezvyšuje, no naopak začína klesať.

Pri slovníkových útokoch má silný vplyv réžia na prenos čiastkového slovníka, jeho uloženie do súboru a načítanie nástrojom hashcat. Taktiež musíme podotknúť, že pri lámaní jednoduchších typov hashu je pre dosiahnutie dobrých výsledkov potrebná väčšia dávka hesiel v slovníku. Túto situáciu sme zaznamenali v druhej časti tabuľky 6.6. Pri dávke 500 tisíc hesiel je efektivita na úrovni 82,56% a rýchlosť lámania všetkých uzlov 279 861 hesiel za sekundu avšak pri dávke 6 miliónov hesiel je efektivita iba 39,14%, ale rýchlosť lámania je viac ako 3 krát väčšia a teda aj potrebný čas na celý útok je viac ako 3 krát nižší.

Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť celková [P/s]
rockyou.txt	2	500000	55,20	90,17	81,68%	18,32%	259 876
rockyou.txt	4	500000	30,82	93,75	76,05%	23,95%	465 460
rockyou.txt	8	500000	**19,23**	102,02	66,30%	33,70%	745 812
rockyou.txt	16	500000	**19,25**	131,83	42,79%	57,21%	745 014
rockyou.txt	2	6000000	21,46	17,57	40,93%	59,07%	668 302
rockyou.txt	4	6000000	20,76	23,37	28,14%	71,86%	690 877
rockyou.txt	8	6000000	19,58	35,05	22,37%	77,63%	732 521
rockyou.txt	16	6000000	20,28	57,95	17,86%	82,14%	707 435
6xrockyou.txt	2	500000	307,53	508,34	**82,65%**	17,35%	**279 861**
6xrockyou.txt	2	1000000	190,06	266,04	69,99%	30,01%	452 847
6xrockyou.txt	2	3000000	135,00	129,68	48,03%	51,97%	637 550
6xrockyou.txt	2	6000000	90,07	70,50	**39,14%**	60,86%	**955 499**
24xrockyou.txt	2	6000000	348,40	258,31	37,07%	62,93%	988 136
24xrockyou.txt	4	6000000	344,41	257,80	18,71%	81,29%	999 593
24xrockyou.txt	8	6000000	350,49	266,82	9,52%	90,48%	982 244
24xrockyou.txt	16	6000000	346,78	285,20	**5,14%**	94,86%	992 736

Tabuľka 6.6: Slovníkový útok na MD5 - mpiHashcat s distribuovanou pamäťou

Lámanie hash typu SHA-1 sme znázornili v grafe 6.2. Pri vyšších čiastkových úlohách sa súčet strojových časov približuje k reálnemu času, čo je rovnako dôsledkom časovo náročného posielania slovníku. Pri veľkosti úlohy 6 miliónov hesiel trvá réžia posielania slovníku viac, ako samotné lámanie. Úzkym hrdlom je server a preto pridávaním uzlov zvýšenie výkonnosti nedosiahneme.
Pre náročný hash typu Whirlpool bola situácia o trocha lepšia. Pri slovníku `24xrockyou.txt` sme dosiahli maximálnu rýchlosť 1 195 270 hesiel za sekundu, čo je pri tomto type hashu v pomere k 16-tim uzlom prijateľné. Celkový čas potrebný na lámanie je pri slovníku `rockyou.txt` necelých 18 sekúnd a pri slovníku `24xrockyou.txt` 288 sekúnd.

Alternatíva so zdieľanou pamäťou, ktorá využíva službu NFS, bola použitá iba pre merania slovníkového útoku. I keď sme pri implementácií očakávali od nej jednoznačne lepšie výsledky, opak bol pravdou. Príklad porovnania so zdieľanou verziou je v tabuľke 6.7.

Takmer v každom meraní (s výnimkou malých slovníkov na malý počet uzlov) slovníkového útoku bola zdieľaná verzia pomalšia. Paradoxne priemerná efektivita je výrazne vyššia pre zdieľanú verziu, napríklad pre hash typu MD5 je priemerná efektivita zdieľanej verzie 89,26% a pre distribuovanú verziu iba 42,83%. Najväčší rozdiel v priemernej efektivite bol pre hash Whirlpool (90,61% ku 26,31%) avšak s takmer o polovicu nižšou priemernou rýchlosťou lámania (pre zdieľanú verziu 612 137 a pre distribuovanú verziu 991 814 hesiel

Obr. 6.2: Slovníkový útok - graf lámania 9 znakového (SHA-1) hesla nástrojom mpiHashcat

za sekundu). Tento nepomer medzi efektivitou je zapríčinený iným charakterom prenosu slovníka. Kým v distribovanej verzií sa nástroj hashcat spúšťa až po prijatí a uložení celého slovníka, v zdieľanej verzií sa spustí ihneď a priamo nástroj hashcat slovník sťahuje zo zdieľaného úložiska. Efektivita je preto vyššia a rýchlosť lámania nižšia.

Z uvedených testov je zrejmé, že technológia MPI nie je príliš vhodná na prenos veľkého objemu dát a ani odporúčané riešenie prenosu cez NFS tento problém neodstraňuje. Pri maskových útokoch sú však výsledky sľubné a pri dlhších úlohách sa dá dosiahnuť vysoká efektivita aj výkonnosť. Výhodou riešenia cez MPI je aj jednoduché spúšťanie nástroja. Nie je potrebné prihlasovanie, ani spúšťanie klientskych aplikácií na výpočtových uzloch (server priamo spúšťa klientske procesy cez službu SSH).V nástrojoch na distribúciu, ktoré nemajú príliš inteligentné adaptívne priraďovanie úloh je viac ako nutné správne zvoliť veľkosť čiastkových úloh.

6.3.2 Hashtopolis

Nástroj Hashtopolis si vyžaduje, oproti nástroju mpiHashcat, iný typ počiatočnej konfigurácie. Kým u mpiHashcat je potrebná konfigurácia na nižšie úrovní (MPI, SSH, NFS), pri nástroji Hashtopolis je potrebné vytvorenie nového klienta na strane serveru a vygenerovanie unikátneho žetónu (ang. *token*), ktorým sa klient autentizuje voči serveru. Je tak viac užívateľsky prívetivý, minimálne vďaka poskytovanému webovému rozhraniu. Aj keď niektoré merania v podobných podmienkach už v minulosti prebehli (čitateľ ich môže nájsť v kapitole 5 článku [15]), v tejto kapitole budú prezentované iba nami namerané testy z rovnakej sady útokov ako pre ostatné technológie.

Slovník	Počet uzlov	Čas zdieľ. [s]	Stroj. čas zdieľ. [s]	Čas dist. [s]	Stroj. čas dist. [s]	Rozdiel čas [s]	Rozdiel stroj. čas [s]
rockyou.txt	2	56,5	106,6	55,2	90,2	+1,3	+16,5
rockyou.txt	4	33,0	121,1	30,8	93,7	+2,2	+27,4
rockyou.txt	8	24,3	168,0	19,2	102,0	+5,0	+66,0
rockyou.txt	16	27,3	390,7	19,3	131,8	+8,1	+258,9
24xrockyou.txt	2	1337,9	2605,4	731,7	1039,8	**+606,2**	+1565,6
24xrockyou.txt	4	728,3	2782,5	391,0	1039,2	+337,3	+1743,3
24xrockyou.txt	8	520,2	3873,8	299,4	1031,3	+220,7	+2842,5
24xrockyou.txt	16	574,5	8614,0	293,7	1046,8	+280,9	+7567,1

Tabuľka 6.7: Porovnanie slovníkového útoku pre zdieľanú a distribuovanú verziu nástroja mpiHashcat

Maskový útok

Počiatočná réžia na zahájenie útoku pri nástroji Hashtopolis nie je zanedbateľná, hlavne pri krátkych útokoch. Pravidlo, že krátke úlohy nie sú vhodné na distribúciu sa potvrdzuje.

Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť na 1 uzol [P/s]
7	2	60	20	9	**22,50%**	77,50%	200 795 254
7	4	60	18	8	11,11%	88,89%	111 552 919
7	8	60	17	9	6,62%	93,38%	59 057 428
7	16	60	17	8	**2,94%**	97,06%	29 528 714
8	2	60	90	140	77,78%	22,22%	**1 160 150 359**
8	4	60	85	142	41,76%	58,24%	614 197 249
8	8	60	88	143	20,31%	79,69%	296 629 353
8	16	60	85	140	10,29%	89,71%	153 549 312
9	2	60	1935	3766	97,31%	2,69%	1 402 972 527
9	4	60	1001	3769	94,13%	5,87%	1 356 019 900
9	8	60	534	3793	88,79%	11,21%	1 270 951 236
9	16	60	275	3762	85,50%	14,50%	1 233 978 109
9	2	1800	1890	3573	94,52%	5,48%	1 436 376 635
9	4	1800	1860	3574	48,04%	51,96%	729 772 000
9	8	1800	1852	3629	24,49%	75,51%	366 462 181
9	16	1800	1864	3565	11,95%	88,05%	182 051 491

Tabuľka 6.8: Vybrané výsledky maskového útoku nástrojom Hashtopolis na hash SHA-1

V tabuľke 6.8 sú uvedené niektoré výsledky pre maskový útok na hash typu SHA-1. Pri dĺžke hesla 7 znakov trvá útok takmer rovnako pri 2, 4, 8 aj 16 uzloch. Po preskúmaní priebehu útoku zistíme, že prvých približne 7 sekúnd si vyžaduje prijatie úlohy, počiatočný benchmark a potvrdenie prijatia úlohy. Následne server vyhodnotí (podľa získaných rýchlostí klientov), že celá úloha bude trvať kratšie ako 60 sekúnd a priradí ju jednému klientovi. Ak si pri dvoch uzloch vyžiada polovicu času počiatočná réžia a druhú polovicu času po-

číta jeden z dvoch uzlov, dostávame efektivitu okolo 25%. Tá s pribúdajúcim počtom uzlov klesá, až na 2,94%.

Dĺžka hesla 8 znakov je vhodnejšia na distribúciu a pri 2 uzloch dosahujeme priemernú rýchlosť lámania na 1 uzol 1 160 150 359 hesiel za sekundu pri efektivite 77,78%. Pri 4 a viac uzloch narážame na rovnaký problém, úloha nie je dostatočne veľká na vyťaženie všetkých uzlov a viac času sa trávi čakaním ako lámaním.

Pri dĺžke 9 znakov sa s rozumne nízkym nastavením čiastkovej úlohy dostávame na efektivitu nad 85% pre 16 uzlov. Pri veľkosti úlohy 1800 sekúnd je znova pridelená veľká úloha malému počtu klientov. Pri 2 uzloch je to výhoda a dostávame čas výborných 1890 sekúnd avšak pri 16 uzloch tento čas ostáva takmer nezmenený a efektivita klesla pod 12%.

Pre lámaní hesla typu MD5 bol trend podobný ako pri SHA-1, najvyššiu efektivitu sme dosiahli pri hľadaní 9 znakového hesla, 60 sekundových úlohách a 2 uzloch, presných 97%. V rovnakom meraní sme namerali maximálnu rýchlosť na 1 uzol, 2 170 064 419 hesiel za sekundu. Najhoršiu rýchlosť 33 456 876 hesiel za sekundu a efektivitu slabých 2,92% sme dosiahli pri 7 znakovom hesle s čiastkovou úlohou 1800 sekúnd.

Lámanie hesla dĺžky 9 znakov pri použitom type Whirlpool trvalo najkratšie 5 805 sekúnd na 16 uzloch s čiastkovou úlohou 600 sekúnd. Dosiahnutá rýchlosť bola 58 406 881 hesiel za sekundu. Priemerná pre efektivita pre všetky testy bola 51,61% a priemerná rýchlosť na 1 uzol bola 30 845 292 hesiel za sekundu. Priemer znižovali krátke heslá, pri ktorých bola réžia na úrovni 89% a viac.

Slovníkový útok

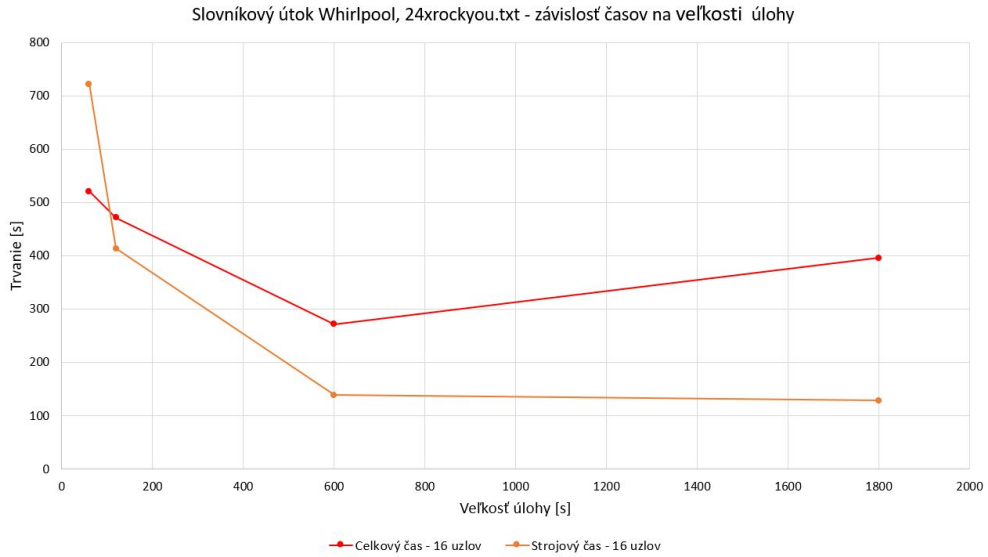
Nástroj Hashtopolis prenáša pri slovníkových útokoch celé slovníky na všetky priradené klientske uzly, paralelne. To znamená, že rýchlosť prenosu na jeden klientsky uzol je rovný šírke pásma podelený počtom klientskych uzlov. Tento fakt v kombinácií s veľkým slovníkom sa môže podpísať pod neúmerne vysokú réžiu na relatívne krátkom útoku.

Slovník	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť celková [P/s]
rockyou.txt	2	60	23	15	32,61%	67,39%	623 669
rockyou.txt	4	60	21	10	11,90%	88,10%	683 066
rockyou.txt	8	60	20	8	5,00%	95,00%	717 220
rockyou.txt	16	60	28	14	3,13%	96,88%	**512 300**
6xrockyou.txt	2	60	90	47	26,11%	73,89%	956 293
6xrockyou.txt	4	60	90	56	15,56%	84,44%	956 293
6xrockyou.txt	8	60	94	77	10,24%	89,76%	915 599
6xrockyou.txt	16	60	122	67	3,43%	96,57%	705 462
24xrockyou.txt	2	60	353	270	38,24%	61,76%	975 256
24xrockyou.txt	4	60	339	318	23,45%	76,55%	**1 015 532**
24xrockyou.txt	8	60	397	313	9,86%	90,14%	867 167
24xrockyou.txt	16	60	496	309	3,89%	96,11%	694 083

Tabuľka 6.9: Slovníkový útok na hash typu SHA-1 - Hashtopolis

V tabuľke 6.9 sú zaznamenané namerané hodnoty slovníkového útoku pre 60 sekundovú úlohu na troch rôznych slovníkoch. Maximálnu rýchlosť sme dosiahli na 4 uzloch so slovníkom `24xrockyou.txt` a minimálnu pre 16 uzlov so slovníkom `rockyou.txt`. Priemerná

efektivita je pre všetky typy pod 10%. Priemerná rýchlosť lámania pre MD5 je 227 123, pre SHA-1 224 208 a pre Whirlpool 45 823 hesiel za sekundu.



Obr. 6.3: Slovníkový útok, Whirlpool, 24xrockyou.txt - porovnanie reálneho a strojového času útoku nástrojom Hashtopolis

Obrázok 6.3 znázorňuje vzťah medzi reálnym a strojovým časom pre slovníkový útok na hash typu Whirlpool s použitým slovníkom `24xrockyou.txt`. Pre 60 sekundové úlohy bol reálny čas kratší ako strojový s efektivitou 8,66% a pri dlhších úlohách bol strojový čas výrazne dlhší. Najnižší reálny čas 272 sekúnd sme namerali pri 600 sekundovej úlohe s celkovou rýchlosťou 1 265 681 hesiel za sekundu.

Realizované testy nástrojom Hashtopolis ukázali potenciál v dlhých maskových útokoch. Pri rozsiahlejších slovníkových útokoch sa prejavila výrazne slabá škálovateľnosť. Nevýhodou nástroja je tiež problematická automatizácia a v niektorých prípadoch užívateľské rozhranie. Ako príklad môžeme uviesť vytváranie novej úlohy, pri ktorej je potrebná znalosť parametrov príkazového riadku nástroja hashcat. Následne, po vytvorení úlohy je nutné manuálne pridávať klientov, jeden po druhom. Bez zásahu do (nezdokumentovanej) databázy je presné meranie alebo automatizácia takmer nemožná. Klientska aplikácia má formu konzolovej aplikácie, čo má v niektorých prípadoch svoje limity (pri veľkom počte uzlov môže byť prihlásenie do operačného systému a spustenie klientskej aplikácie zdĺhavé).

6.3.3 Fitcrack

Nástroj Fitcrack sa rovnako ako Hashtopolis skladá zo serveru a klientov. Na inštaláciu serveru je k dispozícií inštalátor, ktorý pripraví prostredie na beh serveru vrátane všetkých jeho súčastí. Distribúciu a výmenu správ zabezpečuje technológia BOINC. Pre registráciu výpočtového uzlu stačí jednoducho zadať URL adresu serveru, projektu a prihlasovacie údaje nového alebo existujúceho užívateľa.

Maskový útok

V tabuľke 6.10 sú uvedené namerané hodnoty pri útokoch na hash SHA-1. Pravidlo, že kratšie úlohy nie sú vhodné na distribúciu tu platí obzvlášť. Počiatočný benchmark a úvodná

Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť na 1 uzol [P/s]
7	2	600	91	26,3	14,45%	85,55%	44 130 825
7	4	600	92	42,7	11,61%	88,39%	21 825 571
7	8	600	85	75,3	11,07%	88,93%	11 811 486
7	16	600	81	140,9	10,87%	89,13%	6 197 384
8	2	120	180	176,0	48,88%	51,12%	580 075 179
8	4	120	129	191,4	37,09%	62,91%	404 703 614
8	8	120	166	223,8	16,85%	83,15%	157 249 296
8	16	120	122	289,4	14,83%	85,17%	106 981 078
9	2	60	2643	3892,6	73,64%	26,36%	1 027 147 877
9	4	60	1381	3909,3	70,77%	29,23%	982 893 497
9	8	60	772	3967,1	64,23%	35,77%	879 129 482
9	16	60	436	4052,8	58,10%	41,90%	778 311 881
9	2	1800	2323	3786,5	81,50%	18,50%	**1 168 640 482**
9	4	1800	**1410**	3965,7	70,31%	29,69%	962 677 957
9	8	1800	**840**	4089,3	60,85%	39,15%	807 961 857
9	16	1800	**470**	4162,9	55,36%	44,64%	722 008 468

Tabuľka 6.10: Vybrané výsledky maskového útoku nástrojom Fitcrack na hash SHA-1

inicializácia trvá v niektorých prípadoch skoro minútu. Najvyššiu efektivitu 81,5% sme dosiahli pre 2 uzly pri 1800 sekundovej úlohe pri priemernej rýchlosti 1 168 640 482 hesiel za sekundu na 1 uzol. Všimnime si, že pri hesle dĺžky 9 znakov a úlohe 1800 sekúnd skončí útok pre 4, 8 a 16 uzlov skôr ako za 1800 sekúnd. Fitcrack za to vdačí inteligentnému adaptívnemu priraďovaniu úloh, ktoré minimalizuje čakanie uzlov na dokončenie útoku.

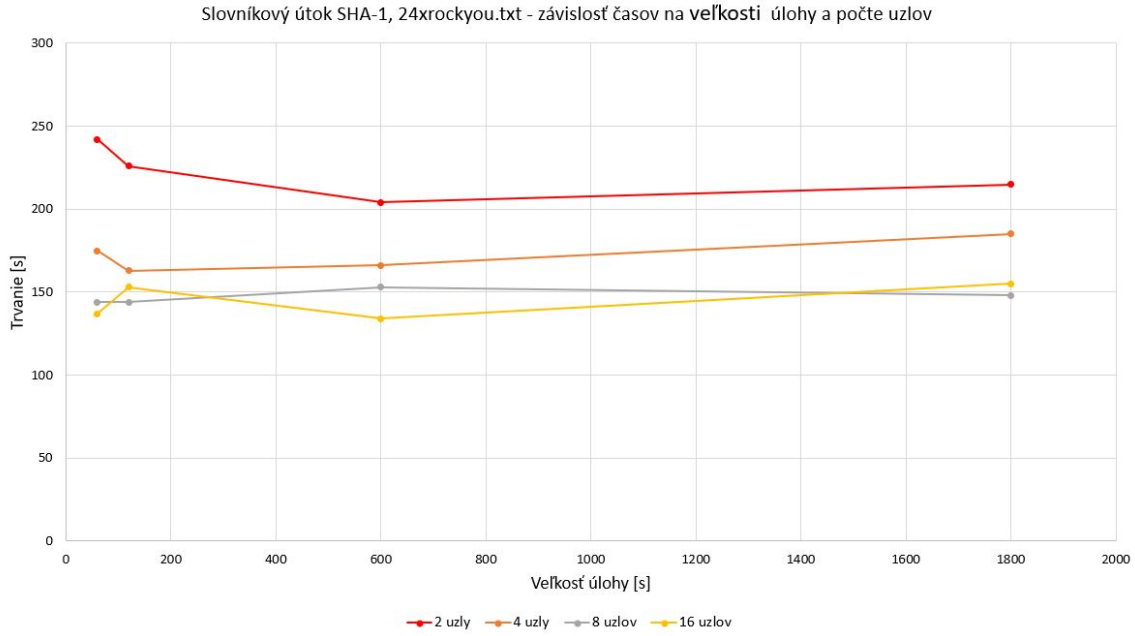
Dĺžka hesla	Počet uzlov	Veľkosť úlohy [s]	Reálny čas [s]	Strojový čas [s]	Efektivita [%]	Réžia [%]	Rýchlosť celková [P/s]
8	16	60	449	4085,8	56,87%	43,13%	465 093 685
8	16	120	434	4089,4	58,89%	41,11%	481 168 352
8	16	600	498	4429,3	55,59%	44,41%	419 331 455
8	16	1800	508	4565,6	56,17%	43,83%	411 076 899
9	16	60	8179	98411,2	**75,20%**	24,80%	663 834 659
9	16	120	6876	93760,0	85,22%	14,78%	789 631 134
9	16	600	6357	91494,0	89,95%	10,05%	854 098 424
9	16	1800	6355	91551,7	**90,04%**	9,96%	**854 367 219**

Tabuľka 6.11: Vybrané výsledky maskového útoku nástrojom Fitcrack na hash Whirlpool

Situácia pri lámaní 8 a 9 znakového hesla typu Whirlpool je znázornená v tabuľke 6.11. Pre 9 znakové heslo pozorujeme, že efektivita rastie so zväčšujúcou sa dĺžkou úlohy. Adaptivitou predchádza čakaniu na posledný uzol a znižuje počet vymenených správ, z ktorých každá si vyžaduje spustenie nástroja na lámanie (a tým zníženie výkonnosti). Úloha dĺžky 60 sekúnd dosiahla efektivitu 75,2% a skladala sa z 1725 čiastkových úloh, kým pri 1800 sekundách stačilo 615 čiastkových úloh a efektivitou prekračujeme hranicu 90%.

Slovníkový útok

Nástroj Fitcrack berie ohľad na fakt, že prenos slovníkov je veľmi zdĺhavý proces a preto slovníkové útoky rieši fragmentáciou. Každému klientovi je zasielaná iba časť slovníka, ktorá je potrebná na vykonanie čiastkovej úlohy a predíde sa tak redundantnému posielaniu toho istého slovníka mnoho uzlom.



Obr. 6.4: Slovníkový útok, Whirlpool, 24xrockyou.txt - porovnanie reálneho a strojového času útoku nástrojom Fitcrack

Na obrázku 6.4 vidíme graf porovnania slovníkového útoku na SHA-1 pre rôzny počet uzlov a rôzne veľkosti úloh. Použitý je najväčší zo slovníkov *24xrockyou.txt*, ktorý má viac ako 3GB. Všimnime si, že aj napriek veľkosti slovníka pridávanie uzlov celkový čas znižuje. Musíme podotknúť, že technológia BOINC podporuje aj paralelné sťahovanie novej úlohy kým pôvodná ešte beží (možnosť *batch workunit assignment*) a tým znížiť réžiu na prenos dát. V našich experimentoch sme túto možnosť mali vypnutú.

Pri útokoch na MD-5 sme dosahovali priemernú efektivitu 15,54% a priemernú rýchlosť lámania na 1 uzol 214 536 hesiel za sekundu. Maximálna efektivita 37,19% a maximálna priemerná rýchlosť 1 uzlu 882 732 hesiel za sekundu bola dosiahnutá pri slovníku `24xrockyou.txt` a čiastkovej úlohe 1800 sekúnd na dvoch uzloch.

Lámanie SHA-1 dopadlo podobne, priemerná efektivita bola na úrovni 15,35%, priemerná rýchlosť lámania na 1 uzol bola 208 696 (SHA-1) hesiel za sekundu a maximálna rýchlosť lámania celého systému 2 569 144 hesiel za sekundu (pri 16 uzloch, 600 sekundovej čiastkovej úlohe a slovníku `24xrockyou.txt`).

Hash typu Whirlpool sme, rovnako ako pri ostatných technológiách, spúšťali iba na 16 uzloch a dosiahli sme maximálnu efektivitu 14,71%, znova pre najväčší slovník so 60 sekundovou čiastkovou úlohou. Priemerná rýchlosť lámania na 1 uzol bola 60 996 hesiel za sekundu, pričom maximálna nameraná bola 139 718 hesiel za sekundu. Z nameraných výsledkov je zrejmé, že efektivita aj priemerná rýchlosť lámania na 1 uzol rastie so zväčšujúcim

sa slovníkom (pri `rockyou.txt` je 9 643, pri `6xrockyou.txt` 42 866 a pri `24xrockyou.txt` 130 478 hesiel za sekundu).

Nástroj Fitcrack predstavuje komplexný systém na distribúciu. Pri jeho adaptívnom plánovači úloh sa netreba báť ani väčších čiastkových úloh, ktoré v prípade potreby upraví tak, aby sa minimalizovalo čakanie uzlov. Platforma BOINC poskytuje komplexné riešenie distribúcie, ktoré si vyžaduje viac času na inicializáciu úloh. Na druhú stranu, je použiteľné v sieti internet, odolné proti výpadkom, môže bežať na pozadí ako služba a poskytuje množstvo doplnkov (možnosť účtovania a pod.).

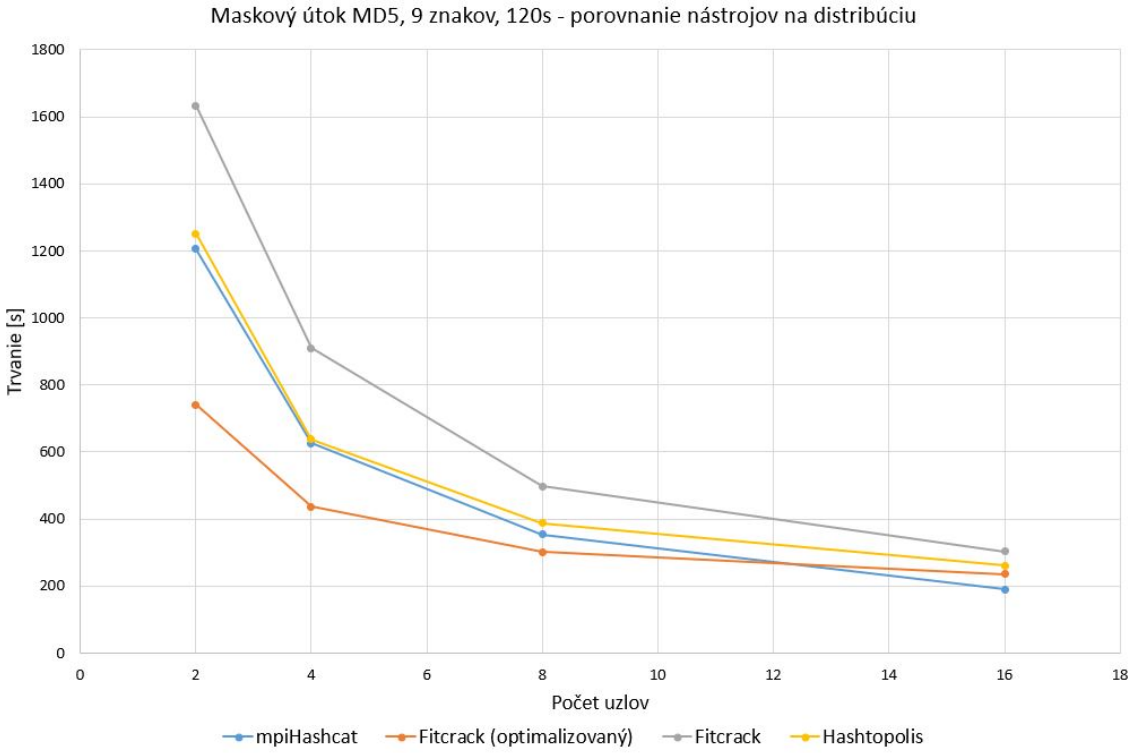
6.4 Porovnanie

Technológie pre distribúciu sme porovnali a z výsledných hodnôt sme vypočítali súhrnné ukazovatele. Pre maskový útok sme priemernú efektivitu a priemernú rýchlosť lámania jedného uzlu zaznamenali do tabuľky 6.12. Ide o priemerné hodnoty celej testovacej sady maskových útokov, preto je nutné brať ohľad na charakter testov, ktoré boli v niektorých prípadoch krátke (čo môže značne znižovať efektivitu aj rýchlosť) alebo iným spôsobom neoptimálne (veľký počet uzlov na kratšiu úlohu, prípadne čakanie na uzol pri vysoko nastavenej čiastkovej úlohe).

	Priemerná efektivita [%]			Priemerná rýchlosť na 1 uzol [P/s]		
	MD5	SHA-1	Whirlpool	MD5	SHA-1	Whirlpool
mpiHashcat	59,56	62,06	61,09	873 980 026	649 543 436	35 851 486
Fitcrack	33,67	35,92	52,44	596 888 196	418 041 380	26 994 589
Hashtopolis	32,68	36,78	51,61	712 433 396	524 043 448	30 845 291

Tabuľka 6.12: Priemerné namerané hodnoty technológií pre maskový útok

Pri meraniach každej technológie bola použitá základná inštalácia nástroja, bez dodatočnej konfigurácie. Pri nástroji Fitcrack sme po nameraní zistili, že v niektorých prípadoch má značne rozdielny súčet strojových časov ako ostatné technológie v rovnakej konfigurácií. Po detailnej analýze sme tento rozdiel pripísali prepínaču nástroja hashcat `--optimized-kernel-enable` (povolí optimalizované OpenCL kernely pre rýchlejšie lámanie hesiel ohraničenej dĺžky), ktorý je v nástroji Fitcrack automaticky zapnutý pri každom útoku. Rozdiel je vidieť pri dlhších úlohách niektorých typov hashu (napr. pri Whirlpool bol rozdiel nepatrný). Jeden z príkladov, kedy má optimalizácia zmysel je na grafe 6.5. Ide o relatívne dlhšie úlohy na rôznom počte uzlov. Pri 2 uzloch zapnutá optimalizácia znížila celkový čas na menej ako polovicu pôvodného.
Aby sme dosiahli rovnakých podmienok, bolo potrebné manuálne odstrániť tento prepínač zo zdrojových kódov modulu `runner`, preložiť ho a prepísať pôvodný modul na klientskych uzloch. Namerané hodnoty pre obe varianty sme sa rozhodli zapracovať do grafu 6.6, ktorý zobrazuje nameranú dĺžku celého útoku na heslá dĺžky 7, 8 a 9 znakov s nastavenou čiastkovou úlohou 60 sekúnd. V ľavej časti grafu, pri krátkych útokoch, pozorujeme réžiu spojenú so spustením útoku. Tento rozdiel je konštantný a ovplyvňuje skôr krátke útoky. Preto veľká časť rozdielu pri 9 znakových heslách v pravej časti grafu vychádza z konštantne vyššej réžie pri spustení.
Priemerné namerané hodnoty efektivity a rýchlosť lámania na 1 uzol pre slovníkové útoky sú uvedené v tabuľke 6.13. Znova ide o súhrnný údaj všetkých testovacích scenárov. Jednoznačne môžeme potvrdiť, že medzi efektivitou a réžiou neexistuje vzťah. Pri každej tech-



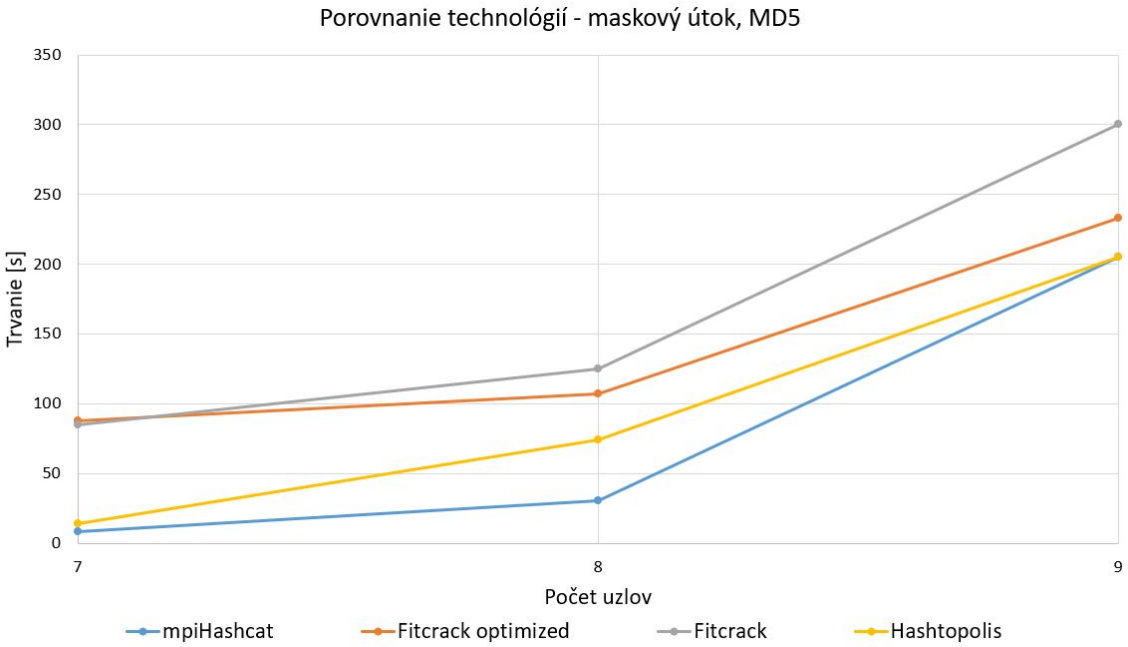
Obr. 6.5: Porovnanie technológií pri maskovom útoku na hash MD5

nológií závisí na mnoho faktoroch, ako sa rýchlo bude útok dokončený. Hlavným faktorom je spôsob prenosu slovníka. Služba NFS, použitá v nástroji mpiHashcat so zdieľanou pamäťou, je vhodná hlavne pri menších slovníkoch. Z výsledkov experimentov bolo zrejmé, že služba NFS prenáša celý slovník v momente, keď ho niektorý proces začne využívať a to bez ohľadu či ho potrebuje celý, alebo iba minimálnu časť z neho. Vďaka tomuto faktu má nástroj mpiHashcat so zdieľanou pamäťou vysokú efektivitu pri nízkej priemernej rýchlosti lámania. Porovnanie technológií v závislosti na veľkosti slovníka pre SHA-1 je znázornené na obrázku 6.7.

Ďalším zistením bolo, že rozhranie MPI nie je vhodné na prenos väčšieho objemu dát. I keď sme implementovali fragmentáciu slovníka, aby sme minimalizovali dopad prenosu, výsledky boli slabé s vysokými nárokmi na zdroje. Maximálna veľkosť jednej prenesenej MPI správy je 2GB, čiže pri väčších súboroch sa fragmentácií nedá vyhnúť. Nevýhodou MPI

	Priemerná efektivita [%]			Priemerná rýchlosť na 1 uzol [P/s]		
	MD5	SHA-1	Whirlpool	MD5	SHA-1	Whirlpool
mpiHashcat distrib. pamäť	42,83	43,88	26,31	160 817	159 390	61 998
mpiHashcat zdieľaná pamäť	89,26	89,38	90,61	173 391	174 288	38 258
Fitcrack	15,54	15,35	11,5	214 536	208 696	60 996
Hashtopolis	9,34	9,77	3,84	227 123	224 208	45 823

Tabuľka 6.13: Priemerné namerané hodnoty technológií pre slovníkový útok

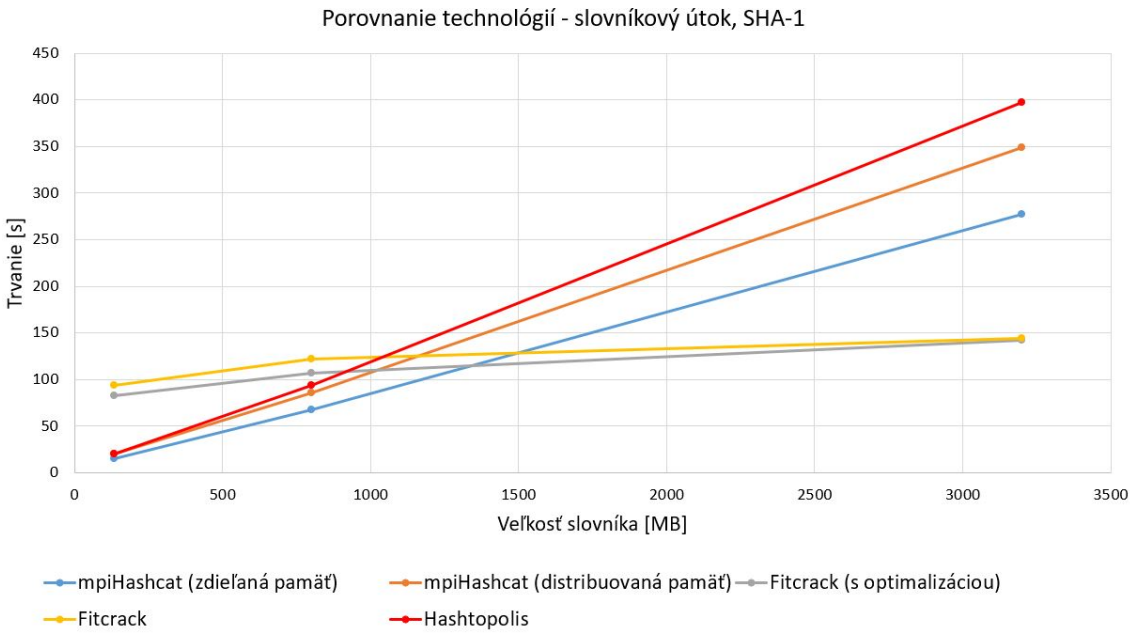


Obr. 6.6: Porovnanie technológií pri maskovom útoku na hash MD5

okrem spomenutého je aj problematické dynamické pridávanie klientov (pomocou funkcie `Spawn`[7] od štandardu MPI-1.3), rovnako aj problematické zotavovanie po výpadku pripojenia. Ideálne využitie MPI je v lokálnej sieti s nízkou latenciou a minimálnymi výpadkami.

Nástroj Hashtopolis využíva protokol HTTP k prenosu dát, prenáša v zásade celý slovník na priradených klientov a to bez výnimky. Problém nastáva, ak je úloha menšia, alebo veľkosť čiastkovej úlohy veľká natoľko, aby bola celá úloha priradená jednému alebo malej množine klientov. I keď je zo vstupných parametrov známe, že úlohu bude riešiť jeden klient, slovník sa pred útokom prenesie všetkým klientom, čo značne spomaľuje útok.

[7] https://www.mpich.org/static//docs/v3.0.x/www3/MPI_Comm_spawn.html



Obr. 6.7: Porovnanie technológií pri slovníkovom útoku na hash SHA-1

Kapitola 7

Záver

Problematika obnovy hesiel je vo všeobecnosti veľmi náročný problém. Akcelerovať ho môžeme nielen sofistikovanejšími generátormi hesiel a lepšími slovníkmi, ale aj zvyšovaním výpočtovej sily. Veľký skok v akcelerácií procesu lámania hesiel bol presun výpočtu na GPU, avšak aj táto akcelerácia je v mnoho prípadoch nedostatočná. Je to z dôvodu exponenciálneho nárastu možných hesiel pri zvyšovaní dĺžky hesla, ale aj zložitosťou použitých hashovacích algoritmov.

Distribuovaný výpočet je v dnešnej dobe jediné riešenie prekonávajúce obmedzenia jedného počítača. Technológií pomocou ktorých môžeme výpočet distribuovať je viac a nie každá je vhodná na lámanie hesiel. V kapitole 3 tejto práce sme uviedli technológie VirtualCL, BOINC, MPI ale aj možnosť využitia vlastného protokolu nad HTTP pre dosiahnutie distribúcie výpočtu. Pre objektívne porovnanie technológií na distribúciu sme v kapitole 4 navrhli jednotné prostredie vrátane samotného nástroja na lámanie hesiel hashcat.

Distribuovaný nástroj nad MPI využívajúci nástroj hashcat podľa našich vedomostí neexistuje, preto sme sa ho rozhodli implementovať. Špecifikácia MPI má viacero open-source aj komerčných implementácií, ktoré sme porovnali a zhodnotili ich silné a slabé stránky. Za najvhodnejšiu implementáciu považujeme MPICH, ktorá je oproti svojmu rivalovi OpenMPI modernejšia. Má efektívnejšie zvládnutú prácu so zdieľanou pamäťou, výbornú prenositeľnosť a kvalitne implementované kolektívne operácie a komunikácie. Potvrdzuje to aj fakt, že z nej vychádzajú ďalšie, odvodené, implementácie ako napr. komerčný IntelMPI alebo open-source MVAPICH. V kapitole 5 sme popísali architektúru nami implementovaného nástroja mpiHashcat, jeho rozhranie aj implementačné detaily. Okrem verzie s distribuovanou pamäťou sme implementovali aj verziu so zdieľanou pamäťou využívajúcu službu NFS, ktorá patrí k odporúčaným v kombinácií s MPI.

V kapitole 6 uvádzame všetky testovacie scenáre, ktoré boli využité na merané technológie. Popísali sme tiež spôsob automatizácie a zberu dát z experimentov. Výsledky pre každú technológiu sme zhodnotili a porovnali.

Technológia VirtualCL bola pri prvých verziách nástroja hashcat priamo odporúčaná na distribúciu výpočtu, aj keď sa javí ako nevhodný kandidát kvôli vysokej réžií a citlivosti na rýchlosť odozvy na sieti. Ide navyše o zastaralú a nepodporovanú technológiu, ktorá má mnohé obmedzenia v kontexte lámania hesiel a to ju dnes tvorí takmer nepoužiteľnou. Platforma BOINC vykazuje veľmi sľubné výsledky a s nástrojom Fitcrack tvorí robustný, odolný, ale zároveň prispôsobivý systém na lámanie hesiel. Svojou povahou sa platforma BOINC hodí pre stredne dlhé až dlhé útoky, pri ktorých je zvýšená réžia na spustenie útoku bezvýznamná. Jeho silnou stránkou je tiež efektívny prenos aj rozsiahlych súborov, čo sa prejavilo pri slovníkových útokoch. Vlastný protokol pre distribúciu využíva

nástroj Hashtopolis, ktorý sa pýši okrem prívetivého webového rozhrania aj automatickou distribúciou aplikácií na lámanie a mnohým ďalším. Ide o platformovo nezávislé riešenie typu klient-server, ktoré umožňuje použiť ľubovoľný nástroj na lámanie hesiel. Táto technológia dosahuje vysokej efektivity pri dlhších maskových útokoch, avšak pri krátkych a stredne dlhých útokoch je potrebné správne zvoliť veľkosť čiastkových úloh. V opačnom prípade nástroj Hashtopolis neefektívne priradí úlohu klientom, čo je dôsledok menej prepracovaného adaptívneho prideľovania úloh. Technológia MPI vykazuje sľubné výsledky, hlavne pri maskových útokoch. Má nízku réžiu na spustenie útoku aj výmenu riadiacich správ, čo sa prejavilo nielen v krátkych útokoch. O niečo menej optimistické boli slovníkové útoky, hlavne pri prenose veľkých slovníkov. Môžeme konštatovať, že technológia MPI nie je vhodná na prenos veľkých dát ani v kombinácií so službou NFS.

Nadväzujúca práca sa môže venovať efektívnejšej distribúcií slovníkov klientom, prípadne integrovaniu distribúcie nad MPI priamo do nástroja hashcat. Pre zjednodušenie ovládania by bolo možné implementovať grafické rozhranie, prípadne doplniť implementáciu dynamického pridávania a odoberania klientov.

Literatúra

[1] Cain & Abel - official web. [Online; navštívené 27.12.2018].
URL http://www.oxid.it/cain.html

[2] Hashcat - official web. [Online; navštívené 27.12.2018].
URL https://hashcat.net/hashcat/

[3] Intel MPI - official web. [Online; navštívené 02.01.2019].
URL https://software.intel.com/en-us/mpi-library

[4] John The Ripper - official web. [Online; navštívené 26.12.2018].
URL https://www.openwall.com/john/doc/

[5] MPI - Official documentation. [Online; navštívené 31.12.2018].
URL https://www.mpi-forum.org/docs/

[6] Barak, A.; Shiloh, A.: The Virtual OpenCL (VCL) Cluster Platform. 2011.

[7] Dalcin, L. D.; Paz, R. R.; Kler, P. A.; aj.: Parallel distributed computing using Python. *Advances in Water Resources*, ročník 34, č. 9, 2011: s. 1124 – 1139, ISSN 0309-1708, new Computational Methods and Software Tools.
URL http://www.sciencedirect.com/science/article/pii/S0309170811000777

[8] Danalis, A.; Marin, G.; McCurdy, C.; aj.: The Scalable HeterOgeneous Computing (SHOC) benchmark suite. 01 2010, s. 63–74, doi:10.1145/1735688.1735702.

[9] Eastlake, D.; Jones, P.: US Secure Hash Algorithm 1 (SHA1). RFC 3174, 9 2001.

[10] Fielding, R.; Gettys, J.; Mogul, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. Technická Správa 2616, Jún 1999, obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.
URL http://www.ietf.org/rfc/rfc2616.txt

[11] Hranický, R.: *Digitální forenzní analýza s pouzitím distribuovaného výpoctu*. Teze disertační práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.

[12] Hranický, R.; Holkovič, M.; Matoušek, P.; aj.: On Efficiency of Distributed Password Recovery. *The Journal of Digital Forensics, Security and Law*, ročník 11, č. 2, 2016: s. 79–96, ISSN 1558-7215.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11276

[13] Hranický, R.; Zobal, L.; Večeřa, V.: Distribuovaná obnova hesel. Technická správa, 2017.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11568

[14] Hranický, R.; Zobal, L.; Večeřa, V.; aj.: Distributed Password Cracking in a Hybrid Environment. In *Proceedings of SPI 2017*, University of defence in Brno, 2017, ISBN 978-80-7231-414-0, s. 75–90.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11358

[15] Hranický, R.; Zobal, L.; Večeřa, V.; aj.: Distribuce výpočtů pro nástroj hashcat. Technická správa, 2018.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11884

[16] Kos, O.: *Obnova hesel v distribuovaném prostředí*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.

[17] Lubeck, T.; Chow, M.: Distributed Password Cracking with John the Ripper. 2013.
URL https://pdfs.semanticscholar.org/1cff/54069db1a77b8799795fd61b903b612622ec.pdf

[18] McClements, E.: *Performance Comparison of Open Source MPI Implementations*. Diplomová práca, The University of Edinburgh, 2006.
URL https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2005-2006/2688821-9h-dissertation1.1.pdf

[19] Rivest, R.: The MD5 Message-Digest Algorithm. Technická Správa 1321, Apríl 1992, updated by RFC 6151.
URL http://www.ietf.org/rfc/rfc1321.txt

[20] Weir, C. M.: *Using Probabilistic Techniques to Aid in Password Cracking Attacks*. Disertační práce, The Florida State University, College of Arts and Sciences, 2010.
URL https://diginole.lib.fsu.edu/islandora/object/fsu%3A175769

[21] Zobal, L.: *Distribuovaná obnova hesel s využitím nástroje hashcat*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=21072

Príloha A

Obsah priloženého CD

- `/latex` - priečinok obsahujúci zdrojové súbory textu práce v LaTeX
- `/src` - priečinok obsahujúci zdrojové súbory nástroja mpiHashcat vrátane skriptov na automatizáciu a zber dát
- `/doc` - priečinok s nameranými hodnotami vrátanie spracovania
- `/DP-PATRIK-MRAZ-2019.pdf` - súbor s textom tejto diplomovej práce

Príloha B

Namerané hodnoty

B.1 mpiHashcat

B.1.1 Maskový útok

Typ hashu	Počet znakov	Počet uzlov	Veľkosť úlohy [s]	Čas [s]	Čas spolu [s]
MD5	7	2	60	12,85	20,74
MD5	8	2	60	81,17	114,65
MD5	9	2	60	1250,28	2473,77
MD5	7	2	120	10,19	18,61
MD5	8	2	120	72,00	106,87
MD5	9	2	120	1205,41	2399,61
MD5	7	2	600	10,61	13,36
MD5	8	2	600	85,83	105,14
MD5	9	2	600	1236,68	2345,19
MD5	7	2	1800	10,60	13,36
MD5	8	2	1800	60,65	105,53
MD5	9	2	1800	1694,26	2339,96
MD5	7	4	60	8,61	25,06
MD5	8	4	60	44,07	130,28
MD5	9	4	60	640,21	2481,38
MD5	7	4	120	9,52	23,84
MD5	8	4	120	71,87	117,84
MD5	9	4	120	625,48	2405,36
MD5	7	4	600	10,54	20,68
MD5	8	4	600	59,19	117,45
MD5	9	4	600	816,02	2353,46
MD5	7	4	1800	10,61	21,49
MD5	8	4	1800	46,65	109,77
MD5	9	4	1800	1656,90	2348,21
MD5	7	8	60	8,50	40,47
MD5	8	8	60	30,76	154,66
MD5	9	8	60	337,22	2519,13
MD5	7	8	120	10,59	37,51

MD5	8	8	120	73,73	144,19
MD5	9	8	120	352,88	2439,98
MD5	7	8	600	10,48	33,36
MD5	8	8	600	21,50	136,19
MD5	9	8	600	517,37	2380,35
MD5	7	8	1800	10,59	34,25
MD5	8	8	1800	47,42	122,96
MD5	9	8	1800	1709,16	2375,85
MD5	7	16	60	8,60	82,93
MD5	8	16	60	30,64	199,79
MD5	9	16	60	204,92	2575,29
MD5	7	16	120	9,49	60,06
MD5	8	16	120	55,68	187,12
MD5	9	16	120	190,29	2497,88
MD5	7	16	600	9,49	59,26
MD5	8	16	600	20,62	162,68
MD5	9	16	600	532,26	2421,29
MD5	7	16	1800	9,49	60,07
MD5	8	16	1800	46,51	151,24
MD5	9	16	1800	1679,43	2411,46
SHA-1	7	2	60	14,18	22,73
SHA-1	8	2	60	88,50	162,41
SHA-1	9	2	60	1906,65	3804,46
SHA-1	7	2	120	11,74	19,89
SHA-1	8	2	120	83,61	159,16
SHA-1	9	2	120	1871,51	3683,01
SHA-1	7	2	600	12,76	15,65
SHA-1	8	2	600	126,85	152,85
SHA-1	9	2	600	1840,60	3597,73
SHA-1	7	2	1800	12,70	16,16
SHA-1	8	2	1800	87,96	151,97
SHA-1	9	2	1800	1795,32	3582,5
SHA-1	7	4	60	9,69	30,30
SHA-1	8	4	60	78,48	178,33
SHA-1	9	4	60	975,72	3813,44
SHA-1	7	4	120	11,69	26,44
SHA-1	8	4	120	81,03	169,51
SHA-1	9	4	120	966,36	3705,65
SHA-1	7	4	600	12,79	23,28
SHA-1	8	4	600	85,13	164,02
SHA-1	9	4	600	1143,53	3609,95
SHA-1	7	4	1800	12,74	23,28
SHA-1	8	4	1800	67,75	156,24
SHA-1	9	4	1800	1744,68	3591,26
SHA-1	7	8	60	9,68	41,87

SHA-1	8	8	60	35,96	207,20
SHA-1	9	8	60	502,37	3851,14
SHA-1	7	8	120	11,58	38,42
SHA-1	8	8	120	81,01	194,90
SHA-1	9	8	120	523,83	3742,32
SHA-1	7	8	600	11,70	35,27
SHA-1	8	8	600	28,40	185,14
SHA-1	9	8	600	567,50	3644,70
SHA-1	7	8	1800	11,69	35,25
SHA-1	8	8	1800	68,66	171,82
SHA-1	9	8	1800	1774,50	3632,59
SHA-1	7	16	60	10,25	94,34
SHA-1	8	16	60	35,15	257,62
SHA-1	9	16	60	266,65	3915,40
SHA-1	7	16	120	10,59	61,51
SHA-1	8	16	120	80,01	248,38
SHA-1	9	16	120	312,33	3789,93
SHA-1	7	16	600	10,59	61,87
SHA-1	8	16	600	27,61	209,05
SHA-1	9	16	600	560,50	3681,97
SHA-1	7	16	1800	10,59	61,87
SHA-1	8	16	1800	68,65	197,43
SHA-1	9	16	1800	1797,52	3684,18
Whirlpool	7	16	60	37,68	220,98
Whirlpool	8	16	60	253,45	3660,52
Whirlpool	9	16	60	5791,81	92229,85
Whirlpool	7	16	120	62,65	211,08
Whirlpool	8	16	120	285,64	3502,73
Whirlpool	9	16	120	5542,36	88414,68
Whirlpool	7	16	600	92,68	205,42
Whirlpool	8	16	600	271,91	3439,32
Whirlpool	9	16	600	5607,74	86319,83
Whirlpool	7	16	1800	93,73	209,35
Whirlpool	8	16	1800	785,44	3421,89
Whirlpool	9	16	1800	6061,90	86031,75

B.1.2 Slovníkový útok s distribuovanou pamäťou

Typ hashu	Slovník	Počet uzlov	Veľkosť úlohy [P]	Čas [s]	Čas spolu [s]
MD5	rockyou.txt	2	500000	55,19	90,16
MD5	6xrockyou.txt	2	500000	307,52	508,34
MD5	24xrockyou.txt	2	500000	1227,49	2040,51
MD5	rockyou.txt	2	1000000	35,94	50,13
MD5	6xrockyou.txt	2	1000000	190,05	266,02
MD5	24xrockyou.txt	2	1000000	731,69	1039,77
MD5	rockyou.txt	2	3000000	28,20	26,76

MD5	6xrockyou.txt	2	3000000	134,99	129,68
MD5	24xrockyou.txt	2	3000000	506,99	497,45
MD5	rockyou.txt	2	6000000	21,46	17,57
MD5	6xrockyou.txt	2	6000000	90,07	70,50
MD5	24xrockyou.txt	2	6000000	348,39	258,31
MD5	rockyou.txt	4	500000	30,81	93,74
MD5	6xrockyou.txt	4	500000	160,44	504,70
MD5	24xrockyou.txt	4	500000	627,11	1993,40
MD5	rockyou.txt	4	1000000	21,33	55,38
MD5	6xrockyou.txt	4	1000000	102,45	267,44
MD5	24xrockyou.txt	4	1000000	390,98	1039,22
MD5	rockyou.txt	4	3000000	21,33	31,44
MD5	6xrockyou.txt	4	3000000	90,65	132,6
MD5	24xrockyou.txt	4	3000000	343,97	492,25
MD5	rockyou.txt	4	6000000	20,76	23,37
MD5	6xrockyou.txt	4	6000000	86,76	75,25
MD5	24xrockyou.txt	4	6000000	344,40	257,79
MD5	rockyou.txt	8	500000	19,23	102,01
MD5	6xrockyou.txt	8	500000	89,43	507,71
MD5	24xrockyou.txt	8	500000	340,47	1971,97
MD5	rockyou.txt	8	1000000	18,21	64,64
MD5	6xrockyou.txt	8	1000000	79,68	273,51
MD5	24xrockyou.txt	8	1000000	299,41	1031,33
MD5	rockyou.txt	8	3000000	20,98	43,92
MD5	6xrockyou.txt	8	3000000	91,38	142,97
MD5	24xrockyou.txt	8	3000000	329,42	498,00
MD5	rockyou.txt	8	6000000	19,57	35,04
MD5	6xrockyou.txt	8	6000000	85,26	85,09
MD5	24xrockyou.txt	8	6000000	350,48	266,81
MD5	rockyou.txt	16	500000	19,25	131,82
MD5	6xrockyou.txt	16	500000	79,61	525,67
MD5	24xrockyou.txt	16	500000	289,02	1969,55
MD5	rockyou.txt	16	1000000	17,44	87,68
MD5	6xrockyou.txt	16	1000000	77,39	296,17
MD5	24xrockyou.txt	16	1000000	293,64	1046,81
MD5	rockyou.txt	16	3000000	19,19	66,59
MD5	6xrockyou.txt	16	3000000	84,58	164,44
MD5	24xrockyou.txt	16	3000000	326,29	519,01
MD5	rockyou.txt	16	6000000	20,27	57,95
MD5	6xrockyou.txt	16	6000000	88,88	108,18
MD5	24xrockyou.txt	16	6000000	346,78	285,19
SHA1	rockyou.txt	2	500000	57,14	93,84
SHA1	6xrockyou.txt	2	500000	317,20	520,49
SHA1	24xrockyou.txt	2	500000	1245,11	2063,40
SHA1	rockyou.txt	2	1000000	36,58	53,08

SHA1	6xrockyou.txt	2	1000000	187,93	265,80
SHA1	24xrockyou.txt	2	1000000	741,07	1058,88
SHA1	rockyou.txt	2	3000000	28,01	28,35
SHA1	6xrockyou.txt	2	3000000	129,46	130,82
SHA1	24xrockyou.txt	2	3000000	484,05	507,38
SHA1	rockyou.txt	2	6000000	22,33	19,73
SHA1	6xrockyou.txt	2	6000000	91,87	70,44
SHA1	24xrockyou.txt	2	6000000	338,96	259,44
SHA1	rockyou.txt	4	500000	31,40	96,04
SHA1	6xrockyou.txt	4	500000	163,46	519,92
SHA1	24xrockyou.txt	4	500000	642,08	2045,8
SHA1	rockyou.txt	4	1000000	23,15	58,12
SHA1	6xrockyou.txt	4	1000000	104,38	273,73
SHA1	24xrockyou.txt	4	1000000	395,81	1061,04
SHA1	rockyou.txt	4	3000000	21,24	31,69
SHA1	6xrockyou.txt	4	3000000	94,00	134,85
SHA1	24xrockyou.txt	4	3000000	342,62	504,67
SHA1	rockyou.txt	4	6000000	20,83	25,03
SHA1	6xrockyou.txt	4	6000000	84,24	75,73
SHA1	24xrockyou.txt	4	6000000	338,95	262,33
SHA1	rockyou.txt	8	500000	19,57	105,22
SHA1	6xrockyou.txt	8	500000	101,10	528,86
SHA1	24xrockyou.txt	8	500000	348,61	2033,21
SHA1	rockyou.txt	8	1000000	18,31	69,95
SHA1	6xrockyou.txt	8	1000000	86,39	281,70
SHA1	24xrockyou.txt	8	1000000	311,31	1057,19
SHA1	rockyou.txt	8	3000000	21,02	47,68
SHA1	6xrockyou.txt	8	3000000	94,14	147,72
SHA1	24xrockyou.txt	8	3000000	337,42	512,65
SHA1	rockyou.txt	8	6000000	20,59	35,90
SHA1	6xrockyou.txt	8	6000000	86,10	88,01
SHA1	24xrockyou.txt	8	6000000	348,41	272,26
SHA1	rockyou.txt	16	500000	19,43	184,89
SHA1	6xrockyou.txt	16	500000	75,97	543,13
SHA1	24xrockyou.txt	16	500000	308,33	2045,56
SHA1	rockyou.txt	16	1000000	17,82	98,76
SHA1	6xrockyou.txt	16	1000000	79,62	302,67
SHA1	24xrockyou.txt	16	1000000	299,89	1087,64
SHA1	rockyou.txt	16	3000000	19,63	76,19
SHA1	6xrockyou.txt	16	3000000	92,53	169,91
SHA1	24xrockyou.txt	16	3000000	332,11	538,45
SHA1	rockyou.txt	16	6000000	18,61	68,70
SHA1	6xrockyou.txt	16	6000000	84,52	116,19
SHA1	24xrockyou.txt	16	6000000	343,33	297,94
Whirlpool	rockyou.txt	16	500000	17,73	148,16

Whirlpool	6xrockyou.txt	16	500000	76,45	567,92
Whirlpool	24xrockyou.txt	16	500000	288,02	2160,73
Whirlpool	rockyou.txt	16	1000000	17,94	98,13
Whirlpool	6xrockyou.txt	16	1000000	78,05	321,99
Whirlpool	24xrockyou.txt	16	1000000	293,36	1141,96
Whirlpool	rockyou.txt	16	3000000	19,79	72,59
Whirlpool	6xrockyou.txt	16	3000000	85,07	183,44
Whirlpool	24xrockyou.txt	16	3000000	318,02	560,72
Whirlpool	rockyou.txt	16	6000000	18,75	71,23
Whirlpool	6xrockyou.txt	16	6000000	84,00	116,47
Whirlpool	24xrockyou.txt	16	6000000	316,32	310,38

B.1.3 Slovníkový útok so zdieľanou pamäťou

Typ hashu	Slovník	Počet uzlov	Veľkosť úlohy [P]	Čas [s]	Čas spolu [s]
MD5	rockyou.txt	2	500000	56,45	106,61
MD5	6xrockyou.txt	2	500000	383,47	750,21
MD5	24xrockyou.txt	2	500000	2524,96	4988,02
MD5	rockyou.txt	2	1000000	30,88	57,86
MD5	6xrockyou.txt	2	1000000	210,34	401,56
MD5	24xrockyou.txt	2	1000000	1337,87	2605,35
MD5	rockyou.txt	2	3000000	18,71	31,07
MD5	6xrockyou.txt	2	3000000	89,49	161,87
MD5	24xrockyou.txt	2	3000000	510,90	953,47
MD5	rockyou.txt	2	6000000	13,96	23,08
MD5	6xrockyou.txt	2	6000000	60,80	103,13
MD5	24xrockyou.txt	2	6000000	308,89	551,86
MD5	rockyou.txt	4	500000	33,04	121,13
MD5	6xrockyou.txt	4	500000	213,61	812,21
MD5	24xrockyou.txt	4	500000	1323,26	5147,65
MD5	rockyou.txt	4	1000000	21,29	72,53
MD5	6xrockyou.txt	4	1000000	126,01	462,12
MD5	24xrockyou.txt	4	1000000	728,25	2782,54
MD5	rockyou.txt	4	3000000	14,40	44,14
MD5	6xrockyou.txt	4	3000000	65,16	218,01
MD5	24xrockyou.txt	4	3000000	323,75	1162,07
MD5	rockyou.txt	4	6000000	11,94	36,78
MD5	6xrockyou.txt	4	6000000	49,36	160,56
MD5	24xrockyou.txt	4	6000000	224,9	766,39
MD5	rockyou.txt	8	500000	24,25	168,01
MD5	6xrockyou.txt	8	500000	143,29	1066,97
MD5	24xrockyou.txt	8	500000	806,92	6180,42
MD5	rockyou.txt	8	1000000	19,17	127,84
MD5	6xrockyou.txt	8	1000000	103,13	741,85
MD5	24xrockyou.txt	8	1000000	520,15	3873,81
MD5	rockyou.txt	8	3000000	16,25	102,36

MD5	6xrockyou.txt	8	3000000	75,2	514,34
MD5	24xrockyou.txt	8	3000000	324,11	2322,71
MD5	rockyou.txt	8	6000000	15,05	94,57
MD5	6xrockyou.txt	8	6000000	67,30	460,41
MD5	24xrockyou.txt	8	6000000	275,19	1929,43
MD5	rockyou.txt	16	500000	27,33	390,72
MD5	6xrockyou.txt	16	500000	151,88	2270,14
MD5	24xrockyou.txt	16	500000	722,81	10996,10
MD5	rockyou.txt	16	1000000	24,94	349,77
MD5	6xrockyou.txt	16	1000000	133,79	1958,46
MD5	24xrockyou.txt	16	1000000	574,53	8613,96
MD5	rockyou.txt	16	3000000	23,18	327,82
MD5	6xrockyou.txt	16	3000000	119,77	1739,78
MD5	24xrockyou.txt	16	3000000	478,22	7085,22
MD5	rockyou.txt	16	6000000	23,11	320,94
MD5	6xrockyou.txt	16	6000000	118,92	1660,08
MD5	24xrockyou.txt	16	6000000	465,36	6688,28
SHA1	rockyou.txt	2	500000	54,60	103,43
SHA1	6xrockyou.txt	2	500000	380,37	743,99
SHA1	24xrockyou.txt	2	500000	2509,52	4949,16
SHA1	rockyou.txt	2	1000000	30,66	58,18
SHA1	6xrockyou.txt	2	1000000	206,37	393,12
SHA1	24xrockyou.txt	2	1000000	1318,36	2576,04
SHA1	rockyou.txt	2	3000000	18,10	30,84
SHA1	6xrockyou.txt	2	3000000	89,81	160,84
SHA1	24xrockyou.txt	2	3000000	509,88	951,49
SHA1	rockyou.txt	2	6000000	13,97	22,68
SHA1	6xrockyou.txt	2	6000000	61,10	104,77
SHA1	24xrockyou.txt	2	6000000	309,26	557,70
SHA1	rockyou.txt	4	500000	31,99	116,87
SHA1	6xrockyou.txt	4	500000	204,09	777,14
SHA1	24xrockyou.txt	4	500000	1286,99	5016,28
SHA1	rockyou.txt	4	1000000	20,69	72,45
SHA1	6xrockyou.txt	4	1000000	120,50	441,58
SHA1	24xrockyou.txt	4	1000000	716,33	2727,19
SHA1	rockyou.txt	4	3000000	13,98	45,32
SHA1	6xrockyou.txt	4	3000000	64,15	214,90
SHA1	24xrockyou.txt	4	3000000	321,19	1142,09
SHA1	rockyou.txt	4	6000000	12,02	37,28
SHA1	6xrockyou.txt	4	6000000	49,55	161,01
SHA1	24xrockyou.txt	4	6000000	221,57	757,78
SHA1	rockyou.txt	8	500000	24,53	170,52
SHA1	6xrockyou.txt	8	500000	144,57	1077,89
SHA1	24xrockyou.txt	8	500000	810,11	6189,12
SHA1	rockyou.txt	8	1000000	19,82	129,92

SHA1	6xrockyou.txt	8	1000000	102,04	740,03
SHA1	24xrockyou.txt	8	1000000	523,49	3894,39
SHA1	rockyou.txt	8	3000000	17,45	105,2
SHA1	6xrockyou.txt	8	3000000	76,06	522,42
SHA1	24xrockyou.txt	8	3000000	324,74	2319,57
SHA1	rockyou.txt	8	6000000	15,15	96,05
SHA1	6xrockyou.txt	8	6000000	67,49	462,69
SHA1	24xrockyou.txt	8	6000000	277,24	1936,52
SHA1	rockyou.txt	16	500000	28,80	401,85
SHA1	6xrockyou.txt	16	500000	153,54	2284,17
SHA1	24xrockyou.txt	16	500000	725,36	11038,15
SHA1	rockyou.txt	16	1000000	24,77	352,24
SHA1	6xrockyou.txt	16	1000000	132,85	1943,15
SHA1	24xrockyou.txt	16	1000000	575,54	8629,56
SHA1	rockyou.txt	16	3000000	23,22	330,24
SHA1	6xrockyou.txt	16	3000000	120,03	1744,08
SHA1	24xrockyou.txt	16	3000000	479,81	7096,11
SHA1	rockyou.txt	16	6000000	23,32	323,55
SHA1	6xrockyou.txt	16	6000000	119,80	1677,11
SHA1	24xrockyou.txt	16	6000000	466,00	6763,55
Whirlpool	rockyou.txt	16	500000	28,73	409,43
Whirlpool	6xrockyou.txt	16	500000	157,47	2364,95
Whirlpool	24xrockyou.txt	16	500000	764,56	11286,65
Whirlpool	rockyou.txt	16	1000000	25,56	364,41
Whirlpool	6xrockyou.txt	16	1000000	137,35	2034,51
Whirlpool	24xrockyou.txt	16	1000000	592,35	8905,43
Whirlpool	rockyou.txt	16	3000000	24,52	333,18
Whirlpool	6xrockyou.txt	16	3000000	121,81	1784,94
Whirlpool	24xrockyou.txt	16	3000000	481,96	7143,76
Whirlpool	rockyou.txt	16	6000000	23,28	326,35
Whirlpool	6xrockyou.txt	16	6000000	119,84	1699,26
Whirlpool	24xrockyou.txt	16	6000000	464,65	6778,03

B.2 Hashtopolis

B.2.1 Maskový útok

Typ hashu	Počet znakov	Počet uzlov	Veľkosť úlohy [s]	Čas [s]	Čas spolu [s]
MD5	7	2	60	17	7
MD5	8	2	60	73	94
MD5	9	2	60	1251	2427
MD5	7	2	120	14	7
MD5	8	2	120	101	91
MD5	9	2	120	1251	2359
MD5	7	2	600	18	6
MD5	8	2	600	102	90

MD5	9	2	600	1236	2307
MD5	7	2	1800	18	7
MD5	8	2	1800	104	92
MD5	9	2	1800	1840	2298
MD5	7	4	60	15	6
MD5	8	4	60	76	95
MD5	9	4	60	660	2437
MD5	7	4	120	15	7
MD5	8	4	120	100	91
MD5	9	4	120	637	2370
MD5	7	4	600	16	6
MD5	8	4	600	100	91
MD5	9	4	600	627	2320
MD5	7	4	1800	17	7
MD5	8	4	1800	100	89
MD5	9	4	1800	1841	2319
MD5	7	8	60	15	7
MD5	8	8	60	72	97
MD5	9	8	60	348	2469
MD5	7	8	120	16	8
MD5	8	8	120	102	93
MD5	9	8	120	388	2396
MD5	7	8	600	14	6
MD5	8	8	600	103	94
MD5	9	8	600	649	2385
MD5	7	8	1800	15	7
MD5	8	8	1800	102	93
MD5	9	8	1800	1920	2402
MD5	7	16	60	14	7
MD5	8	16	60	74	95
MD5	9	16	60	205	2452
MD5	7	16	120	14	7
MD5	8	16	120	100	92
MD5	9	16	120	260	2380
MD5	7	16	600	14	6
MD5	8	16	600	98	90
MD5	9	16	600	647	2339
MD5	7	16	1800	15	7
MD5	8	16	1800	98	89
MD5	9	16	1800	1933	2368
SHA1	7	2	60	20	9
SHA1	8	2	60	90	140
SHA1	9	2	60	1935	3766
SHA1	7	2	120	19	9
SHA1	8	2	120	150	140

SHA1	9	2	120	1917	3677
SHA1	7	2	600	18	9
SHA1	8	2	600	152	140
SHA1	9	2	600	1873	3609
SHA1	7	2	1800	18	9
SHA1	8	2	1800	150	137
SHA1	9	2	1800	1890	3573
SHA1	7	4	60	18	8
SHA1	8	4	60	85	142
SHA1	9	4	60	1001	3769
SHA1	7	4	120	19	9
SHA1	8	4	120	149	139
SHA1	9	4	120	1025	3666
SHA1	7	4	600	19	9
SHA1	8	4	600	148	139
SHA1	9	4	600	1236	3583
SHA1	7	4	1800	21	9
SHA1	8	4	1800	148	139
SHA1	9	4	1800	1860	3574
SHA1	7	8	60	17	9
SHA1	8	8	60	88	143
SHA1	9	8	60	534	3793
SHA1	7	8	120	17	8
SHA1	8	8	120	147	139
SHA1	9	8	120	522	3687
SHA1	7	8	600	20	9
SHA1	8	8	600	151	143
SHA1	9	8	600	652	3605
SHA1	7	8	1800	17	8
SHA1	8	8	1800	147	136
SHA1	9	8	1800	1852	3629
SHA1	7	16	60	17	8
SHA1	8	16	60	85	140
SHA1	9	16	60	275	3762
SHA1	7	16	120	17	9
SHA1	8	16	120	148	139
SHA1	9	16	120	272	3663
SHA1	7	16	600	17	9
SHA1	8	16	600	146	138
SHA1	9	16	600	651	3616
SHA1	7	16	1800	17	9
SHA1	8	16	1800	145	137
SHA1	9	16	1800	1864	3565
Whirlpool	7	16	60	80	137
Whirlpool	8	16	60	281	3587

Whirlpool	9	16	60	6013	94118
Whirlpool	7	16	120	142	133
Whirlpool	8	16	120	270	3490
Whirlpool	9	16	120	5810	91231
Whirlpool	7	16	600	146	136
Whirlpool	8	16	600	641	3402
Whirlpool	9	16	600	5805	88775
Whirlpool	7	16	1800	146	138
Whirlpool	8	16	1800	1864	3414
Whirlpool	9	16	1800	5850	88262

B.2.2 Slovníkový útok

Typ hashu	Slovník	Počet uzlov	Veľkosť úlohy [P]	Čas [s]	Čas spolu [s]
MD5	rockyou.txt	2	60	26	16
MD5	6xrockyou.txt	2	60	98	50
MD5	24xrockyou.txt	2	60	344	268
MD5	rockyou.txt	2	120	21	8
MD5	6xrockyou.txt	2	120	83	32
MD5	24xrockyou.txt	2	120	285	167
MD5	rockyou.txt	2	600	19	5
MD5	6xrockyou.txt	2	600	72	20
MD5	24xrockyou.txt	2	600	249	85
MD5	rockyou.txt	2	1800	22	5
MD5	6xrockyou.txt	2	1800	78	20
MD5	24xrockyou.txt	2	1800	259	72
MD5	rockyou.txt	4	60	21	7
MD5	6xrockyou.txt	4	60	83	54
MD5	24xrockyou.txt	4	60	332	320
MD5	rockyou.txt	4	120	15	4
MD5	6xrockyou.txt	4	120	86	37
MD5	24xrockyou.txt	4	120	297	197
MD5	rockyou.txt	4	600	17	5
MD5	6xrockyou.txt	4	600	87	20
MD5	24xrockyou.txt	4	600	267	83
MD5	rockyou.txt	4	1800	17	4
MD5	6xrockyou.txt	4	1800	78	11
MD5	24xrockyou.txt	4	1800	305	72
MD5	rockyou.txt	8	60	20	8
MD5	6xrockyou.txt	8	60	92	58
MD5	24xrockyou.txt	8	60	385	314
MD5	rockyou.txt	8	120	20	6
MD5	6xrockyou.txt	8	120	88	41
MD5	24xrockyou.txt	8	120	359	193
MD5	rockyou.txt	8	600	20	4
MD5	6xrockyou.txt	8	600	102	20

MD5	24xrockyou.txt	8	600	308	86
MD5	rockyou.txt	8	1800	16	4
MD5	6xrockyou.txt	8	1800	90	20
MD5	24xrockyou.txt	8	1800	345	72
MD5	rockyou.txt	16	60	18	12
MD5	6xrockyou.txt	16	60	88	54
MD5	24xrockyou.txt	16	60	475	300
MD5	rockyou.txt	16	120	20	10
MD5	6xrockyou.txt	16	120	109	36
MD5	24xrockyou.txt	16	120	411	190
MD5	rockyou.txt	16	600	26	5
MD5	6xrockyou.txt	16	600	127	20
MD5	24xrockyou.txt	16	600	219	71
MD5	rockyou.txt	16	1800	25	5
MD5	6xrockyou.txt	16	1800	109	12
MD5	24xrockyou.txt	16	1800	235	71
SHA1	rockyou.txt	2	60	23	15
SHA1	6xrockyou.txt	2	60	90	47
SHA1	24xrockyou.txt	2	60	353	270
SHA1	rockyou.txt	2	120	21	10
SHA1	6xrockyou.txt	2	120	82	34
SHA1	24xrockyou.txt	2	120	288	172
SHA1	rockyou.txt	2	600	18	6
SHA1	6xrockyou.txt	2	600	80	21
SHA1	24xrockyou.txt	2	600	242	84
SHA1	rockyou.txt	2	1800	18	6
SHA1	6xrockyou.txt	2	1800	77	20
SHA1	24xrockyou.txt	2	1800	250	72
SHA1	rockyou.txt	4	60	21	10
SHA1	6xrockyou.txt	4	60	90	56
SHA1	24xrockyou.txt	4	60	339	318
SHA1	rockyou.txt	4	120	19	6
SHA1	6xrockyou.txt	4	120	87	35
SHA1	24xrockyou.txt	4	120	296	197
SHA1	rockyou.txt	4	600	18	4
SHA1	6xrockyou.txt	4	600	90	21
SHA1	24xrockyou.txt	4	600	272	80
SHA1	rockyou.txt	4	1800	17	4
SHA1	6xrockyou.txt	4	1800	86	20
SHA1	24xrockyou.txt	4	1800	305	72
SHA1	rockyou.txt	8	60	20	8
SHA1	6xrockyou.txt	8	60	94	77
SHA1	24xrockyou.txt	8	60	397	313
SHA1	rockyou.txt	8	120	19	10
SHA1	6xrockyou.txt	8	120	91	32

SHA1	24xrockyou.txt	8	120	353	173
SHA1	rockyou.txt	8	600	20	6
SHA1	6xrockyou.txt	8	600	96	20
SHA1	24xrockyou.txt	8	600	312	87
SHA1	rockyou.txt	8	1800	19	6
SHA1	6xrockyou.txt	8	1800	96	20
SHA1	24xrockyou.txt	8	1800	371	72
SHA1	rockyou.txt	16	60	28	14
SHA1	6xrockyou.txt	16	60	122	67
SHA1	24xrockyou.txt	16	60	496	309
SHA1	rockyou.txt	16	120	23	5
SHA1	6xrockyou.txt	16	120	119	37
SHA1	24xrockyou.txt	16	120	420	170
SHA1	rockyou.txt	16	600	24	6
SHA1	6xrockyou.txt	16	600	118	20
SHA1	24xrockyou.txt	16	600	453	87
SHA1	rockyou.txt	16	1800	22	6
SHA1	6xrockyou.txt	16	1800	99	12
SHA1	24xrockyou.txt	16	1800	441	72
Whirlpool	rockyou.txt	16	60	24	18
Whirlpool	6xrockyou.txt	16	60	122	128
Whirlpool	24xrockyou.txt	16	60	521	722
Whirlpool	rockyou.txt	16	120	21	11
Whirlpool	6xrockyou.txt	16	120	111	95
Whirlpool	24xrockyou.txt	16	120	472	414
Whirlpool	rockyou.txt	16	600	27	7
Whirlpool	6xrockyou.txt	16	600	137	36
Whirlpool	24xrockyou.txt	16	600	272	139
Whirlpool	rockyou.txt	16	1800	22	7
Whirlpool	6xrockyou.txt	16	1800	123	32
Whirlpool	24xrockyou.txt	16	1800	396	129

B.3 Fitcrack

B.3.1 Maskový útok

Typ hashu	Počet znakov	Počet uzlov	Veľkosť úlohy [s]	Čas [s]	Čas spolu [s]
MD5	9	2	60	1728,00	2532,59
MD5	8	2	60	133,00	113,75
MD5	7	2	60	85,00	24,00
MD5	9	4	60	896,00	2541,38
MD5	8	4	60	184,00	129,15
MD5	7	4	60	82,00	40,44
MD5	9	8	60	511,00	2577,01
MD5	8	8	60	129,00	161,79
MD5	7	8	60	88,00	73,07

MD5	9	16	60	300,00	2629,89
MD5	8	16	60	125,00	226,73
MD5	7	16	60	85,00	138,06
MD5	9	2	120	1633,00	2494,27
MD5	8	2	120	120,00	112,59
MD5	7	2	120	69,00	24,00
MD5	9	4	120	910,00	2522,42
MD5	8	4	120	123,00	129,05
MD5	7	4	120	82,00	40,23
MD5	9	8	120	498,00	2573,51
MD5	8	8	120	132,00	162,88
MD5	7	8	120	76,00	73,13
MD5	9	16	120	303,00	2631,19
MD5	8	16	120	128,00	226,94
MD5	7	16	120	79,00	138,48
MD5	9	2	600	1605,00	2491,95
MD5	8	2	600	193,00	111,67
MD5	7	2	600	78,00	23,91
MD5	9	4	600	839,00	2518,89
MD5	8	4	600	140,00	128,85
MD5	7	4	600	95,00	40,37
MD5	9	8	600	491,00	2561,91
MD5	8	8	600	118,00	161,60
MD5	7	8	600	69,00	72,91
MD5	9	16	600	305,00	2625,26
MD5	8	16	600	133,00	227,08
MD5	7	16	600	79,00	137,77
MD5	9	2	1800	1631,00	2500,09
MD5	8	2	1800	131,00	112,86
MD5	7	2	1800	96,00	23,91
MD5	9	4	1800	888,00	2532,20
MD5	8	4	1800	173,00	128,91
MD5	7	4	1800	87,00	40,36
MD5	9	8	1800	485,00	2560,52
MD5	8	8	1800	173,00	161,72
MD5	7	8	1800	80,00	72,78
MD5	9	16	1800	296,00	2634,00
MD5	8	16	1800	126,00	226,94
MD5	7	16	1800	73,00	138,07
SHA-1	9	2	60	2643,00	3892,58
SHA-1	8	2	60	185,00	174,70
SHA-1	7	2	60	94,00	26,24
SHA-1	9	4	60	1381,00	3909,28
SHA-1	8	4	60	171,00	191,20
SHA-1	7	4	60	81,00	42,59

SHA-1	9	8	60	772,00	3967,14
SHA-1	8	8	60	130,00	223,83
SHA-1	7	8	60	80,00	75,22
SHA-1	9	16	60	436,00	4052,77
SHA-1	8	16	60	169,00	288,27
SHA-1	7	16	60	79,00	140,74
SHA-1	9	2	120	2363,00	3808,55
SHA-1	8	2	120	180,00	175,95
SHA-1	7	2	120	90,00	26,26
SHA-1	9	4	120	1318,00	3875,29
SHA-1	8	4	120	129,00	191,36
SHA-1	7	4	120	81,00	42,68
SHA-1	9	8	120	706,00	3948,52
SHA-1	8	8	120	166,00	223,82
SHA-1	7	8	120	95,00	75,29
SHA-1	9	16	120	417,00	4056,89
SHA-1	8	16	120	122,00	289,40
SHA-1	7	16	120	99,00	140,85
SHA-1	9	2	600	2340,00	3798,16
SHA-1	8	2	600	199,00	175,75
SHA-1	7	2	600	91,00	26,30
SHA-1	9	4	600	1300,00	3870,01
SHA-1	8	4	600	170,00	190,19
SHA-1	7	4	600	92,00	42,72
SHA-1	9	8	600	761,00	3968,67
SHA-1	8	8	600	166,00	224,12
SHA-1	7	8	600	85,00	75,29
SHA-1	9	16	600	474,00	4170,43
SHA-1	8	16	600	170,00	290,25
SHA-1	7	16	600	81,00	140,88
SHA-1	9	2	1800	2323,00	3786,53
SHA-1	8	2	1800	192,00	175,94
SHA-1	7	2	1800	97,00	26,37
SHA-1	9	4	1800	1410,00	3965,66
SHA-1	8	4	1800	171,00	191,23
SHA-1	7	4	1800	80,00	42,77
SHA-1	9	8	1800	840,00	4089,25
SHA-1	8	8	1800	122,00	225,06
SHA-1	7	8	1800	81,00	75,49
SHA-1	9	16	1800	470,00	4162,92
SHA-1	8	16	1800	123,00	289,53
SHA-1	7	16	1800	95,00	141,03
Whirlpool	9	16	60	8179,00	98411,15
Whirlpool	8	16	60	449,00	4085,80
Whirlpool	7	16	60	134,00	321,63

Whirlpool	9	16	120	6876,00	93759,98
Whirlpool	8	16	120	434,00	4089,37
Whirlpool	7	16	120	113,00	323,25
Whirlpool	9	16	600	6357,00	91493,98
Whirlpool	8	16	600	498,00	4429,33
Whirlpool	7	16	600	132,00	322,30
Whirlpool	9	16	1800	6355,00	91551,74
Whirlpool	8	16	1800	508,00	4565,63
Whirlpool	7	16	1800	153,00	323,14

B.3.2 Slovníkový útok

Typ hashu	Slovník	Počet uzlov	Veľkosť úlohy [P]	Čas [s]	Čas spolu [s]
MD5	rockyou.txt	2	60	26,00	16,00
MD5	24xrockyou.txt	2	60	250,00	141,91
MD5	6xrockyou.txt	2	60	128,00	51,53
MD5	rockyou.txt	2	60	83,00	24,95
MD5	24xrockyou.txt	2	120	228,00	139,70
MD5	6xrockyou.txt	2	120	139,00	51,41
MD5	rockyou.txt	2	120	92,00	24,94
MD5	24xrockyou.txt	2	600	203,00	143,20
MD5	6xrockyou.txt	2	600	101,00	51,33
MD5	rockyou.txt	2	600	81,00	24,83
MD5	24xrockyou.txt	2	1800	195,00	145,03
MD5	6xrockyou.txt	2	1800	154,00	51,24
MD5	rockyou.txt	2	1800	88,00	24,94
MD5	24xrockyou.txt	4	60	175,00	161,32
MD5	6xrockyou.txt	4	60	109,00	66,66
MD5	rockyou.txt	4	60	80,00	41,22
MD5	24xrockyou.txt	4	120	160,00	161,67
MD5	6xrockyou.txt	4	120	139,00	67,78
MD5	rockyou.txt	4	120	89,00	41,29
MD5	24xrockyou.txt	4	600	160,00	161,62
MD5	6xrockyou.txt	4	600	123,00	67,75
MD5	rockyou.txt	4	600	69,00	41,23
MD5	24xrockyou.txt	4	1800	165,00	161,36
MD5	6xrockyou.txt	4	1800	124,00	67,58
MD5	rockyou.txt	4	1800	90,00	41,26
MD5	24xrockyou.txt	8	60	146,00	195,34
MD5	6xrockyou.txt	8	60	116,00	100,33
MD5	rockyou.txt	8	60	85,00	74,13
MD5	24xrockyou.txt	8	120	154,00	195,41
MD5	6xrockyou.txt	8	120	110,00	100,74
MD5	rockyou.txt	8	120	92,00	74,39
MD5	24xrockyou.txt	8	600	140,00	195,37
MD5	6xrockyou.txt	8	600	125,00	99,40

MD5	rockyou.txt	8	600	89,00	74,26
MD5	24xrockyou.txt	8	1800	140,00	195,40
MD5	6xrockyou.txt	8	1800	110,00	100,63
MD5	rockyou.txt	8	1800	84,00	74,07
MD5	24xrockyou.txt	16	60	144,00	233,42
MD5	6xrockyou.txt	16	60	115,00	166,27
MD5	rockyou.txt	16	60	88,00	139,90
MD5	24xrockyou.txt	16	120	149,00	260,00
MD5	6xrockyou.txt	16	120	105,00	165,02
MD5	rockyou.txt	16	120	93,00	139,70
MD5	24xrockyou.txt	16	600	136,00	260,88
MD5	6xrockyou.txt	16	600	143,00	166,43
MD5	rockyou.txt	16	600	93,00	139,49
MD5	6xrockyou.txt	16	1800	105,00	166,48
MD5	rockyou.txt	16	1800	91,00	139,79
MD5	24xrockyou.txt	16	1800	146,00	260,83
SHA-1	24xrockyou.txt	2	60	242,00	143,23
SHA-1	6xrockyou.txt	2	60	161,00	52,07
SHA-1	rockyou.txt	2	60	101,00	25,26
SHA-1	24xrockyou.txt	2	120	226,00	141,99
SHA-1	6xrockyou.txt	2	120	136,00	51,98
SHA-1	rockyou.txt	2	120	85,00	25,12
SHA-1	24xrockyou.txt	2	600	204,00	146,83
SHA-1	6xrockyou.txt	2	600	146,00	52,01
SHA-1	rockyou.txt	2	600	96,00	25,20
SHA-1	24xrockyou.txt	2	1800	215,00	146,77
SHA-1	6xrockyou.txt	2	1800	119,00	52,05
SHA-1	rockyou.txt	2	1800	83,00	25,07
SHA-1	24xrockyou.txt	4	60	175,00	164,29
SHA-1	6xrockyou.txt	4	60	113,00	68,44
SHA-1	rockyou.txt	4	60	91,00	41,77
SHA-1	24xrockyou.txt	4	120	163,00	163,41
SHA-1	6xrockyou.txt	4	120	125,00	68,39
SHA-1	rockyou.txt	4	120	85,00	41,67
SHA-1	24xrockyou.txt	4	600	166,00	131,25
SHA-1	6xrockyou.txt	4	600	124,00	68,30
SHA-1	rockyou.txt	4	600	83,00	41,80
SHA-1	24xrockyou.txt	4	1800	185,00	163,30
SHA-1	6xrockyou.txt	4	1800	114,00	68,41
SHA-1	rockyou.txt	4	1800	85,00	41,87
SHA-1	24xrockyou.txt	8	60	144,00	197,38
SHA-1	6xrockyou.txt	8	60	122,00	101,66
SHA-1	rockyou.txt	8	60	94,00	74,49
SHA-1	24xrockyou.txt	8	120	144,00	197,32
SHA-1	6xrockyou.txt	8	120	123,00	101,52

SHA-1	rockyou.txt	8	120	78,00	74,59
SHA-1	24xrockyou.txt	8	600	153,00	195,41
SHA-1	6xrockyou.txt	8	600	103,00	101,18
SHA-1	rockyou.txt	8	600	78,00	74,54
SHA-1	24xrockyou.txt	8	1800	148,00	197,38
SHA-1	6xrockyou.txt	8	1800	124,00	101,27
SHA-1	rockyou.txt	8	1800	84,00	74,36
SHA-1	24xrockyou.txt	16	60	137,00	262,92
SHA-1	6xrockyou.txt	16	60	121,00	166,96
SHA-1	rockyou.txt	16	60	82,00	140,19
SHA-1	24xrockyou.txt	16	120	153,00	263,40
SHA-1	6xrockyou.txt	16	120	125,00	167,15
SHA-1	rockyou.txt	16	120	85,00	140,46
SHA-1	24xrockyou.txt	16	600	134,00	263,29
SHA-1	6xrockyou.txt	16	600	103,00	166,11
SHA-1	rockyou.txt	16	600	84,00	140,61
SHA-1	24xrockyou.txt	16	1800	155,00	263,26
SHA-1	6xrockyou.txt	16	1800	118,00	167,23
SHA-1	rockyou.txt	16	1800	84,00	140,51
Whirlpool	24xrockyou.txt	16	60	154,00	362,45
Whirlpool	6xrockyou.txt	16	60	118,00	205,92
Whirlpool	rockyou.txt	16	60	82,00	155,04
Whirlpool	24xrockyou.txt	16	120	165,00	361,94
Whirlpool	6xrockyou.txt	16	120	124,00	199,37
Whirlpool	rockyou.txt	16	120	92,00	155,44
Whirlpool	24xrockyou.txt	16	600	167,00	361,79
Whirlpool	6xrockyou.txt	16	600	134,00	204,75
Whirlpool	rockyou.txt	16	600	106,00	158,50
Whirlpool	24xrockyou.txt	16	1800	175,00	371,17
Whirlpool	6xrockyou.txt	16	1800	127,00	205,79
Whirlpool	rockyou.txt	16	1800	95,00	158,65