Distribuce výpočtů pro nástroj hashcat

Skupina Zabezpečná data

Technická zpráva FIT VUT v Brně

Radek Hranický, Lukáš Zobal, Vojtěch Večeřa, Matúš Múčka

Technická zpráva č. FIT-TR-2018-04
Fakulta informačních technologií, Vysoké učení technické v Brně

Poslední úprava: 29. prosince 2018

Distribuce výpočtů pro nástroj hashcat

Radek Hranický, Lukáš Zobal, Vojtěch Večeřa, Matúš Múčka

Vysoké učení technické v Brně, email:
{ihranicky,izobal}@fit.vutbr.cz, {xvecer18,xmucka03}@stud.fit.vutbr.cz

Abstrakt Tato souhrnná technická zpráva popisuje techniky distribuce výpočtu, které byly implementovány do systému Fitcrack v rámci projektu *Integrovaná platforma pro zpracování digitálních dat z bezpečnostních incidentů*. Cílem těchto technik je provoz nástroje hashcat na výpočetních uzlech a jejich kooperace při úlohách lámání hesel. Technická zpráva dále zahrnuje dvě případové studie. První studie se popisuje útok na bezdrátovou síť zapezbečenou pomocí WPA a WPA2. Druhá studie se věnuje distribuovanému slovníkovému útoku.

1 Úvod

V rámci předchozího výzkumu [8, 7, 11, 6] distribuovaných metod v oblasti forenzní analýzy digitálních ve skupině *NES@FIT*[1] byl vytvořen prototyp systému *Fitcrack*[2]. Tento slouží k distribuovanému lámání kryptografických hešů a hledání hesel pro přístup k různým formátům šifrovaných médií, jako jsou dokumenty Office [19, 20], PDF [2], či archivy Zip, 7z [17] a RAR [13].

Původní verze systému Fitcrack využívala své vlastní implementace algoritmů pro výpočet a ověřování kryptografických hešů, která vycházela z nástroje Wrathion[3]. Jednotlivé algoritmy byly implementovány ve třech variantách: 1) pro výpočet na CPU, 2) pro akcelerovaný výpočet na principu GPGPU (*General-purpose computing on graphics processing units*) [9, 12] s využitím OpenCL[4], 3) totéž s využivím NVIDIA CUDA[5]. Pro výpočet na více uzlech v počítačové sítí Fitcrack využívá platformy BOINC (*Berkeley Open Infrastructure for Network Computing*)[6], přičemž samotné úlohy jsou rozdělovány na principu distribuce *indexů hesel* [10]. Pro efektivní využití výpočetních kapacit byl navržen a implementován adaptivní algoritmus pro plánování úloh, který se vyrovná i s výkyvy výkonu jednotlivých výpočetních stanic [7].

Pro účely dosažení maximálního výkonu byl na klientské straně namísto původního řešení nasazen nástroj *hashcat*[7], čemuž byly přizpůsobeny zbylé části

[1] http://www.fit.vutbr.cz/research/groups/nes@fit/.cs
[2] http://fitcrack.fit.vutbr.cz
[3] https://wrathion.fit.vutbr.cz/
[4] https://www.khronos.org/opencl/
[5] https://www.geforce.com/hardware/technology/cuda
[6] https://boinc.berkeley.edu
[7] https://hashcat.net/

systému [10]. Pro plné využití možností nástroje hashcat však bylo potřeba zvolit odpovídající strategie, jak výpočetní úlohy distribuovat.

Tato technická zpráva se proto zaměřuje na problematiku jednotlivých útoků, které nástroj hashcat podporuje, diskutuje možnosti jejich distribuce včetně konkrétního řešení, které bylo použito v systému Fitcrack.

Součástí zprávy jsou také dvě případové studie. První z nich ukazuje možnosti prolomení zabezpečení bezdrátové sítě využívající WPA a WPA2. Studie demonstruje, jak pomocí nástrojů *Aircrack-ng*, popř. *Hxcdumptool* zachytit část síťové komunikace potřebné pro realizaci útoku. Následně pak ukazuje, jak je možné na základě této komunikace nalézt heslo k síti s využitím nástroje hashcat na jednom stroji, a dále s využitím systému Fitcrack a distribuovaného výpočtu.

Druhá případová studie popisuje praktické aspekty distribučních strategií zvolených pro slovníkový útok a ukazuje jejich reálný dopad při distribuovaném útoku s velkými slovníky hesel. Studie srovnává řešení použité v systému Fitcrack s řešením, které využívá nástroj *Hashtopolis*[8].

V sekci 2 jsou popsány jednotlivé typy útoků u nástroje hashcat, jejich princip a příklady použití. Sekce 3 ukazuje, jak lze tyto útoky distribuovat na více uzlů. Další prostor je věnován případovým studiím. Sekce 4 se věnuje případové studii útoku na bezdrátovou síť zapezpečenou pocí WPA a WPA2. Sekce 5 obsahuje případovou studii distribuovaného slovníkového útoku.

2 Typy útoků u nástroje hashcat

Nástroj hashcat podporuje více způsobů, kterými je možné generovat hesla, ze kterých jsou počítány kryptografické heše. K verzi 4.2.1 jsou podporodány následující typy útoku:

- **slovníkový útok**,
- **kombinační útok**,
- **útok hrubou silou**,
- **hybridní útok s použitím slovníku a masky**,
- **hybridní útok s použitím masky a slovníku**.

Tyto útoky budou dále popsány. Časová i prostorová složitost jednotlivých útoků je přímo úměrná počtu zkoušených hesel (p), jehož výpočet bude u každého útoku ilustrován.

2.1 Slovníkový útok

Slovníkový útok (dictionary attack, wordlist attack), označován též jako přímý útok (straight attack), využívá textového souboru, ve kterém jsou uložena jednotlivá hesla. Každé heslo je uloženo na samostatném řádku. Při slovníkovém

[8] https://github.com/s3inlc/hashtopolis

útoku nástroj hashcat postupně čte jednotlivé řádky, z hesel počítá kryptografické heše a tyto porovánává s heši hledanými.

V praxi může slovník obsahovat reálná slova nějakého jazyka, nebo např. reálná hesla získaná z úniků[9] hesel různých internetových služeb. Jedním z nejznámějších takových slovníků je *rockyou.txt*, který vznikl na konci roku 2009 při krádeži[10] osobních údajů ze stejnojmenné sítě. Tento slovník obsahuje přibližně 15 milionů unikátních hesel.

Z matematického hlediska můžeme slovník hesel chápat jako uspořádanou množinu D, kde je uspořádání dáno pořadím hesel ve slovníku. Celkový počet hesel p je pak roven mohutnosti takové množiny:

$$p = |D|$$

2.1.1 Použití pravidel Slovníkový útok v nástroji hashcat může být také doplněn sadou speciálních *pravidel pro modifikaci hesel*. Tato pravidla definují nejrůznější způsoby úprav existujících slovníkových hesel jako jsou nahrazování, či záměna znaků a podřetězců, doplnění, oříznutí hesla, apod. Soubor pravidel zahrnuje všechna pravidla podporovaná nástrojem John the ripper a řadu dalších. Celkem je podporováno přes 70[11] druhů pravidel. Tabulka 1 ukazuje některá z nich.

Pravidlo	**Popis**	**Vstup**	**Výstup**
l	Zmenší všechny znaky A–Z	h3sLo	h3slo
C	Zvětší první písmeno a zmenší ostatní	h3sLo	H3slo
t	Změní velká písmena za malá a naopak	h3sLo	H3SlO
r	Převrátí pořadí znaků	h3sLo	oLs3h
]	Smaže poslední znak	h3sLo	h3sL
k	Prohodí první dva znaky	h3sLo	3hsLo

Tabulka 1: Ukázka několika hashcat pravidel

Při slovníkovém útoku jsou na každé heslo ze slovníku postupně aplikována všechna pravidla způsobem: na heslo aplikujeme první pravidlo, výsledek se použije; následně na původní heslo druhé pravidlo, atp. Je-li v souboru s pravidly na jednom řádku více pravidel, jsou před použitím hesla aplikována veškhna pravidla z řádku současně. Obsabuje li soubor s pravidly r řádků, celkový počet generovaných hesel p vypočteme následovně:

$$p = |D| * r$$

[9] https://wiki.skullsecurity.org/Passwords

[10] https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/

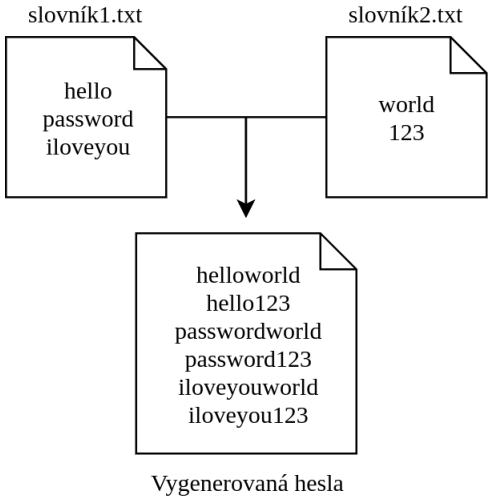
[11] https://hashcat.net/wiki/doku.php?id=rule_based_attack

2.2 Kombinační útok

Kombinační útok (combination attack, combinator attack) využívá dvou slovníků hesel: *levého* a *pravého* slovníku. Zkoušená hesla jsou tvořena konkatenací hesel z levého a pravého slovníku. Jednoduchý příklad je ukázán na obrázku 1.

Pro získání a použití vhodných slovníků zde platí stejné praktiky jako u slovníkového útoku. Nechť D_1 je levý a D_2 pravý slovník. Celkový počet zkoušených hesel p je pak dán součinem mohutností obou množin:

$$p = |D_1| \times |D_2|$$



Obrázek 1: Ukázka použití kombinačního útoku

2.3 Útok hrubou silou

Klasický útok hrubou silou (brute-force attack) spočívá v tvoření všech permutací dané délky nad danou abecedou (množinou znaků). Nástroj hashcat však tuto techniku rozšiřuje zavedením tzv. *masky*.

2.3.1 Maska hesla Maska umožňuje specifikovat podobu generovaných hesel, konkrétně udává, které znaky se mohou vyskytovat na kterých pozicích. Maska je definována jako posloupnost symbolů. Každý symbol v masce je buď:

- **konkrétní znak** z ASCII tabulky, nebo
- **zástupný symbol** množiny znaků.

Konkrétní znaky z masky jsou v generovaných heslech přímo použity na stejných pozicích, jako v masce. Na pozicích, kde se vyskytují zástupné symboly, hashcat postupně generuje všechny znaky z množiny dané příslušných zástupným symbolem. Rozlišovány jsou následující zástupné symboly:

- `?l` – zkratka pro *lower*, malá písmena latinky, tedy *a–z*,
- `?u` – zkratka pro *upper*, velká písmena latinky, tedy *A –Z*,
- `?d` – zkratka pro *digit*, arabské číslice, tedy *0 – 9*,
- `?s` – zkratka pro *special*, speciální znaky ASCII: mezera, interpunkce, aj.,
- `?a` – zkratka pro *all*, zahrnuje všechny předchozí možnosti, tedy `?l?u?d` a `?s`,
- `?h` – zkratka pro *hexa*, znaky šestnáctkové soustavy, tedy *0–9*, *a–f*,
- `?H` – zkratka pro *HEXA*, velké znaky šestnáctkové soustavy, tedy *0–9*, *A–F*,
- `?b` – zkratka pro *binary*, všechny možné hodnoty znaku, tedy `0x00` – `0xFF`,
- `?1` až `?4` – uživatelem definované množiny znaků.

Příklad použití masky ilustruje obrázek 2. Celkový počet hesel závisí pouze na uvedených zástupných symbolech. Konkrétní znaky uvedené v masce nemají na celkový počet hesel vliv. Nechť $G_1, G_2, ..., G_n$ jsou množiny reprezentované jednotlivými zástupnými symboly v masce. Pak celkový počet hesel získáme jako součin mohutností těchto množin:

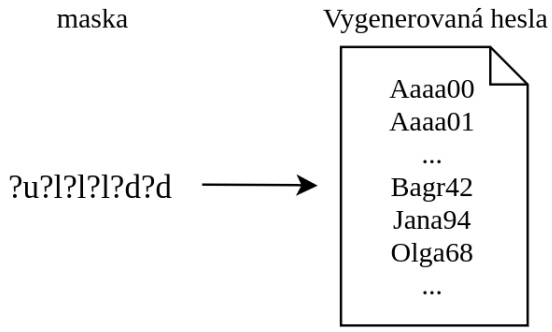
$$p = \prod_{i=1}^{n} |G_i|$$

Např. pro masku `Ahoj?u?d?d`, která generuje hesla `AhojA00` až `AhojZ99` platí:

$$\begin{aligned} G_1 &= \{A, B, ..., Z\} \\ G_2 = G_3 &= \{0, 1, ..., 9\} \\ |G_1| &= 26 \\ |G_2| = |G_3| &= 10 \\ p &= |G_1| \cdot |G_2| \cdot |G_3| \\ p &= 26 \cdot 10 \cdot 10 \\ p &= 2600 \end{aligned}$$

Zástupné symboly `?1` až `?4` lze použít v případě, kdy jsou při spouštění nástroje nastaveny parametry `-1` až `-4`. Tyto slouží k definování vlastních množin znaků [12], čímž rozšiřujeme původní možnosti útoku hrubou silou. Uživatelem definované znakové sady mohou stavět na standardních množinách znaků z ASCII tabulky, nicméně je lze použít též pro reprezentaci národních abeced v různých kódováních. Toto však platí pouze pro kódování s fixní délkou znaku 1B (např. *Windows-1250*). Kódování s délkou znaku vyšší než 1B (Např. *UTF-8*, *UTF-16*, apod.) je nutno řešit rozdělením částí znaků do vícero množin znaků, např.: pokud symbol `?1` zastupuje množinu bajtů s hodnotami `0xC3`, `0xC4` a `0xC5`, zatímco symbol `?2` množinu bajtů v rozsahu `0x81` až `0xBE`, bude maska `?1?2?1?2?1?2`

[12] https://hashcat.net/wiki/doku.php?id=mask_attack



Obrázek 2: Ukázka generování hesel pomocí masky

generovat různé tříznakové permutace českých a slovenských znaků s diakritikou v kódování UTF-8. Nevýhodou tohoto přístupu je skutečnost, že daný postup bude generovat i neplatné znaky - tedy dvojice bajtů, které nepotřebujeme.

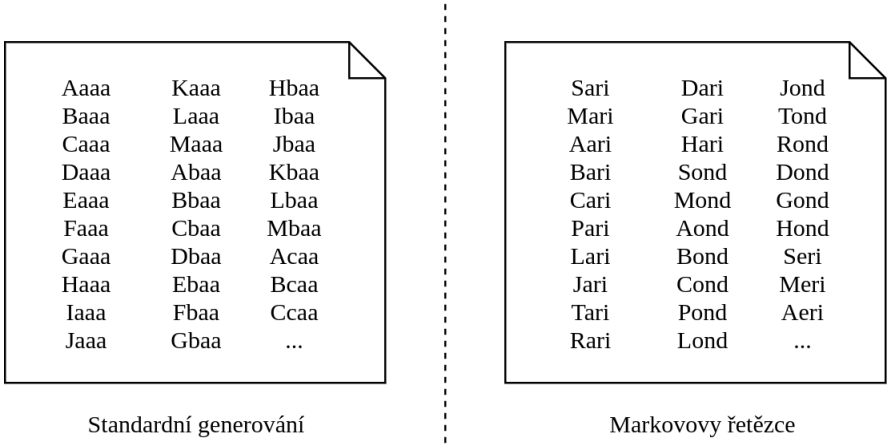
2.3.2 Generování hesel Markovovými řetězci Nejjednodušší způsob generování znaků je v pořadí daném lexikografickým uspořádáním, tj. např. pro množinu malých písmen (se zástupným symbolem `?l`) nejprve vygenerujeme znak `a`, následně `b`, apod. Posledním vygenerovaným znakem je `c`. Takový způsob generování však nereflektuje pravděpodobnost správnosti jednotlivých hesel. Máme-li velké množství hesel, může být výhodné vyzkoušet pravděpodobnější hesla dříve.

V nástroji hashcat jsou k tomuto účely využity Markovovy řetězce [14]. Ty obecně popisují proces, u něhož dochází ke změně stavu s určitou pravděpodobností, založenou na znalosti stavu aktuálního. Pokud máme k dispozici slovník reálných hesel, můžeme určit pravděpodobnost, se kterou bude po znaku `X`, což je aktuální stav, následovat znak `Y`. Tyto pravděpodobnosti následně seřadíme a pro každý znak zapíšeme do matice. První řádek udává pravděpodobnosti prvního znaku v hesle, poté znaku následujícího za znakem `a`, `b` až `z`. Příklad takové matice je možné vidět v tabulce 2. Tuto 2D matici je možné dále rozšířit do 3D prostoru pomocí vrstvového modelu, kde matice v každé vrstvě reprezentuje jednu pozici znaku v hesle. Tedy například první sloupec a třetí řádek v páté vrstvě matice by udával nejpravděpodobnější znak následující po znaku *b*, které se nachází v hesle na páté pozici. Pokud máme k dispozici dostatečně velkou trénovací množinu hesel, vrstvový model vykazuje o něco lepší úspěšnost než model klasický [4].
Kromě pořadí vygenerovaných hesel nám Markovovy řetězce dovolují rovněž omezit stavový prostor hesel zvolením hodnoty prahu (*threshold*). Ten nám určí maximální index sloupce matice, který bude při generování uvažován. Tím dosáhneme vygenerování jen těch nejpravděpodobnějších hesel, relativně k trénovací množině. Bylo ukázáno, že s využitím Markovových řetězců je možné výrazně zmenšit stavový prostor hesel se zachováním velmi vysoké úspěšnosti při hádání

ϵ	n p s ...
a	n l t ...
b	o e a ...
c	h i e ...
⋮	⋮ ⋮ ⋮ ⋱
z	a e o ...

Tabulka 2: Ukázka matice nahrazující pravděpodobnostní přechody

hesel [4]. Na obrázku 3 je možné vidět porovnání začátku slovníku, vygenerovaného pomocí masky `?u?l?l?l`. Vlevo je generování standardním způsobem nástroje hashcat, který se vzdáleně podobá opačnému lexikografickému uspořádání. Napravo je pak vidět použití Markovových řetězců, které byly natrénovány na známém slovníku hesel *rockyou.txt*, který obsahuje asi 15 milionů hesel.



Obrázek 3: Ukázka generovaných hesel pomocí Markovových řetězců

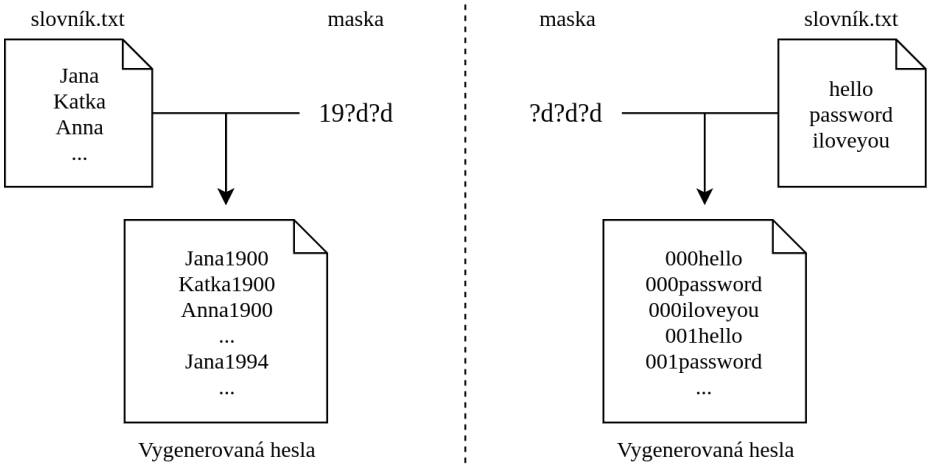
2.4 Hybridní útoky

Hybridní útoky představují kombinaci přístupů slovníkového útoku a útoku hrubou silou s využitím masky. Podobně jako u útoku kombinačního, dochází ke skládání hesel ze dvou separátních částí: levé a pravé. Jedna z nich je tvořena slovníkovými hesly (viz sekci 2.1) a druhou generujeme na základě masky (viz sekci 2.3.1). Podle toho, která část je generována jakým způsobem, rozlišujeme dva podtypy:

- **hybridní útok slovník + maska**, kdy levou část hesla generujeme ze slovníku a pravou z masky;

- **hybridní útok maska + slovník**, kdy levou část hesla generujeme z masky a pravou ze slovníku.

Oba útoky jsou k sobě navzájem zrcadlové, jak ukazuje obrázek 4. Můžeme si všimnout, že hybridní útok na obrázku generuje podobné výsledky jako příklad s použitím masky v sekci 2.3.1, ale mnohem efektivněji. Do slovníku totiž můžeme uvést pouze seznam jmen a nemusíme se spoléhat na náhodně generované řetězce. Tím rapidně snížíme stavový prostor vygenerovaných hesel. Na straně pravé jsme upravili masku tak, aby nám generovala roky narození a to v rozmezí let 1900 – 1999. Tato maska `19?d?d` nám zřejmě vygeneruje stejný počet hesel jako předchozí maska `?d?d`, protože první dvě číslice jsou pouze statický prefix.



Obrázek 4: Vlevo ukázka použití hybridního útoku typu 6, vpravo typu 7

Tento typ útoku je možné také nahradit použitím pravidel pro modifikaci hesel (viz sekci 2.1.1) a to tak, že na každém řádku použitého souboru pravidel umístíme jednu variantu použité masky. V praxi lze použít nástroj *maskprocessor*, dostupný v repozitáři nástroje hashcat, který dokáže vygenerovat na základě masky odpovídající pravidla.

3 Distribuce výpočtu

V předchozí sekci byly představeny jednotlivé typy útoků podporované nástrojem hashcat. Základní myšlenka realizace výpočtu na více uzlech spočívá v distribuci indexů hesel [10]. S ohledem na specifika nástroje hashcat však bylo nutné pro každý typ útoku zvolit odlišnou strategii. Jednotlivé techniky budou dále popsány.

3.1 Distribuce slovníkového útoku

Pro distribuci slovníkového útoku je nutné každému výpočetnímu uzlu zaslat všechna hesla, která má ověřit. Oproti útoku hrubou silou, kde zasíláme pouze masku a rozsah indexů hesel [10], zde přenášíme znatelně vyšší objem dat. Efektivitu takového útoku silně ovlivňuje velikost vstupního slovníku, rychlost přenosové sítě a také samotná náročnost výpočtu daného typu kryptografického heše. Na základě předchozí analýzy [11] docházíme k závěru, že pro jednodušší formáty může docházet k jevu, kdy výpočetní uzly zpracovávají hesla natolik rychle, že značnou část doby lámání pouze čekají na hesla nová. V extrémních případech může být útok na jednom stroji dokonce výhodnější, než útok distribuovaný.

U nástroje Hashtopolis dochází před zahájením samotného výpočtu k distribuci kompletního vstupního slovníku všem výpočetním uzlům. Následně jsou uzlům přidělovány pouze indexy hesel, které mají zkoušet. Tento způsob je jednoduchý na implementaci, ale v praxi není ideální, neboť každý uzel dostane i množství hesel, která nikdy nevyužije. Negativní dopad tohoto přístupu diskutuje případová studie v sekci 5.

Výhodou slovníkového útoku pomocí nástroje hashcat je fakt, že stavový prostor udávaný tímto nástrojem je roven počtu hesel ve slovníku. Tím nám odpadá složitý přepočet, který je nutné provádět u maskového útoku.

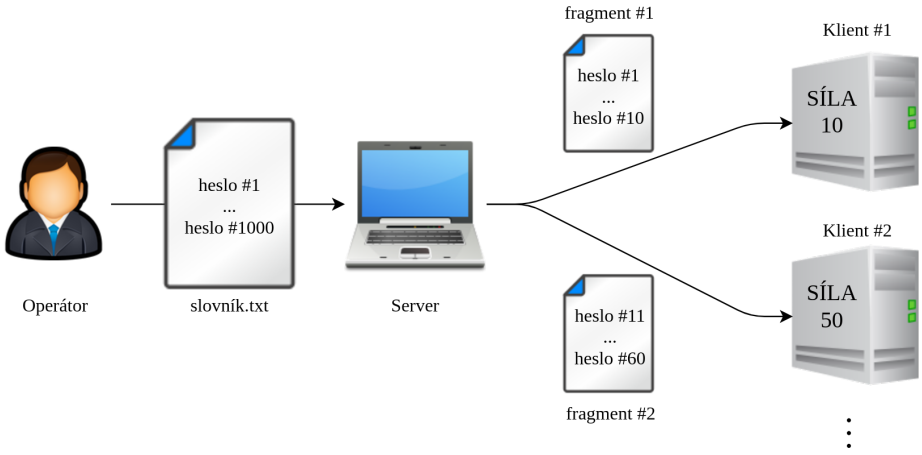
V případě systému Fitcrack je zvolen přístup, kdy uzlu zasíláme pouze hesla, která bude skutečně ověřovat. Vstupní slovník hesel je postupně rozdělován do fragmentů, které jsou následně zasílány jednotlivým klientům. Server si pro daný útok ve vstupním slovníku udržuje pozici posledního přiděleného hesla. Do fragmentu slovníku zasílaném v rámci každé dílčí úlohy pak použije pouze část hesel od uložené pozice, která je následně posunuta dále. Velikost fragmentů určuje adaptivní plánovací algoritmus [7] na základě aktuálního výpočetního výkonu uzlu a požadované době běhu jedné pracovní jednotky.

Princip generování úloh slovníkového útoku je možné vidět na obrázku 5. Je zde také zobrazena síla jednotlivých stanic a k nim odpovídající velikost fragmentu.

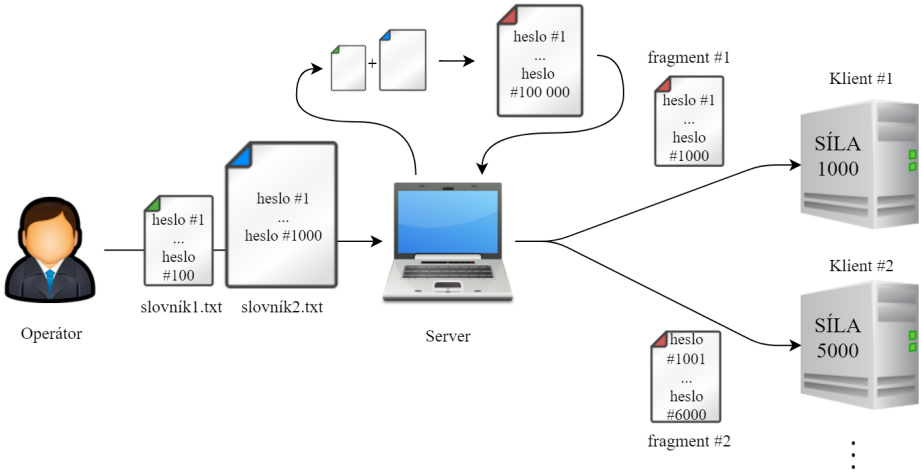
3.2 Distribuce kombinačního útoku

Nejnaivnějším přístupem by bylo využití nástroje dostupného v repozitáři *hashcat-utils*[13] nazývaného *combinator*. Tento nástroj funguje jako samostatná implementace kombinačního útoku. Na vstupu očekává 2 slovníky a jeho výstupem je seznam všech možných kombinací hesel z obou slovníků. Následně bychom s takto získaným slovníkem provedli klasický slovníkový útok, popsaný v předchozí kapitole. Tento přístup je však nanejvýš neefektivní, protože by bylo nutné přenášet obrovské množství hesel navíc. U původního kombinačního útoku je potřeba v nejlepším možném případě přenést pouze $(m + n)$ hesel, a to v případě, kdy by celou úlohu zvládl vypočítat jediný uzel. Ve výše navrhovaném řešení by to však bylo $(m \times n)$ hesel – prostorová složitost by se tedy zvýšila z lineární na kvadratickou. Takovýto přístup je možné vidět na obrázku 6.

[13] https://github.com/hashcat/hashcat-utils

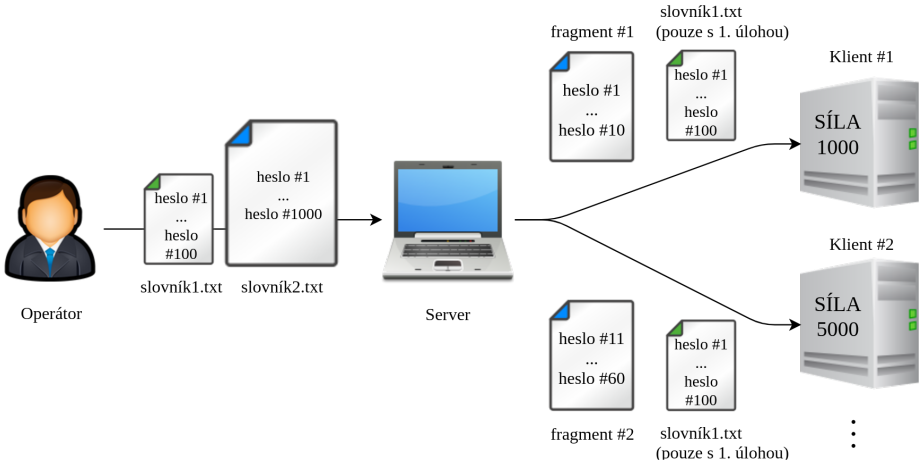


Obrázek 5: Distribuce slovníku mezi klienty



Obrázek 6: Naivní přístup ke kombinačnímu útoku – generování na straně serveru

Slovníky je tedy nutné určitým způsobem rozdělit již před distribucí. Bohužel není možné fragmentovat oba slovníky zároveň, protože by nedošlo k ověření kombinací hesel z dvou rozdílných úloh. Zbývající možností je tedy zaslat každému výpočetnímu uzlu jeden slovník celý a další slovník poté dělit na jednotlivé fragmenty. Zřejmě zde tedy nedosáhneme ideální lineární prostorové složitosti zmíněné výše, ve většině případů bude ale prostorová složitost drasticky nižší, než pokud bychom posílali již zkombinovaný slovník. Opačný případ by nastal jen tehdy, pokud by klienti vždy vyzkoušeli pouze 1 heslo z druhého slovníku (či méně, viz dále) nebo by bylo připojeno více klientů, než je hesel v zasílaném slovníku, což při běžné velikosti slovníků desítek čí stovek megabajtů v praxi nenastane. Diagram tohoto řešení vidět na obrázku 7. Je možné si všimnout právě velké úspory zasílaných dat mezi obrázky 6, kde je nutné klientovi o síle 1000 zaslat vždy tisíc hesel, zatímco v přístupu 7 je to při každé nové úloze pouze 10 hesel.



Obrázek 7: Reálné provedení kombinačního útoku – generování na klientské straně

Při takovémto přístupu se vyskytují ještě dva zřejmé problémy. Vzhledem k tomu, že první zaslaný slovník se s časem nemění, není nutné jej zasílat všem klientům při každé přidělené úloze znovu, tedy tak, jak to probíhá u typických projektových souborů. To by způsobilo obrovské zpomalení celého procesu, a to kvůli již zmíněné velmi značné velikosti slovníků. Řešením je využití tzv. *sticky* příznaku ve vstupní šabloně útoku, kterým klientům sdělíme, aby daný soubor po dokončení úlohy nemazali, ale nechali si jej pro další práci. Navíc je nutné si informaci o zaslání pamatovat na serveru. Toto však již systém BOINC řeší sám. Modul Generator tedy naplánuje odeslání všech souborů. Pokud se ale již na klientu nachází soubor se stejným názvem a obsahem, k opětovnému přenosu nedojde.

Druhým problémem je minimální počet hesel, který je možné v jedné úloze vyzkoušet. Ve výše navrženém postupu by to bylo minimálně $(n \times 1)$ hesel. Pokud však bude n příliš velké nebo rychlost obnovy velmi malá, může to mít za následek mnohem delší dobu výpočtu jedné úlohy, než požadoval uživatel. Taková situace by mohla v praxi jednoduše nastat, protože velikost reálných slovníků může být v jednotkách milionů hesel a ještě mnohem více, pokud by šlo o slovník vygenerovaný na základě pravděpodobností, gramatik [18] či jiných technik [5]. Ty totiž často dovolují generovat předem neomezené množství hesel a rovněž nástroj hashcat není nijak limitován velikostí slovníků. Řešením je zavedení nového sloupce v tabulce *fc_package – hc_index_2*. Zatímco první index bude sloužit pro počítání hesel zaslaných z druhého slovníku, počítadlo *hc_index_2* bude použito pro indexování ve slovníku prvním. Tím dosáhneme, pomocí argumentů nástroje hashcat `--limit` a `--skip`, rozdělení i prvního slovníku.

V tomto ohledu bude také vhodně implementována jistá inteligence plánovače. Můžeme očekávat, že fragmentace prvního slovníku může mít negativní dopad na výkon klienta. To například v případě, že by nám zbyl jen zlomek obsahu slovníku, který by následně dostal přidělen velmi výkonný klient. Případně, pokud by se taková situace periodicky opakovala. Počet přidělených hesel tedy bude vhodně zaokrouhlován. Pro výkonné stroje, které zvládnou v rámci úlohy vypočítat více než jedno heslo z druhého slovníku, budou plánovány úlohy bez fragmentace prvního slovníku. Implementační detaily této problematiky budou ukázány v následující kapitole.

3.3 Distribuce útoku hrubou silou

Zatímco u předchozího útoku je zřejmé, jakým způsobem je prováděno indexování ve stavovém prostoru hesel, u maskového útoku se jedná o značný problém. Nástroj hashcat totiž u maskového útoku nedokáže poskytnout reálný počet hesel a naopak pracuje se svou vlastní aritmetikou. Poskytovaný počet hesel tak záleží na délce masky a často také na obnovovaném formátu. Ve výsledku bychom mohli říct, že poskytnutý stavový prostor nemá s reálným stavovým prostorem nic společného. Bohužel tuto aritmetiku musíme využít pro plánování maskových útoků a přidělovat klientům odpovídající množství práce na základě znalosti rychlosti procesu obnovy, která je dána v heslech (respektive heších) za sekundu.

Tento problém je vyřešen naším vlastním skriptem, který byl implementován jako součást nové webové administrace. Ten dokáže na základě masky vypočítat reálný počet hesel. Při následném porovnání s hashcat stavovým prostorem zjistíme, kolik reálných hesel je reprezentováno jediným hashcat indexem. Na základě této informace pak můžeme vytvořit dílčí úlohu s odpovídající velikostí.

V nástroji hashcat máme možnost s využitím parametrů `--increment-min` a `--increment-max` specifikovat minimální a maximální délku hesla s danou maskou. Tento útok však nástroj hashcat provede jako více útoků s jednotlivými maskami. V distribuovaném prostředí jsme tuto situaci vyřešili s využitím tabulky *fc_mask*. Jediný balík nyní může obsahovat více masek a to nejen těch

stejných s odlišnou délkou. Při plánování tohoto útoku jsou pak jednotlivé masky procházeny sekvenčně a postupně přidělovány jednotlivým výpočetním uzlům.

Ve výsledku tak klient počítá vždy s jedinou maskou a to ve stanoveném rozsahu indexů. Ty jsou na klientské straně zadány pomocí hashcat parametrů `--limit` a `--skip`. Pokud navíc modul Generator naplánuje dílčí úlohu, která dosahuje až na konec stavového prostoru dané masky, je vytvořena úloha pouze s daným začátkem stavového prostoru, bez omezení velikosti. Jedná se tedy o pojistku, pokud by došlo k nesrovnalosti v převodu reálných hesel a hashcat indexů. Klient s takovou úlohou prověří masku vždy až do konce. Díky tomu nemůže dojít k vynechání části masky, ve které by se například nacházelo heslo.

3.4 Distribuce hybridních útoků

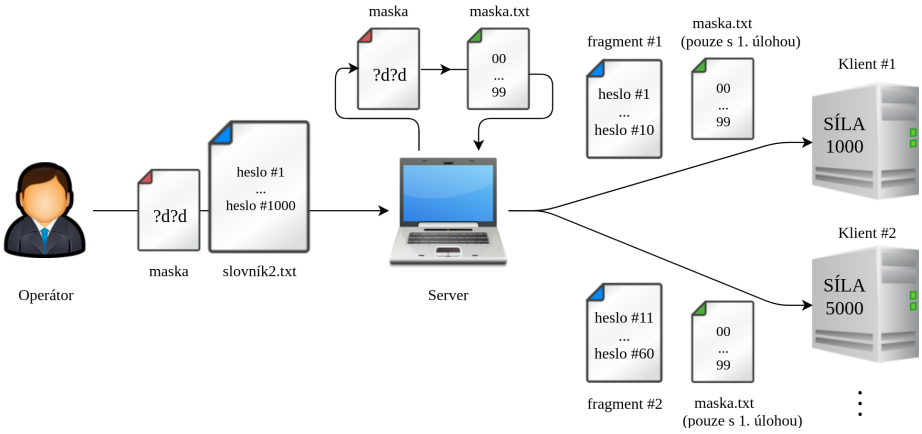
Pokud se podíváme na funkčnost hybridního útoku z pohledu plánování, zjistíme, že se velmi podobá kombinačnímu útoku. Opět zde máme na jedné straně množinu hesel uloženou ve slovníku. Na straně druhé je maska, která taktéž reprezentuje jistou množinu hesel. Bohužel zde opět narážíme na problém s reprezentací stavového prostoru nástrojem hashcat. Ten uvádí pro hybridní útoky stavový prostorný počtu hesel ve slovníku. V kombinačním útoku jsme měli navíc možnost zaslat v druhém slovníku pouze malý počet hesel a rozsah ověřovaných hesel v prvním slovníku kontrolovat argumentem `--limit`. To samé se bohužel nedá provést u hybridních útoků, protože nemáme prostředky, jak zaslat pouze část masky. V praxi by se tak při vyzkoušení jediného indexu stavového prostoru musela ověřit kompletní maska a to v rámci jediné dílčí úlohy. Ve výsledku by tak délka úlohy byla nekontrolovatelná a často v řádech několika dní, místo uživatelem zvoleného času.

Pro řešení tohoto problému bychom mohli aplikovat naivní postup popsaný v sekci kombinačního útoku 3.2. Tedy generovat dopředu všechny kombinace a tyto následně distribuovat jednotlivým klientům jako klasický slovníkový útok. Nicméně zde opět narážíme na zvýšení prostorové složitosti na kvadratickou, což by vedlo ke zdlouhavému přenosu všech kombinací hesel k ověření.

Další možností by bylo tento proces provádět až na klientské části – tedy například klientu zaslat jedno heslo s onou dlouhou maskou spolu s informací, kolik hesel má v dané úloze ověřit. Klient by si následně vygeneroval možné kombinace a indexování by prováděl až nad nimi. Tato metoda s sebou ovšem nese několik problémů. První z nich je ten, že Generator při přidělování úlohy hybridního útoku nemá k dispozici informaci o reálném ani hashcat stavovém prostoru masky, pouze samotného útoku, který se rovná, jak jsme již zmiňovaly, počtu hesel ve slovníku. Museli bychom tedy provádět vlastní propočet reálného stavového prostoru a ten přepočítávat na hashcat formát. Dalším problémem by mohla být velikost generovaného slovníku na klientské straně. U delších masek by mohlo generování trvat velmi dlouho a výsledný slovník by mohl zabírat gigabajty pevného disku. Navíc je problematické vygenerovaný slovník zachovat napříč jednotlivými úlohami, protože aktuálně na klientské straně neexistuje unikátní identifikátor jednoho obnovovaného souboru. Runner následně nemá

informaci o tom, kdy by mohl vygenerovaný slovník smazat. Pokud by se generování provádělo při každé úloze znovu, mohlo by se jednat o ještě pomalejší variantu, než první navrhované naivní řešení.

Konečné řešení, které je kompromisem mezi generováním jednoho velikého slovníku a složitými procesy na klientské straně, je transformace tohoto útoku na kombinační útok. Při vytvoření hybridního útoku se ze zadané masky vygeneruje slovník hesel, pomocí hashcat nástroje *maskprocessor*. Modul Generator a stejně tak klient budou následně s tímto útokem pracovat jako s kombinačním, kde jsou na vstupu zadané dva slovníky. Diagram tohoto řešení je vidět na obrázku 8.



Obrázek 8: Reálné provedení hybridních útoků – transformace na kombinační útok

4 Případová studie: Útok na bezdrátovou síť zabezpečenou pomocí WPA a WPA2

Tato případová studie popisuje techniky pro odchyt komunikace v bezdrátové síti zabezpečené pomocí WPA/WPA2 a ukazuje, jak lze zachycenou komunikaci použít pro zjištění hesla v síti. Techniky pro odchyt komunikace jsou vybrány tak, aby bylo následně k prolomení hesla možno použít nástroj hashcat, respektive Fitcrack. ubsectionPřehled metod nalezení hesel WPA/WPA2 sítí

Na WPA a WPA2 existují dva hlavní typy útoků:

- útok využívající klíče **Pairwise-Transient-Key** (PTK),
- útok využívající identifikátoru **Pairwise-Master-Key** (PMKID).

Oba typy útoku se pak následně v nástroji hashcat dělí na dva typy hešů, které lze nástroji poskytnout. První verze útoků (heše č. 2500 a 16800) vyžadují část

odchycených dat nebo informací z nich obsažených. Zatímco druhé verze (heše č. 2501 a 16801) navíc ještě obsahují před-generovanou část Pair-Master-Key (PMK). Výpočet PMK je proveden pomocí funkce PBKDF2 s 4096 iteracemi a 256-bitovým výstupem a je závislý na hodnotách[14]:

- **Pre-Shared-Key** (PSK) nebo hodnotě klíče vytvořeného pomocí *Extensible Authentication Protocol* (EAP),
- **Service Set Identifier** (SSID) - jméně Wi-Fi sítě.

PMK pro určité SSID lze tedy před-počítat. Existují nástroje jako je například *genpmk*[15], které umožňují vytvořit si soubor s před-počítanými hodnotami. Nástroj vyžaduje jméno SSID a soubor obsahující slovník hesel.

Vlastnictví souboru s PMK nám umožňuje urychlení následného obnovení hesla, ovšem pouze za předpokladu, že provádíme více záchytů více 4-cestných navázání spojení na síti se stejným SSID a že obnovu hesla provádíme vícekrát než jednou. V případě, kdy provádíme odchyt a následné obnovení hesla pouze jedenkráte žádného zrychlení nedosáhneme[16]. Musím totiž provést stejné operace:

1. vygenerování hodnoty PMK,
2. využití hodnoty pro vygenerování hodnoty zaslané mezi klientem a přístupovým bodem (podle typu útoku),
3. porovnání hodnoty vygenerované v předchozím bodě s odpovídající hodnotou v odchycené komunikaci.

4.0.1 Využití Pairwise-Transient-Key

V tomto prvním případě je, jak již bylo zmíněno výše, nutné vygenerovat z hesla a SSID PMK. To dále s dalšími hodnotami slouží k vytvoření PTK. Pomocí PTK jsme schopni z odchycené komunikace extrahovat Key-Confirmation-Key, díky kterému jsme pak schopni vypočítat Message Integrity Code (MIC). Tento vypočtený kód se nakonec porovnává s odchyceným MIC. V případně shody bylo heslo správné a v opačném případě je potřeba vzít nové heslo a všechny operace znovu.

Pro získání všech potřebných hodnot je třeba odchytit 4-cestné navázání spojení. Tato operace může být časově náročná obzvláště pokud chceme pouze pasivně odposlouchávat a neprozrazovat svoji přítomnost. Více viz sekce 4.1.

4.0.2 Využití identifikátoru Pairwise-Master-Key

Tento útok byl objeven teprve v dubnu 2018 během analýzy bezpečnostního standardu WPA3. Je dostupný pouze na některý, převážně moderních směrovačích podporujících standardy 802.11i/p/q/r a se zapnutou podporou přecházení mezi sítěmi. Není však zatím známo jestli je tato technika použitelná na všechny takové směrovače od všech výrobců [16].

[14] https://www.ins1gn1a.com/understanding-wpa-psk-cracking/
[15] https://github.com/joswr1ght/cowpatty
[16] https://hashcat.net/forum/thread-7337-post-39566.html#pid39566

Útok [16] využívá pole Robust Security Network Information Element (RSN IE), které může být obsaženo v řídícím rámci 802.11 komunikace (EAPOL[17]). Z něj konkrétně hodnotu RSN PMKID . Výpočet RSN PMKID využívá metodu HMAC-SHA1 s hodnotami:

- PMK jako klíč (vyžaduje heslo + SSID),
- vstupní data jsou sjednocením:
  - řetězce "PMK Name",
  - Media Access Control (MAC) adresy přístupového bodu,
  - MAC adresy připojené stanice.

Ověření pak probíhá porovnáním odchycené a vypočtené hodnoty RSN PMKID.

4.1 Odchyt informací o spojení

Pro odchyt WPA a WPA2 komunikace existuje několik sad nástrojů, které lze použít k získání potřebné informace zmíněné v sekci 4. Tyto nástroje se lehce liší ve funkcionalitě a formátech výstupů, které jsou schopné produkovat. Níže jsou uvedeny balíky nástrojů preferované komunitou nástroje Hashcat, balíků je ovšem více. Kromě níže uvedených je to například *Besside-ng*.

4.1.1 Nástroje pro odchyt WPA/WPA2 komunikace

Aircrack-ng Prvním balíkem nástrojů je *Aircrack-ng*[18]. Balík lze získat buďto z uvedené adresy a nebo z oficiálních repozitářů většiny Linuxových distribucí. *Aircrack-ng* je také možné použít na operačním systému MS Windows. Z této sady potřebujeme pouze omezené množství nástrojů. Konkrétně:

- *Airmon-ng* - slouží k přepínání bezdrátového síťového rozhraní do monitorovacího módu (nutné pouze na zařízeních s OS Linux)[19],
- *Airodump-ng* - slouží k odchycení a uložení nemodifikovaných 802.11 rámců[20],
- *Aireplay-ng* - slouží k provedení de-autentizačního útoku na WEP, WPA a WPA2 spojení s cílem vynutit provedení nového 4-cestného navázání spojení[21].

V případě, kdy odchytíme velké množství dat lze využít nástroje *Wireshark*[22] pro filtraci .cap souboru vytvořeného *Airodump-ng*.

Odchyt pomocí této sady nástrojů je již považován, pro účely obnovy hesel, za zastaralý a komunita preferuje nástroje uvedené v následující sekci.

[17] Extensible Authentication Protocol over LAN
[18] https://www.aircrack-ng.org/
[19] https://www.aircrack-ng.org/doku.php?id=airmon-ng
[20] https://www.aircrack-ng.org/doku.php?id=airodump-ng
[21] https://www.aircrack-ng.org/doku.php?id=aireplay-ng
[22] https://www.wireshark.org/

Hcxdumptool Druhým a komunitou doporučovaným balíkem jsou *Hcxtools*[23]. Jedná se o balík nástrojů vytvořených pro odchycení a nachystání dat pro použití s nástroji *Hashcat* a *John the Ripper*. Pro odchytávání dat nás zajímá pouze nástroj *hcxdumptool*[24], dříve známý jako *wlandump-ng*.

Hcxdumptool, narozdíl od *Airodump-ng*, který ukládá data jako .cap, ukládá data jako .pcapng. To po odchytu vede na použití jiných nástrojů na převod odchycených dat do formátů, kterým rozumí nástroje pro obnovu hesel.

Hlavní výhodou tohoto nástroje oproti předchozímu je o něco jednodušší použití viz praktická ukázka v sekci 4.1.2.

4.1.2 Ukázka odchytu komunikace

Aircrack-ng Prvním krokem k odchycení komunikace je přepnutí adaptéru do monitorovacího režimu. K tomu slouží nástroj *Airmon-ng*. Celý příkaz zajišťující přepnutí režimu bezdrátového adaptéru je `airmon-ng start <název_bezdrátového_rozhraní>`, konkrétně viz Obrázek 9. Pro zjištění názvu bezdrátového rozhraní lze využít taktéž nástroj *Airmon-ng* a to následovně `airmon-ng`.

```
root@demo-UBNT14:/tmp# airmon-ng start wlan0


Found 6 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
967     avahi-daemon
975     avahi-daemon
1102    NetworkManager
1434    wpa_supplicant
1509    dhclient
3066    dhclient
Process with PID 1509 (dhclient) is running on interface wlan0


Interface       Chipset         Driver

wlan0           Unknown         rtl8192ce - [phy0]
                                (monitor mode enabled on mon0)
```

Obrázek 9: Ukázka přepnutí adaptéru wlan0 do monitorovacího režimu.

Druhým krokem vyhledání Wi-Fi sítí v okolí. V rámci tohoto kroku nám jde hlavně o získání MAC adresy přístupového bodu a kanálu, na kterém je daná síť vysílá. K tomuto účelu nám poslouží nástroj *Airodump-ng*. Příkaz, který nám k tomu poslouží, je `airodump-ng <název_monitorovacího_rozhraní>` (viz Obrázek 10).

[23] https://github.com/ZerBea/hcxtools
[24] https://github.com/ZerBea/hcxdumptool

```
root@demo-UBNT14:/tmp# airodump-ng mon0
 CH -1 ][ Elapsed: 4 s ][ 2018-10-11 19:22

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 F4:EC:38:F9:54:DC  -19       71        0    0   1  54e. WPA2 CCMP   PSK  to_cr
 5C:F4:AB:05:A4:FB  -70        3        0    0   1  54e  WPA2 CCMP   PSK  Fialo

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes
```

Obrázek 10: Ukázka vyhledání Wi-Fi sítí v okolí.

Posledním krokem odchytu informací je samotný odchyt a uložení dat do souboru. Znovu použijeme nástroj *Airodump-ng*. Tentokrát mu ovšem předáme další parametry (BSSID, kanál vysílání, název souboru pro uložení odchytu). Příkaz pak vypadá následovně `airodump-ng -bssid <MAC_přístupového_bodu> -channel <číslo_kanálu> -w <název_souboru> <název_monitorovacího_rozhraní>` (viz Obrázek 11).

```
root@demo-UBNT14:/tmp# airodump-ng --bssid F4:EC:38:F9:54:DC --channel 1 -w to_c
rack.cap mon0
 CH  1 ][ Elapsed: 28 s ][ 2018-10-11 19:39

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH E

 F4:EC:38:F9:54:DC  -24  96      262      171    0   1  54e. WPA2 CCMP   PSK  t

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 F4:EC:38:F9:54:DC  28:ED:6A:41:4D:B3  -53    1e- 1     22      592
```

Obrázek 11: Ukázka odchycení 4-cestného navázání spojení.

Hcxtools Jak již zmíněno, druhým postupem je použití nástroje *Hcxdumptool*. Zde je pouze jeden krok a jen minimum informací, které musíme nástroji ručně předat. Stačí nám pouze název souboru, do kterého mají být odposlechnutá data uložena a jméno bezdrátového rozhraní. Příkaz vypadá následovně `hcxdumptool -o <název_soubor> -i <název_bezdrátového_rozhraní> -t <doba_odposlouchávání_kanálu> -enable-status=1`. Ukázka použití uvedeného příkazu s očekávaným výstupem je ukázána na Obrázku 12.

4.2 Ukázka obnovy hesel

Po provedení úspěšného odchytu komunikace je dalším krokem příprava odchycených dat pro použití s nástrojem pro obnovu hesel, v našem případě nástrojem *Hashcat*. Protože kroky uvedené v sekcích 4.1.1 a 4.1.1 produkují výstupní data

```
root@demo-UBNT14:/tmp# hcxdumptool -o to_crack.pcapng -i wlan0 -t 5 --enable_sta
tus=1
warning: NetworkManager is running with pid 1102
warning: wpa_supplicant is running with pid 1434

start capturing (stop with ctrl+c)
INTERFACE:...............: wlan0
FILTERLIST...............: 0 entries
MAC CLIENT...............: f0a225075bbc
MAC ACCESS POINT.........: f04f7cb2b792 (incremented on every new client)
EAPOL TIMEOUT............: 150000
REPLAYCOUNT..............: 64374
ANONCE...................: 79951d00a2f184a3bcf04d4280010458e2edb88d697e903f08117
bed0ae0ba03

[19:04:00 - 001] 28ed6a414db3 -> f4ec38f954dc [FOUND HANDSHAKE AP-LESS, EAPOL TI
MEOUT 37431]
INFO: cha=1, rx=2140, rx(dropped)=415, tx=137, powned=1, err=0^C
terminated...
```

Obrázek 12: Ukázka odchytu komunikace pomocí *Hcxdumptool*.

v rozdílných formátech (.cap a .pcapng) potřebujeme dva rozdílné nástroje pro zpracování dat. Jedná se o nástroje *cap2hccapx* z balíku nástrojů *Hashcat-utils*[25] a o nástroj *Hcxpcaptool* z balíku nástrojů *Hcxtools*.

4.2.1 Příprava odchycených dat Nástroje na přípravu dat mají za úkol v odchycených datech identifikovat hodnoty, které jsou nutné pro ověření správnosti použitého hesla a k vytvoření PTK nebo PMKID.

4.2.2 Aircrack-ng Z dat poskytnutých nástrojem *Airodump-ng* můžeme pomocí nástroje *cap2hccapx* dostat pouze informace pro útoku na PTK (sekce 4.0.1), tedy informace o 4-cestném navázání spojení. Použití nástroje je jednoduché `cap2hccapx <souboru_s_komunikací> <souboru_s_daty_pro_obnovu>` (viz Obrázek 13).

```
root@demo-UBNT14:/tmp# ./cap2hccapx.bin to_crack.cap-03.cap to_crack_handshake_h
ash.hccapx
Networks detected: 1

[*] BSSID=f4:ec:38:f9:54:dc ESSID=to_crack (Length: 8)
 --> STA=28:ed:6a:41:4d:b3, Message Pair=0, Replay Counter=1
 --> STA=28:ed:6a:41:4d:b3, Message Pair=2, Replay Counter=1

Written 2 WPA Handshakes to: to_crack_handshake_hash.hccapx
```

Obrázek 13: Ukázka extrakce dat z odchycené komunikace pomocí *cap2hccapx*.

[25] https://github.com/hashcat/hashcat-utils

4.2.3 Hcxtools Pomocí nástroje *Hcxpcaptool* můžeme z komunikace dostat nejen data pro útok na PTK, ale i data pro útok na PMKID (pokud byla v komunikaci přítomna). To jaké informace bude nástroj vyhledávat a i jaký bude formát výstupu určují přepínače při spouštění nástroje. Pro získání informací o 4-cestném navázání spojení slouží přepínač `-o`, také viz Obrázek 14. Informace pro případný útok na PMKID jde získat použitím přepínače `-z`. Pro více o přepínačích nástroje a jejich použití použijte `hcpcaptool -h`. Příkaz použitý v ukázce níže (Obrázek 14) má podobu `hcpcaptool -o <soubor_s_daty_pro_obnovu> <soubor_s_komunikací>`.

```
root@demo-UBNT14:/tmp# hcxpcaptool -o to_crack_handshake_hash.hccapx ./to_crack.
pcapng

summary:
--------
file name.....................: to_crack.pcapng
file type.....................: pcapng 1.0
file hardware information....: x86_64
file os information..........: Linux 3.19.0-79-generic
file application information.: hcxdumptool 4.2.1
network type..................: DLT_IEEE802_11_RADIO (127)
endianess.....................: little endian
read errors...................: flawless
packets inside................: 79
skipped packets...............: 0
packets with GPS data.........: 0
packets with FCS..............: 68
beacons (with ESSID inside)..: 2
probe requests................: 3
probe responses...............: 5
association requests.........: 2
association responses........: 5
authentications (OPEN SYSTEM): 21
authentications (BROADCOM)...: 2
EAPOL packets.................: 42
best handshakes..............: 1 (ap-less: 0)

1 handshake(s) written to to_crack_handshake_hash.hccapx
```

Obrázek 14: Ukázka extrakce dat z odchycené komunikace pomocí *Hcxpcaptool*.

4.3 Obnova hesla

Ať už jste použili nástroje ze sady *Aircrack-ng* nebo *Hcxtools*, následující a již poslední krok je společný, protože po provedení sekcí 4.2.2 a 4.2.3 jsou data již identická. Posledním krokem je samotna obnova hesla nástrojem *Hashcat*. Nástroj má mnoho přepínačů a parametrů[26], kterými jde nastavit spousta věcí. V ukázce na Obrázku 15 jsou použity pouze ty nejzákladnější a nejnutnější pro úspěšný běh na stroji, který byl vybrán pro vytvoření ukázky. Obecně základní

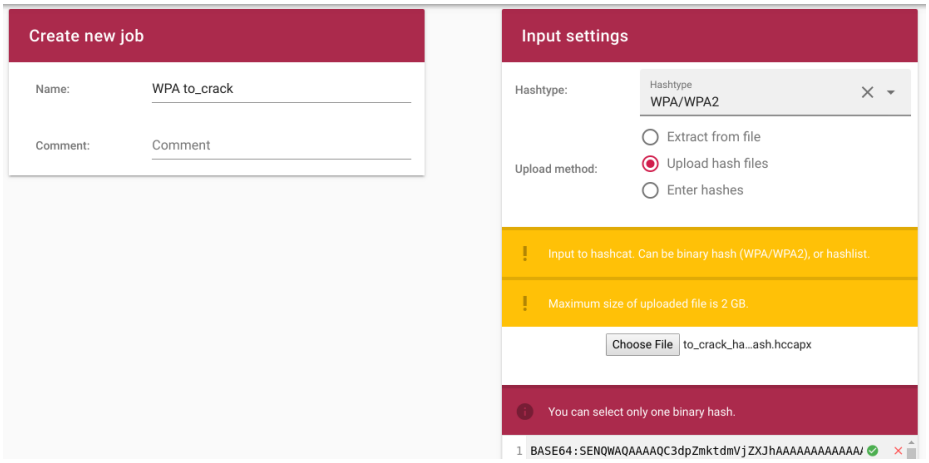
[26] https://hashcat.net/hashcat/

podoba příkazu, spouštějícího útok hrubou silou, vypadá následovně `hashcat -m 2500 <soubor_s_informacemi_pro_obnovu> -a 3 <maska_hesla>`. Dále byly také použity i doplňující parametry maximalizující rychlost obnovy hesla a zajišťující běh na systému se zastaralými ovladači pro grafickou kartu. Při vytváření masky byla brána v potaz rychlost obnovy hesla na dané sestavě a proto byla zakomponována i částečná znalost hesla tak, aby ukázková obnova hesla netrvala několik hodin nebo dokonce dní.

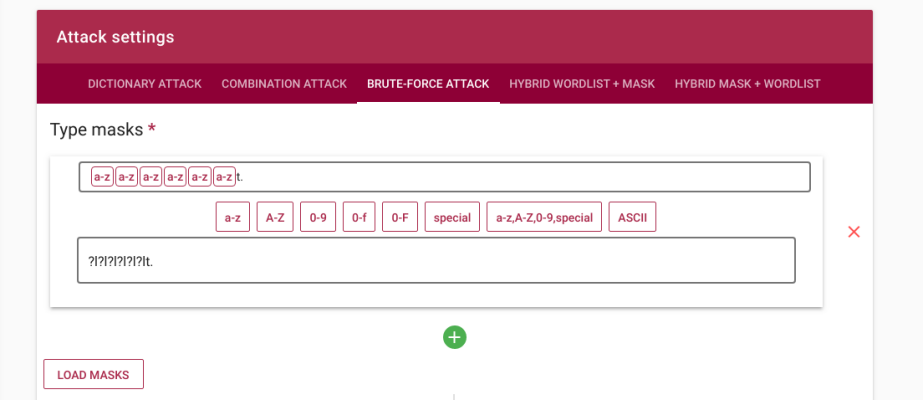
```
root@demo-UBNT14:/tmp# hashcat -a 3 -m 2500 to_crack_handshake_hash.hccapx ?l?l?
l?l?l?lt. -w 4 -O --force
hashcat (v4.2.1) starting...

OpenCL Platform #1: Advanced Micro Devices, Inc.
================================================
* Device #1: Tahiti, 1685/2420 MB allocatable, 32MCU
* Device #2: AMD FX(tm)-8320 Eight-Core Processor, skipped.

/usr/local/share/hashcat/OpenCL/m02500-optimized.cl: Optimized OpenCL kernel req
uested but not needed - falling back to pure OpenCL kernel
Hashes: 2 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force
* Slow-Hash-SIMD-LOOP

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63

Watchdog: Temperature abort trigger set to 90c

d7a85d17591aaf1ade0270123d504928:f4ec38f954dc:28ed6a414db3:to_crack:hashcat.

Session..........: hashcat
Status...........: Cracked
Hash.Type........: WPA-EAPOL-PBKDF2
Hash.Target......: to_crack (AP:f4:ec:38:f9:54:dc STA:28:ed:6a:41:4d:b3)
Time.Started.....: Thu Oct 11 19:45:24 2018 (3 mins, 34 secs)
Time.Estimated...: Thu Oct 11 19:48:58 2018 (0 secs)
Guess.Mask.......: ?l?l?l?l?l?lt. [8]
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:    63585 H/s (503.27ms) @ Accel:128 Loops:128 Thr:256 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 13631488/308915776 (4.41%)
Rejected.........: 0/13631488 (0.00%)
Restore.Point....: 0/11881376 (0.00%)
Candidates.#1....: hariert. -> hggcybt.
HWMon.Dev.#1.....: Temp: 71c Fan: 76% Util: 99% Core:1100MHz Mem:1500MHz Bus:16

Started: Thu Oct 11 19:45:18 2018
Stopped: Thu Oct 11 19:48:59 2018
```

Obrázek 15: Ukázka obnovení hesla nástrojem *Hashcat*.

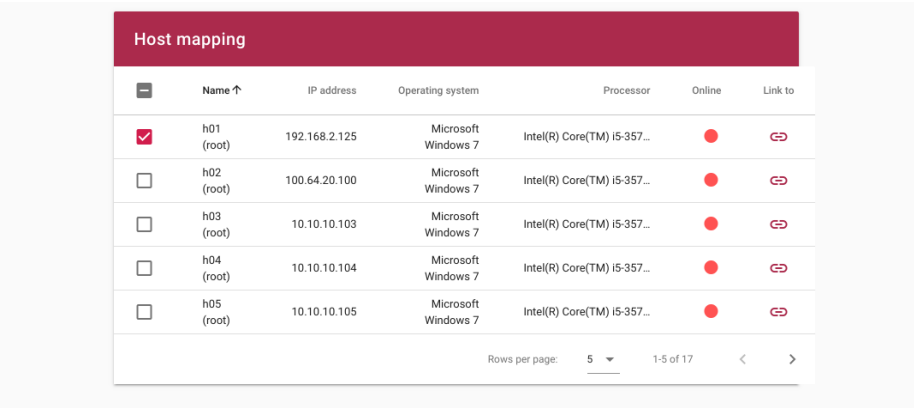
Druhý způsobem obnovy hesla je pomocí nástroje *Fitcrack*. Přidání WPA-/WPA2 heše do systému je na obrázku 16. Následně je potřeba na stejné stránce přidat masku (obrázek 17) a nakonec namapovat k úloze výpočetní uzly (obrázek 18). Po provedení kroků z obrázků je potřeba kliknout na tlačítko `SUBMIT`, které se nachází na konci stránky. Úlohu je následně potřeba spustit pomocí zeleného start tlačítka.



Obrázek 16: Ukázka přidání WPA/WPA2 heše do nástroje *Fitcrack*.



Obrázek 17: Ukázka přidání požadované masky v nástroji *Fitcrack*.



Obrázek 18: Ukázka přiřazení uzlů k nové úloze v nástroji *Fitcrack*.

4.4 Závěr

Sada nástrojů *Hcxtools* je po právu komunitou preferovanou volbou pro získání informací WPA/WPA2 spojení. Je tomu tak díky:

- velkému množství nástrojů podporujících různé formáty dat,
- jednoduchosti použití,
- aktuálnosti nástrojů,
- spolupráci autorů nástrojů *Hcxtools* a *Hashcat*.

Nástroje *Aircrack-ng* sice již od verze 1.4 podporují odchyt informací pro útok na PMKID ovšem nástroj *cap2hccapx* je již zastaralý[27] a nedokáže tyto informace zpracovat do formátu, ve kterém je *Hashcat* vyžaduje.

Pro získání hesla je také možné využít nástroje *Aircrack-ng*. Ten má ovšem oproti nástroji *Hashcat* značné nevýhody[28]:

- podporuje běh pouze na procesoru (menší úroveň paralelizace -> nižší rychlost obnovy hesla),
- podporuje pouze útok pomocí před-generovaného slovníku hesel.

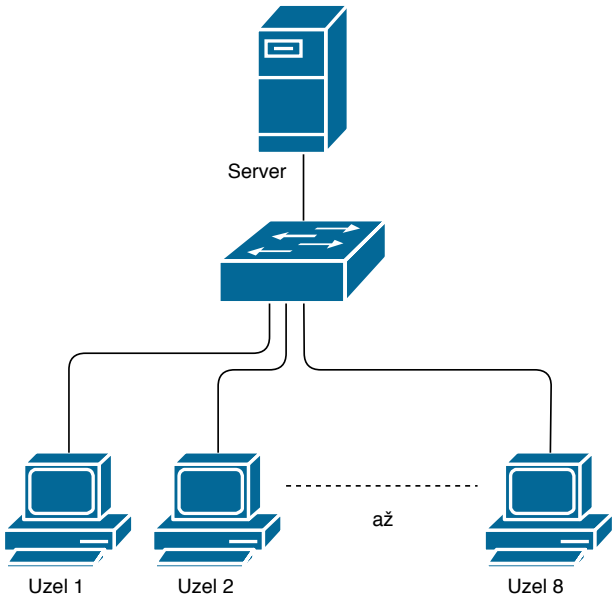
[27] https://hashcat.net/forum/thread-7772-post-41814.html#pid41814
[28] https://www.aircrack-ng.org/doku.php?id=aircrack-ng

5 Případová studie: Distribuovaný slovníkový útok

Tato případová studie ukazuje praktické důsledky použití odlišných strategií při realizaci distribuovaného slovníkového útoku.

V sekci 3.1 bylo popsáno, že zatímco nástroj Hashtopolis na začátku útoku provede distribuci celého slovníku všem výpočetním uzlům, systém Fitcrack provádí jeho fragmentaci přímo na serveru. Jednotlivým uzlům jsou pak zasílány pouze fragmenty tohoto slovníku. Cílem studie je experimentálně demonstrovat efektivitu fragmentace slovníků v systému Fitcrack oproti přenášení celého slovníku v nástroji Hashtopolis.

Během experimentů byl objeven nedostatek v implementaci bloku *Generator* [10] systému Fitcrack. Díky tomuto docházelo v každé iteraci k opětovnému otevírání původního slovníku a přeskakování hesel - tento případ ukazují měření měření označené jako **Fitcrack(2)** v sekcích 5.1 a 5.3. Tento problém byl následně vyřešen zdokonalením implementace, kdy je slovník otevírán pouze jednou a blok Generator si uchovává pozici v souboru - měření po zdokonalení označujeme jako **Fitcrack(3)**.



Obrázek 19: Topologie experimentální sítě

5.1 Podmínky

V počítačové síti s topologií ilustrovanou na obrázku 19 je přepínači připojen server a 8 stejných výpočetních uzlů. Konfigurace jednotlivých prvků je následující:

- **Server**:
  - **CPU**: Intel(R) Xeon(R) CPU E31230 @ 3.20GHz
  - **RAM**: 32,0 GB RAM DDR3
  - **HDD**: WDC WD10EFRX-68F
- **Uzel 1** až **Uzel 8**:
  - PC LYNX PowerMedia i5
  - **CPU**: Intel(R) Core(TM) i5-3570K CPU @ 3.40GHz
  - **RAM**: 8,0 GB
  - **GPU**: GIGABYTE GTX 1050 Ti
  - **HDD**: WD Caviar RE WD5000ABYS
- Propojovací síť:
  - ethernetové spoje s rychlostí 1Gb/s.

V síti budeme provádět distribuovaný slovníkový útok na heše získané pomocí algoritmu SHA-1 [1] a výpočetně náročnějšího algoritmu Whirlpool [15] s použitím slovníků o velikostech 1,1 GB, 2,1 GB, 4,2 GB a 8,3 GB. Hledané heslo, tj. vstupní hodnota hešovací funkce, je vždy poslední v použitém slovníku. Pro distribuovaný útok provedeme vždy čtyři sady experimentů:

- **Fitcrack(1)** - S použitím systému Fitcrack a použitím pouze **jediného** výpočetního uzlu, kterému předáme celý slovník hesel.
- **Fitcrack(2)** - S použitím systému Fitcrack, všech osmi výpočetních uzlů a původní metody fragmentace slovníku. Velikost dílčí úlohy (*seconds per workunit*) [7, 10] je stanovena na 60s.
- **Fitcrack(3)** - S použitím systému Fitcrack, všech osmi výpočetních uzlů a zdokonalé techniky fragmentace slovníku. Velikost dílčí úlohy je stanovena na 60s.
- **Hashtopolis** - S použitím nástroje Hashtopolis, všech osmi výpočetních uzlů a původní metody fragmentace slovníku. Velikost dílčí úlohy (u Hashtopolis známá jako *chunk size*] stanovena na 60s.

5.2 Hypotéza

Protože použité nástroje (Fitcrack, Hashtopolis) využívají protkolu HTTP [3] a nepodporují přenos typu broadcast, předpokládáme, že při přenosu velkých slovníků dojde k zalhcení linky mezi přepínačem a serverem. Zatímco v případě systému Fitcrack díky fragmentaci bude po lince každé slovníkové heslo přenášeno pouze jednou, v případě Hashtopolis je nutné přenosem typu unicast přenést celý slovník (tedy všechny hesla) ke všem uzlům. Očekáváme, že v případě systému Fitcrack bude doba nalezení hesla výrazně kratší, neboť zde neočekáváme takovou přenosovou režii.

5.3 Výsledky

Naměřené výsledky všech čtyř sad experimentů vyobrazuje tabulka 3. Tyto byly také vyneseny do grafů doby nalezení hesla v závislosti na velikosti použitého slovníku. Graf pro algoritmus SHA-1 ukazuje obrázek 20, pro algoritmus Whirlpool pak obrázek 21.

heš	slovník	Fitcrack(1)	Fitcrack(2)	Fitcrack(3)	Hashtopolis
SHA1	1,1 GB	3m 15s	2m 42s	2m 40s	2m 22s
SHA1	2,1 GB	3m 20s	3m 20s	3m 28s	4m 52s
SHA1	4,2 GB	6m 5s	4m 34s	4m 2s	11m 14s
SHA1	8,3 GB	12m 49s	10m 1s	4m 58s	29m 51s
Whirlpool	1,1 GB	3m 15s	3m 36s	3m 11s	2m 39s
Whirlpool	2,1 GB	4m 36s	3m 25s	3m 16s	5m 56s
Whirlpool	4,2 GB	8m 0s	4m 0s	4m 17s	12m 13s
Whirlpool	8,3 GB	17m 31s	8m 42s	5m 2s	45m 51s

Tabulka 3: Distribuovaný slovníkový útok u Fitcracku a Hashtopolis

Z výsledků měření vidíme, že pro slovník velikosti 1,1 GB na použitém hardware, nemá použitá technika fragmentace, zásadnější vliv. V tomto případě dosahuje nástroj Hastopolis kratší doby prolomení hesla, poněvadž zde režie fragmentace komlpetně odpadá.

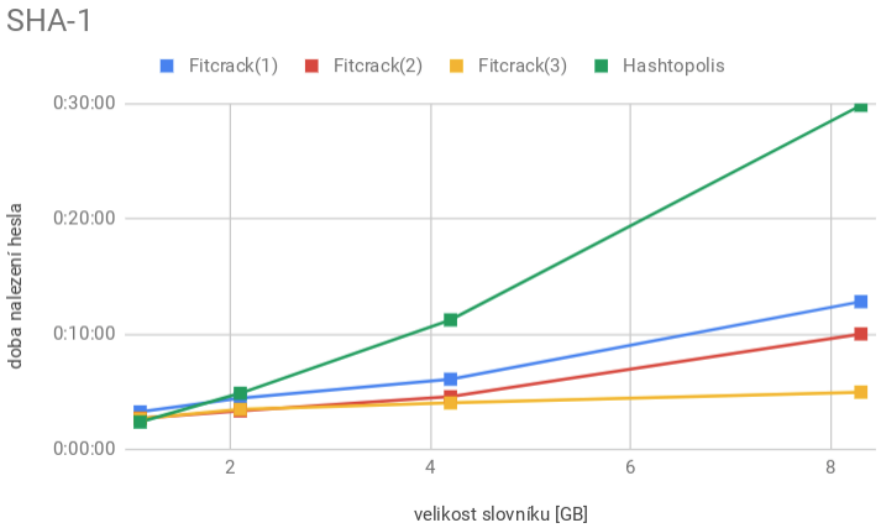
U slovníku velikosti 2,1 GB na osmi uzlech již dosahuje systém Fitcrack lepších výsledků. Efektivita fragmentace se plně projevuje při použití slovníků velikosti 4,2 a 8,3 GB, kdy v případě nástroje Hashtopolis dochází k zahlcení linky a systém Fitcrack dosahuje výrazně kratších dob nalezení hesla i při útoku s použitím jediného uzlu. Nutno podotknout, že většina času potřebného k prolomení hesla byla u nástroje Hashtopolis strávena přenosem slovníku. Hypotézu uvedenou v sekci 5.2 jsme tedy potvrdili.

5.4 Závěr

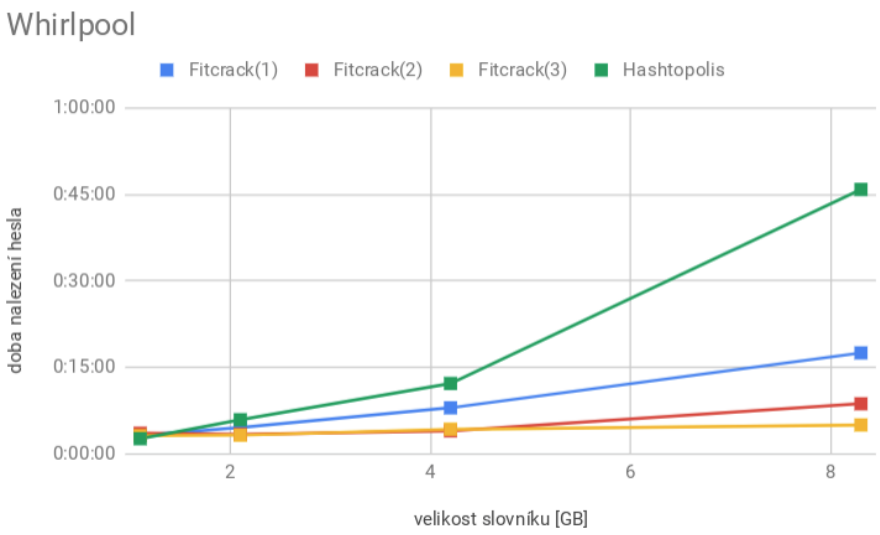
Na základě naměřených výsledků vidíme, že pro velké slovníky je technika fragmentace nezbytnou nutností. Distribuce celého slovníku je sice principielně jednodušší a pro malé slovníky může být vhodnější, nicméně obecně vede ke zbytečné zátěži sítě. S rostoucí velikostí slovníku pak stoupá i režie nutná pro přenos. Ta je navíc ještě násobena počtem výpočetních uzlů.

Odkazy

[1] D. Eastlake 3rd a P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. Tech. zpr. 3174. Updated by RFCs 4634, 6234. Zář. 2001. URL: http://www.ietf.org/rfc/rfc3174.txt.



Obrázek 20: Slovníkový útok na SHA-1



Obrázek 21: Slovníkový útok na Whirlpool

[2] Adobe Systems Incorporated. *Document management — Portable document format — Part 1: PDF 1.7*. 32000-1:2008. Geneva, Switzerland: ISO, čvc 2008.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk a T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Tech. zpr. 2068. Obsoleted by RFC 2616. Led. 1997. URL: http://www.ietf.org/rfc/rfc2068.txt.

[4] Peter Gazdík. „Využití heuristik při obnově hesel pomocí GPU“. czech. Bakalářská práce. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2016. URL: http://www.fit.vutbr.cz/study/DP/BP.php?id=18210.

[5] Shiva Houshmand, Sudhir Aggarwal a Randy Flood. „Next gen PCFG password cracking“. In: *IEEE Transactions on Information Forensics and Security* 10.8 (2015), s. 1776–1791. ISSN: 1556-6013.

[6] Radek Hranický. *Digitální forenzní analýza s použitím distribuovaného prostředí*. Teze disertační práce. Fakulta informačních tehnologií VUT v Brně. 2016.

[7] Radek Hranický, Martin Holkovič, Petr Matoušek a 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. URL: http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11276.

[8] Radek Hranický, Petr Matoušek, Ondřej Ryšavý a Vladimír Veselý. „Experimental Evaluation of Password Recovery in Encrypted Documents“. In: *Proceedings of ICISSP 2016*. Roma, IT: SciTePress - Science a Technology Publications, 2016, s. 299–306. ISBN: 978-989-758-167-0. URL: http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11052.

[9] Radek Hranický, Petr Matoušek, Ondřej Ryšavý a Vladimír Veselý. „Experimental Evaluation of Password Recovery in Encrypted Documents“. Angl. In: *Proceedings of ICISSP 2016*. Roma, IT: SciTePress - Science a Technology Publications, 2016, s. 299–306. ISBN: 978-989-758-167-0. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11052.

[10] Radek Hranický, Lukáš Zobal a Vojtěch Večeřa. *Distribuovaná obnova hesel*. czech. Tech. zpr. FIT-TR-2017-04, CZ, 2017, s. 14. URL: http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11568.

[11] Radek Hranický, Lukáš Zobal, Vojtěch Večeřa a Petr Matoušek. „Distributed Password Cracking in a Hybrid Environment“. Angl. In: *Proceedings of SPI 2017*. Brno, CZ: Brno University of Defence, 2017, s. 75–90. ISBN: 978-80-7231-414-0. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11358.

[12] Keonwoo Kim. „Distributed password cracking on GPU nodes“. In: *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*. Pros. 2012, s. 647–650.

[13] Dávid Mikuš. „Obnova hesel archivů RAR, BZIP a GZIP s využitím GPU“. czech. Bakalářská práce. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2015. URL: http://www.fit.vutbr.cz/study/DP/BP.php?id=18740.

[14] Arvind Narayanan a Vitaly Shmatikov. „Fast dictionary attacks on passwords using time-space tradeoff“. In: *Proceedings of the 12th ACM conference on Computer and communications security*. ACM. 2005, s. 364–372. ISBN: 1-59593-226-7.

[15] William Stallings. „The Whirlpool Secure Hash Function“. In: *Cryptologia* 30 (2006), s. 55–67.

[16] Jens Steube. *New attack on WPA/WPA2 using PMKID*. [online]. 2018. URL: https://hashcat.net/forum/thread-7717.html (cit. 10. 10. 2018).

[17] Vojtěch Večeřa. „Obnova hesel archivů ZIP s využitím GPU“. czech. Bakalářská práce. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2015. URL: http://www.fit.vutbr.cz/study/DP/BP.php?id=18211.

[18] Matt Weir, Sudhir Aggarwal, Breno De Medeiros a Bill Glodek. „Password cracking using probabilistic context-free grammars“. In: *30th IEEE Symposium on Security and Privacy*. IEEE. 2009, s. 391–405. ISBN: 978-0-7695-3633-0.

[19] X. Wu, J. Hong a Y. Zhang. „Analysis of OpenXML-based office encryption mechanism“. In: *2012 7th International Conference on Computer Science Education (ICCSE)*. Čvc 2012, s. 521–524. DOI: 10.1109/ICCSE.2012.6295128.

[20] Lukáš Zobal. „Obnova hesel dokumentů Microsoft Office s využitím GPU“. czech. Bakalářská práce. Brno, CZ: Vysoké učení technické v Brně, Fakulta informačních technologií, 2015. URL: http://www.fit.vutbr.cz/study/DP/BP.php?id=18341.