VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

OBNOVA SDÍLENÝCH KLÍČŮ PROTOKOLU WIRELESS M-BUS

WIRELESS M-BUS KEY RETRIEVAL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ MIKULA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Mikula Ondřej**
Program: Informační technologie
Název: **Obnova sdílených klíčů protokolu Wireless M-Bus**
Wireless M-Bus Key Retrieval
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s nástroji Hashcat, John the Ripper, Fitcrack a obdobnými nástroji pro obnovu hesel.
2. Seznamte se s protokolem Wireless M-Bus a jeho existujícími bezpečnostními módy.
3. S využitím existujících nástrojů pro obnovu hesel navrhněte algoritmus pro obnovu klíčů používaných bezpečnostních módů protokolu Wireless M-Bus, zaměřte se zejména na bezpečnostní mód 5.
4. Algoritmus implementujte po dohodě s vedoucím např. jako modul pro Hashcat.
5. Implementaci otestujte.
6. Práci vyhodnoťte a navrhněte možná rozšíření.

Literatura:

- ČSN EN 13757-1. *Komunikační systémy pro měřidla - Část 1: Výměna dat*. Praha: Česká agentura pro standardizaci, 2018.
- ČSN EN 13757-4. *Komunikační systémy pro měřidla - Část 4: Bezdrátová komunikace M-Bus*. Praha: Česká agentura pro standardizaci, 2019.
- ČSN EN 13757-7. *Komunikační systémy pro měřidla - Část 7: Doprava a bezpečnostní služby*. Praha: Česká agentura pro standardizaci, 2019.

Pro udělení zápočtu za první semestr je požadováno:

- První 3 body zadání.

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

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Konzultant: Večeřa Vojtěch, Ing., UIFS FIT VUT
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Tato práce se zabývá vytvořením pluginu k nástroji pro obnovu hesel, který přidá podporu pro šifrované zprávy protokolu Wireless Meter Bus. V rámci práce byly implementovány dva pluginy pro program Hashcat, které podporují bezpečnostní módy 05 a 07, používající šifrovací algoritmus AES-128. Na současném hardware vytvořené pluginy dosahují rychlosti desítek až stovek Mhash/s, což postačuje k prohledání pouze malého zlomku množiny možných klíčů. Přesto je možné, při vhodném omezení prohledávané množiny, výsledek této práce použít k provedení bezpečnostní analýzy, kterou norma Wireless M-Bus při nasazení požaduje.

Abstract

This thesis aims to create a plugin for one of existing password recovery tools to extend its functionality to support encrypted messages of Wireless Meter Bus protocol. As a result, two plugins for the Hashcat tool have been implemented, supporting security modes 05 and 07 of Wireless M-Bus. Those modes are making use of the AES-128 encryption algorithm, so it is dealt with throughout this thesis. The resulting plugins are capable of speeds in range of tens to hundreds Mhash/s, which makes it unfeasible to check the whole AES-128 keyspace in a reasonable time. Nevertheless, if the searched keyspace is appropriately reduced, the created plugins make it possible to perform an security audit of the Wireless M-Bus deployment, which is required to be done by the specification.

Klíčová slova

Wireless Meter Bus, Hashcat, AES-128, obnova klíčů, šifrování, plugin, bezdrátový odečet, měřiče spotřeby

Keywords

Wireless Meter Bus, Hashcat, AES-128, key recovery, encryption, plugin, wireless readout, consumption meter

Citace

MIKULA, Ondřej. *Obnova sdílených klíčů protokolu Wireless M-Bus*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Obnova sdílených klíčů protokolu Wireless M-Bus

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka Ph.D. Další informace mi poskytl Ing. Vojtěch Večeřa. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Mikula
10. května 2022

Poděkování

Tímto děkuji vedoucímu mé bakalářské práce, panu Ing. Liboru Polčákovi, Ph.D., za hodnotné rady a připomínky k textu práce a k práci samotné a také panu Ing. Vojtěchu Večeřovi za konzultace a podněty ohledem obnovy klíčů a implementačních detailů programu Hashcat.

Obsah

Kapitola 1

Úvod

Od roku 2022 mají být v České republice přístroje pro odečet spotřeby energií v bytových blocích, v případě, že je to ekonomicky výhodné a technicky proveditelné, nahrazovány novějšími přístroji, které umožňují dálkový odečet [21]. Jedním z možných řešení dálkového odečtu je bezdrátový odečet, při kterém měřící přístroj vysílá naměřené hodnoty do svého okolí v pravidelných intervalech, například jednou za minutu. Díky tomu již nemusí pověření zaměstnanci vstupovat do jednotlivých bytů a odečítat hodnoty z ciferníku na měřících přístrojích, místo toho jim stačí počkat na signál před domem, případně ve společných prostorách domu.

Tento nově zaváděný způsob odečtu měřících přístrojů vyvolává problém – kdokoli s potřebným vybavením (jehož cena začíná již na desítkách amerických dolarů [10]) může, aniž by vstupoval do soukromých prostor, bez oprávnění přistupovat k datům, která jsou dle GDPR klasifikována jako osobní informace. Tento problém je řešen šifrováním přenášených dat. K nešifrovaným datům má tak přístup jen osoba, která zná šifrovací klíč.

Cílem této práce je navrhnout a implementovat nástroj, který se pokusí obnovit klíč ze zachycených šifrovaných zpráv pomocí vyzkoušení každého ze zadané množiny klíčů. V případě úspěchu, tedy nalezení správného klíče, bude moci uživatel data následně rozšifrovat a přečíst v nezašifrované podobě některým z již existujících programů (například *wmbusmeters*[1]). Důležitým požadavkem na výsledek práce je její integrace do některého z již ustálených nástrojů pro obnovu klíčů.

Práce se zabývá jedním z hojně používaných protokolů pro bezdrátový odečet měřících přístrojů, protokolem *Wireless Meter Bus*. V době psaní této práce není autorovi znám jiný nástroj, který umožňuje pokusit se obnovit klíč z šifrovaných zpráv tohoto protokolu. Existence takového nástroje má dva hlavní důsledky: dokazuje proveditelnost[2] takového typu útoku pro nezákonné aktivity, což je jeden z výstupů bezpečnostní analýzy, která má být provedena před nasazením, a také tím nutí odbornou veřejnost neustále vyhodnocovat bezpečnost používaných technologií. Zadruhé, umožňuje získat větší kontrolu legitimním uživatelům měřících přístrojů, kterým v mnoha případech výrobce klíč nedodá, aby byli donuceni používat přijímače stejného výrobce (tzv. *vendor lock-in*).

V textu práce je čtenář proveden od rozboru relevantních detailů komunikačního protokolu a použitého šifrovacího algoritmu (kapitola 2 – *Protokol Wireless Meter Bus*), teorie obnovy klíčů v kapitole 3 – *Nástroje pro obnovu hesel*, přes návrh a optimalizaci řešení v kapitole 4 – *Návrh nástroje pro obnovu klíčů*, popis jeho implementace (kapitola 5 – *Im-*

[1] https://github.com/weetmuts/wmbusmeters

[2] V případě, že proveditelnost dokázána není, tím není dokázána neproveditelnost.

plementace nástroje pro obnovu klíčů), až k ověřování vlastností vytvořeného nástroje a proveditelnosti útoku v kapitole 6 – *Vyhodnocení a testování* a nakonec kapitole 7 – *Závěr*.

Kapitola 2

Protokol Wireless Meter Bus

Tato kapitola se věnuje obecnému popisu protokolu Wireless Meter Bus a podrobnějšímu popisu jeho částí, které jsou pro tuto práci podstatné. V následujících podkapitolách je krátce nastíněna jeho historie, obecný popis, síťový zásobník, použitý šifrovací algoritmus a části specifikace relevantní pro tuto práci.

Protokol Wireless Meter Bus[1] je určen k dálkovým, bezdrátovým odečtům měřících přístrojů na vzdálenost v řádech desítek až stovek metrů. Podporuje obousměrnou komunikaci, kterou vždy zahajuje měřící přístroj [8].

Wireless Meter Bus je určen k přenášení dat, jako jsou informace o spotřebě plynu a vody v domácnostech, která mohou být, vzhledem k jejich povaze, klasifikovány zákony jako osobní informace [7]. Norma ČSN EN–13757–7 vyžaduje, aby provozovatel sítě před nasazením tohoto protokolu do provozu vytvořil bezpečnostní politiku, provedl analýzu bezpečnostních požadavků a analýzu hrozeb. Nástroj, který je výsledkem této práce, má umožnit některé aspekty těchto analýz vyhodnocovat.

Norma ČSN EN–13757–1 stanovuje tyto cíle zabezpečení bezdrátových odečtů [6]:

- zajištění, že k informacím mají přístup jen ty osoby, které k nim mají povolen přístup (soukromnost),
- zajištění, že informace nejsou pozměněny, omylem či úmyslně (integrita),
- zajištění, že zdroj informace nemůže být zfalšován (autentifikace),
- zajištění, že zdroj nemůže být odmítnut (neodmítnutelnost).

Soukromnost je zajištěna šifrováním. Cílem této práce je vytvořit nástroj, který dokáže odhalit případný nedostatek ve splnění bodu 1 v konkrétní implementaci v daném měřícím přístroji.

2.1 Meter Bus

Meter Bus, zkráceně M-Bus, je průmyslový standard, určený pro dálkové odečty vodoměrů, plynoměrů a elektroměrů [6]. Podporuje také obousměrnou komunikaci. Původně byl vyvinut pro drátovou komunikaci Prof. Dr. H. Zieglerem na univerzitě v Paderbornu [19]. V současnosti je definován skupinou evropských standardů EN–13757 (viz tabulku 6.4) a spravován skupinou OMS (Open Metering System) [9]. Do českých norem je přejat pod

[1] Dále v práci jsou používány také zkrácené formy názvu – Wireless M-Bus a WM-Bus.

označením ČSN EN–13757 v původním anglickém znění. Protokolem Meter Bus se zabývá norma ČSN EN 13757–1 [6], bezdrátovým přenosem M-Bus norma ČSN EN 13757–4 [8], bezpečností přenosu se zabývá norma ČSN EN 13757–7 [7]. Rozbor samotného M-Bus není pro tuto práci podstatný, jelikož v rámci ní představuje jen přenášená šifrovaná data v aplikační vrstvě.

Název	**popis**	**poznámka**
ČSN EN–13757–1	**Výměna dat**	
ČSN EN–13757–2	Komunikace pomocí kabelové sběrnice M-Bus	
ČSN EN–13757–3	Aplikační protokoly	
ČSN EN–13757–4	**Bezdrátová komunikace M-Bus**	
ČSN EN–13757–5	Bezdrátový přenos M-Bus	(*relaying* – předávání přes prostředníky)
ČSN EN–13757–6	Lokální sběrnice	
ČSN EN–13757–7	**Doprava a bezpečnostní služby**	specifikace bezpečnostních módů a požadavků na bezpečnostní analýzu

Tabulka 2.1: Normy definující M-Bus. Normy podstatné pro tuto práci jsou zvýrazněny tučně.

2.2 Vrstvy

Koncept síťových vrstev (zásobníku) WM-Bus je založen na sedmivrstvém ISO-OSI modelu dle ISO/IEC 7498–1 [5]. V případě, že zpráva není po cestě od odesílatele k příjemci předávána pomocí prostředníka (anglicky tzv. *relaying*), je zásobník zjednodušen na třívrstvý model dle EN 61334–4–1, který znázorňuje tabulka 2.2.

	Vrstva ISO-OSI modelu	**WM-Bus**	**Norma**
7.	Aplikační	nezávislá vrstva, přenášená data	-
6.	Prezentační	-	-
5.	Relační	-	-
4.	Transportní	Transport Layer (TPL)	EN 13757-7
3.	Síťová	-	-
2.	Linková	Data Link Layer (DLL)	EN 13757–3
1.	Fyzická	bezdrátový přenos	EN 13757–4 a EN 13757–5

Tabulka 2.2: Zásobník protokolů WM-Bus.

Aplikační vrstva

Obsahuje samotná přenášená data a je (typicky) šifrována. V případě použití algoritmu AES–CBC–128 musí být její obsah zarovnán na násobky 16 bytů. K tomu má být pro

Meter Bus dle specifikace použita konstanta $2F2F_H$[2] [7]. Díky rozvrstvení lze na aplikační vrstvě použít i jiné protokoly, například COSEM [6], KNX [12], či PRIOS [15].

Transportní vrstva (Transport Layer – TPL)

V hlavičce této vrstvy je v bitech 8 až 12 konfiguračního pole (CI – *Configuration Field*) uvedeno číslo použitého bezpečnostního módu. Dále se zde vyskytují hodnoty, ze kterých je vytvořen inicializační vektor, jak je zobrazeno v tabulce 2.3.

význam	ID výrobce		adresa						zarovnání (padding)							
velikost [B]	2		6						8							
příklad	6D	14	84	22	50	14	05	07	A4	A4	A4	A4	A4	A4	A4	A4

Tabulka 2.3: Struktura inicializačního vektoru (IV).

Linková vrstva (Data Link Layer – DLL)

Wireless M-Bus používá dva rámce s různou strukturou, A a B. Formát rámce je detekován z preambule se synchronizačním slovem. Zkratka *CF* má význam *Control Field*, česky pole s kontrolními informacemi. V tabulce 2.4 je uvedena struktura bloků v častěji používaném formátu B.

1. blok

význam	délka	CF	výrobce	adresa	celkem
velikost [B]	1	1	2	6	10

2. blok

význam	CF	data	CRC	celkem
velikost [B]	1	až 115	2	až 118

3. blok

význam	data	CRC	celkem
velikost [B]	až 126	2	až 128

Tabulka 2.4: Struktura rámce ve formátu B.

Na linkové vrstvě je také prováděna kontrola CRC (*Cyclic Redundancy Check*, algoritmus používaný k detekci a opravení chyb při přenosu). Polynom pro WM-Bus je definován takto:

$$x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + 1$$

a jeho počáteční hodnota je 0 [6].

Fyzická vrstva

Standard pro tuto vrstvu definuje frekvence bezdrátového vysílání v pásmu do 1 GHz a možné modulace:

[2]Hodnota konstanty záleží na aplikačním protokolu. Tato práce se zabývá M Bus a dále pracuje jen s touto konstantou.

- 2-FSK – Frequency Shift-Keying – modulace využívající posun frekvence k přenosu informací.
- 2-GFSK – Gaussian Frequency Shift-Keying - využívá gaussovský filtr.
- 4-GFSK – GFSK se čtyřmi stavy.

2.3 Bezpečnostní módy Wireless M-Bus

Norma Wireless M-Bus specifikuje položku bezpečnostní mód (anglicky *security mode*) v konfiguračním poli (anglicky *configuration field*) transportní vrstvy, kde obsazuje bity 8 až 12 [7]. Bezpečnostní mód určuje, který šifrovací algoritmus bude použit pro zajištění soukromnosti dat na aplikační vrstvě pomocí šifrování. Tabulka 2.5 uvádí módy specifikované normou.

Z podstaty nasazení měřících přístrojů, které jsou typicky napájeny baterií, norma ČSN EN–13757–7 specifikuje pro Wireless Meter Bus pouze symetrické šifry, které jsou, v porovnání s asymetrickými šiframi, méně výpočetně a paměťově, a tím i energeticky, náročné. Normou definované služby jako *derivace klíčů* a *distribuce klíčů* řeší problém životnosti (doby, po kterou může být považován za bezpečný) klíčů. Ty, díky jejich jednorázovému použití, nevyžadují inicializační vektor. [7]

mód	**šifrování**	**IV**
0	bez nešifrování	-
1	ponecháno na výrobci	-
2	DES	nulové
3	DES	nenulové
4	specifické využití	-
5	AES–CBC–128	nenulové
6	rezervováno pro budoucí využití	-
7	AES–CBC–128 (typicky časově omezený klíč)	nulové
8	AES–CTR–128, CMAC	-
9	AES–GCM–128	-
10	AES–CCM–128	-
11	rezervováno pro budoucí využití	-
12	rezervováno pro budoucí využití	-
13	specifické využití	-
14	rezervováno pro budoucí využití	-
15	specifické využití	-
16 – 31	rezervováno pro budoucí využití	-

Tabulka 2.5: Bezpečnostní módy WM-Bus podle [7]

Ze sedmi módů, které používají šifrování, dva používají algoritmus DES (*Data Encryption Standard*). Tento algoritmus, standardizovaný v roce 1977, používá klíč o velikosti 56 bitů a v roce 2005 byl Americkým úřadem pro standardizaci (NIST) označen jako zasta-

ralý[3]. Specifikace WM-Bus také označuje DES jako zastaralý [7] a zbytek práce se jimi nebude zabývat.

V Bezpečnostních módech 5 a 3 je každému zařízení přidělen jeden konstantní klíč. Ten má být dle specifikace jedinečný pro každé zařízení [7]. Při využití módu 7 je klíč obměňován. Podrobný rozbor často se vyskytujícího módu 5[4] – AES-CBC s IV – se nachází v následující kapitole.

2.4 Šifrovací algoritmus AES–128

AES, *Advanced Encryption Standard*, je algoritmus pro symetrické šifrování dat. Byl definován Americkým úřadem pro standardizaci (NIST) v roce 2001, na základě šifrovacího algoritmu *Rijndael* [13]. AES je bloková šifra, používající blok o velikosti 128 bitů (16 bytů). To znamená, že data je potřeba před zašifrováním zarovnat na násobek této velikosti (anglicky *padding*). Velikost šifrovacího klíče pro AES může být 128, 192, nebo 256 bitů. V protokolu WM-Bus je použit klíč o velikosti 128 bitů.

Při šifrování algoritmem AES je možno použít **inicializační vektor** (IV), který v některých ohledech zvyšuje bezpečnost šifry. Velikost IV je stejná, jako velikost bloku (tedy 16 bytů) a může být přenášen nešifrovaně spolu s šifrovanými daty. Využití IV v prvním bloku dat je následující:[5]

$$PT = AES(CT, \text{klíč}) \oplus IV$$

A pro zašifrování (AES je symetrická šifra):

$$CT = AES(PT \oplus IV, \text{klíč})$$

AES definuje několik módů:

- **CBC** (*Cipher Block Chaining*): Tento mód je použit protokolem WM-Bus v bezpečnostním módu 5. Zajišťuje, aby dva bloky stejných dat v jedné zprávě nebyly nebyly identické v šifrované podobě.
- **ECB** (*Electronic Code Book*): Při použití ECB to není zajištěno, a tedy není považován za bezpečný [16]. Je však paralelizovatelný a tedy potenciálně rychlejší. Při šifrování jednoho bloku dat je výsledek identický s CBC. Toho je využito při návrhu v další kapitole.
- existují další módy, jako **GCM** a **CCM**, ty však nejsou v této práci diskutovány.

Jednou z významných vlastností AES je jeho odolnost vůči tzv. *known plaintext* útokům. *Known plaintext* útok se nazývá situace, kdy útočník zná zašifrovanou, i čistou podobu dat a na jejich základě je schopen získat klíč[6]. Kdyby byl proveditelný, byl by tento typ útoku dobře použitelný v případě WM-Bus – přenášená data jsou omezená sémantikou aplikační vrstvy.

Při útoku tzv. hrubou silou na AES–128, tedy vyzkoušením každého možného klíče, je potřeba provést až 2^{128} pokusů. Při $5 \cdot 10^{18}$ pokusech za sekundu, což je přibližný výkon

[3] https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard

[4] Podle vzorku měřících přístrojů z [12] (z roku 2013) a vzorku přístrojů zakoupených VUT pro výzkum.

[5] PT – PlainText, nešifrovaná data, CT – CipherText, šifrovaná data, $\oplus$ – binární operace XOR

[6] za dobu kratší, než trvá útok hrubou silou

sítě pro těžení Bitcoinů v roce 2017, by nalezení správného klíče trvalo $2^{128}/2^{62,117} = 2^{65,883}$ sekund, tj. přibližně $2,158 \cdot 10^{12}$ let[7] [8].

Dosud nejefektivnější publikovaný útok na AES se nazývá *Biclique attack* a náročnost útoku hrubou silou na AES–128 dokáže zredukovat přibližně čtyřnásobně, což však stále znamená $2^{126,1}$ pokusů [11].

Z důvodu praktické neproveditelnosti útoku hrubou silou se práce zabývá také tzv. *slovníkovým útokem* (*dictionary attack*), jehož cílem je omezit množinu potenciálních klíčů předpokládáním určitých vlastností klíče (například „binární hodnota klíče reprezentuje ASCII řetězec s názvem výrobce“).

[7] Statisticky je průměrná doba nalezení správného klíče poloviční, pokud předpokládáme náhodný klíč.

[8] https://crypto.stackexchange.com/questions/48667/how-long-would-it-take-to-brute-force-an-aes-128-key

Kapitola 3

Nástroje pro obnovu hesel

Nástroje pro obnovu hesel[1], anglicky *password recovery tools* či méně formálně *password crackers*, jsou počítačové programy, jejichž účelem je získat původní data (*plaintext*, *cleartext*) ze zašifrovaných dat (*ciphertext*)[2] [3].

Typickým vstupem takového nástroje jsou šifrovaná data a množina možných klíčů (hesel). Nástroj zkouší data dešifrovat každým jednotlivým klíčem, dokud nenarazí na správný klíč, případně do vyčerpání klíčů.

Nástroje pro obnovu hesel často využívají grafických jednotek (*GPU - graphics processing unit*), *FPGA* (*Field-Programmable Gate Array*) a dalších typů akceleračního hardware pro zrychlení výpočtů, jelikož umožňují silnou paralelizaci a optimalizaci pro jednu specifickou úlohu (zde šifrování, dešifrování či výpočet hash).

Důležitou vlastností nástrojů pro obnovu hesel je mimo jiné škálovatelnost, tedy možnost spuštění na více fyzických zařízeních, například na více grafických kartách zapojených do jednoho počítače, nebo na síti samostatných počítačů.

V následujících podkapitolách jsou představeny některé z často používaných nástrojů. Kapitola je zakončena zdůvodněním výběru programu, pro který bude plugin navržen a implementován.

3.1 Množina kandidátních hesel

Existuje několik způsobů, kterými nástroj může vybírat *kandidátní hesla* (to jsou hesla, která jsou vybrána – vygenerována, přečtena ze seznamu,. . . – k tomu, aby byla vyzkoušena). Nejjednodušší z nich je útok hrubou silou (*bruteforce attack*), to znamená vyzkoušení všech možností. Útok hrubou silou bývá obvykle prakticky neproveditelný kvůli velkému množství možných klíčů, a proto existují způsoby, které vybírají nejpravděpodobnější klíče.

Jedna z nich je tzv. slovníkový útok (*dictionary attack*), při kterém uživatel specifikuje seznam klíčů k vyzkoušení. Slovníkový útok lze dále rozšířit kombinováním jednotlivých klíčů dohromady (anglicky *combinator attack*), různými permutacemi a variacemi pomocí takzvaných pravidel (*rule-based attack*).

[1] dále v této práci jsou termíny „heslo“ a „klíč“ používány jako synonyma. Z hlediska Wireless M-Bus se používá spíše označení klíč, jelikož používá symetrické šifrování, v terminologii Hashcatu je spíše zvykem používat pojem heslo.

[2] Také pojem „šifrovaná data“ je používán v této práci zaměnitelně s termíny jako „hash“ nebo „zahashovaná data“. Obecně je však mezi nimi rozdíl. Vzhledem k tomu, že v kontextu Hashcatu se pracuje více s hash, je v jeho terminologii často používán pro obojí. Naproti tomu Wireless M-Bus používá jen šifrování.

Další variantou je *mask attack*, který vytváří kandidátní klíče na základě masky. Příkladem budiž maska `%u%u%d`, při které jsou vyzkoušeny všechny kombinace dvou velkých písmen (`%u`), následovaných desítkovou číslicí (`%d`).

Pro AES–128 (a tedy WM-Bus) je množina klíčů dána algoritmem, který stanovuje pevnou délku klíče na 128 bitů (tj. 16 bytů). Z toho vyplývá, že celá množina má velikost 2^{128} klíčů. Klíč je pro AES–128 v binární formě, může tudíž nabývat jakékoli sekvence jedniček a nul, na rozdíl od jiných algoritmů, které například připouštějí jen klíče ve formátu ASCII[3].

3.2 Nástroj Hashcat

Hashcat je Open Source (s licencí MIT) nástroj pro obnovu klíčů [3]. Jeho autorem je Jens „atom“ Steube, který jej začal vyvíjet v roce 2009. Oficiální repozitář je hostován na platformě GitHub.

Hashcat podporuje běh na více zařízeních v rámci jednoho počítače zároveň, a to jak CPU, tak GPU a FPGA (pomocí OpenCL[4] a CUDA). Hashcat je dostupný pro operační systémy GNU/Linux, OS X a Windows. Příklad výstupu od spuštění po úspěšné nalezení hesla je na obrázku 3.1.

Mezi výhody, které využití programu Hashcat nabízí pro tuto práci, patří jednotný formát vstupu pro různé šifry, zajištění co nejlepší využití (*utilizace*) dostupného hardware, již zabudované generátory slovníků, a v neposlední řadě uživatelská přívětivost, ještě zesílená dostupností mnoha uživatelských příruček a tutoriálů.

Pro méně zkušené uživatele může být nevýhodou Hashcatu absence grafického uživatelského rozhraní. Dalším specifikem, které může působit jako nevýhoda, je přijímaný formát. Ten je očekáván ve formě redukované pouze na informace potřebné k obnově (typicky zahashovaná data a další informace, jako kryptografická sůl, inicializační vektor, část nešifrovaných dat,...) a tak je často potřeba data naformátovat samostatným skriptem před předáním Hashcatu.

Hashcat nabízí tyto módy útoku (*attack modes*) [3]:

- Dictionary Attack – klasický slovníkový útok. Pokud uživatel nespecifikuje soubor se slovníkem, jsou kandidátní klíče čteny ze standardního vstupu (*stdin*). Díky tomu lze Hashcat použít s externím generátorem slovníku. Tento typ útoku lze navíc kombinovat s pravidly (anglicky rules),
- Combinator attack – slovníkový útok, ve kterém jsou vyzkoušeny všechny kombinace spojení dvou hesel ze slovníku,
- Mask Attack – množina kandidátních hesel je specifikována maskou, ve které se mohou vyskytovat zástupné znaky, jako například `%d` pro desítkové číslice.
- Hybrid Attack – kandidátní hesla jsou brána ze slovníku, na jejichž začátek nebo konec je přidán výstup útoku maskou,
- Association attack – ke každému hash je navíc specifikována nápověda (anglicky hint), na jejímž základu lze omezit kombinace. Tento mód je v době vytváření práce nedostatečně dokumentován.

[3] Například klíče pro WPA–EAPOL–PMK .

[4] viz následující podkapitolu

Hashcat ukládá nalezená hesla do tzv. *potfile*, což je soubor v adresáři, ze kterého je Hashcat spouštěn. Obsahuje záznamy o obnovených heslech spolu s odpovídajícími klíči. Před spuštěním samotného procesu obnovy Hashcat porovná načtené hash s tímto souborem a případně upozorní na již obnovené hash, které vyřadí ze seznamu právě obnovovaných. Funkce potfile může být nežádoucí při vývoji nového pluginu, proto ji lze vypnout přepínačem `-potfile-disable`.

```
hashcat (v6.2.1) starting...

CUDA API (CUDA 11.3)
====================
* Device #1: NVIDIA GeForce RTX 2080 Ti, 10137/11264 MB, 68MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Precompute-Init
* Early-Skip
* Not-Iterated
* Prepended-Salt
* Single-Hash
* Single-Salt
* Brute-Force
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1100 MB

e983672a03adcc9767b24584338eb378:00:hashcat

Session..........: hashcat
Status...........: Cracked
Hash.Name........: SolarWinds Serv-U
Hash.Target......: e983672a03adcc9767b24584338eb378:00
Time.Started.....: Sun May 23 11:43:13 2021 (1 sec)
Time.Estimated...: Sun May 23 11:43:14 2021 (0 secs)
Guess.Mask.......: ?a?a?a?a?a?at [7]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 24620.9 MH/s (32.19ms) @ Accel:32 Loops:1024 Thr:1024 Vec:1
Recovered........: 1/1 (100.00%) Digests
Progress.........: 31606272000/735091890625 (4.30%)
Rejected.........: 0/31606272000 (0.00%)
Restore.Point....: 0/857375 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:35840-36864 Iteration:0-1024
Candidates.#1....: 4{,erat -> cyr ~}t
Hardware.Mon.#1..: Temp: 62c Fan: 31% Util:100% Core:1920MHz Mem:7000MHz Bus:16

Started: Sun May 23 11:43:12 2021
Stopped: Sun May 23 11:43:15 2021
```

Obrázek 3.1: Příklad výstupu programu Hashcat. Převzato z [3]

Pluginy v Hashcatu

Hashcat již v základu podporuje přes 350 hashovacích a šifrovacích algoritmů [1] a lze jej rozšířit o podporu dalších pomocí pluginů. Jednotlivé hashovací algoritmy (módy) mají přiřazena pětimístná identifikační čísla (*hash-mode number*).

Plugin pro Hashcat sestává z *jádra* (kernelu) a samotného *modulu*. Jeden modul využívá právě jeden kernel, ale jeden kernel může být použit ve více modulech. Identifikační čísla, podle kterých uživatel volí typ hash, jsou přiřazena modulům.

Kernel je v terminologii Hashcatu kód v jazyce *OpenCL*[5], který provádí samotné výpočty hash a vyžaduje největší množství výpočetního výkonu. Kernely pro většinu používaných

[5] https://www.khronos.org/opencl/

šifrovacích algoritmů, včetně kernelu pro AES–128–ECB, jsou již v rámci Hashcatu implementovány. V Hashcatu existuje více typů kernelů, které jsou rozděleny podle následujících aspektů [17]:

- podle rychlosti:
  - *slow kernel*, který je použit, pokud:
    - předpokládaná rychlost kernelu je méně než 10 Mhash/s (milionů hash za sekundu) na GPU, nebo
    - provádí více než 100 iterací nad jakýmkoliv kryptografickým algoritmem.
  - *fast kernel*. Účelem rychlého kernelu je překonat limitaci propustnosti, která vzniká na sběrnici PCI Express. Dále se dělí na:
    - *pure* (čistý) kernel, který podporuje kandidátní klíče libovolné délky.
    - optimalizovaný kernel, který je implementován jako až 18 (velice podobných) kernelů, z nichž polovina jsou *single* kernely, pro běh s jedním hash a druhá *multi*, které obnovují více hash současně. *Single* a *multi* kernely je potřeba dodat ve 3 verzích, 04, 08 a 16[6], které se liší maximální délkou kandidátního klíče[7]. Optimalizované kernely poskytují navíc specifické verze pro různé módy útoku, konkrétně 0, 1 a 3 (módy 6 a 7 sdílí kernel s módem 1).

Slow kernel a *fast kernel* se výrazně liší svou strukturou a o typu vyvíjeného kernelu je potřeba se rozhodnout před začátkem implementace.

OpenCL je programovací jazyk, zaměřený na silně paralelní výpočty [20], syntakticky vychází z jazyka C, specificky z jeho verze C99 [20]. Jeden OpenCL kernel odpovídá jedné funkci z hlediska jazyka C. OpenCL kernely koexistují s *hostitelským kódem*, což je kód vykonávaný běžným, sekvenčním způsobem, typicky běžící na *CPU* (*Central Processing Unit*, procesor). Hostitelský kód vyvolává (*invokuje*) kernely pro paralelizovatelné operace a následně z nich dostává zpět výsledek. Nad rámec jazyka C přidává OpenCL *adresní prostory*, které rozlišují specifičnost dané paměti (kolika kernelům je dostupná). Adresní prostory souvisí s fyzickým umístěním paměti:

- *Private memory* je paměť dostupná pouze danému *work item* (pracovní položce), tedy pro danou invokaci kernelu. Přístup do této paměti je nejrychlejší, je na úrovni registrů.
- *Local memory* je sdílená v rámci *workgroup* (pracovní skupiny, což jsou seskupení pracovních položek). Fyzicky se nachází v rámci výpočetního zařízení, například GPU.
- *Global memory* je paměť sdílená všemi pracovními skupinami.
- *Host memory* je paměť *hostitelského kódu*, typicky se nachází v RAM (*Random Access Memory*) počítače a doba přístupu z výpočetního zařízení je tedy velmi dlouhá.

Druhá část, **modul**, obstarává přípravu hash a dalších dat pro kernel. To typicky zahrnuje dekódování, *tokenizaci* (rozdělení vstupu na jednotlivé hodnoty, například rozdělení na šifrovaná data a inicializační vektor) a naplnění připravených datových struktur. Moduly jsou implementovány v jazyce C a Hashcat nabízí 68 [17] funkcí, které může modul

[6] číslo udává maximální délku ve 4bytových slovech. Kernel 04 tak například zpracovává hesla až do délky 16 bytů

[7] pro kernely, kde určité délky klíčů nedávají smysl, jsou ponechány dané implementace prázdné

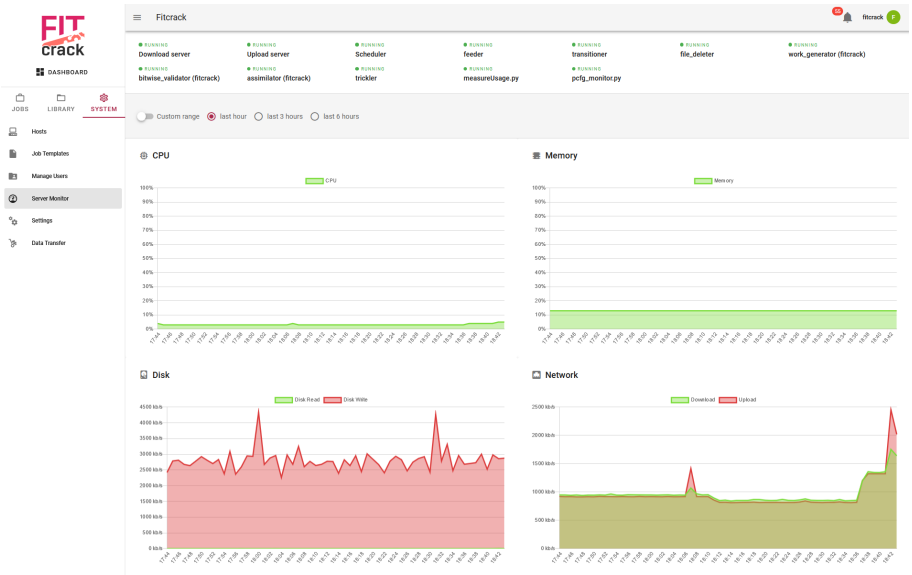
implementovat, z nichž 2 je nutné implementovat. Zbylé funkce určují parametry modulu, například který kernel modul využívá, minimální a maximální délku hesel a podobně. Většina z funkcí má podobu tzv. *getteru* (funkce, které pouze vrací předpřipravenou hodnotu, neprovádí výpočty).

Jednou z funkcí modulu je také poskytnout data pro *self-test*, což je procedura při spouštění Hashcatu, která na dodaném hash a heslu ověří správné fungování pluginu. Filosofií self-testu je ověřit, že plugin vyhodnotí odpovídající heslo jako správné a nemělo by tak dojít k situaci, kdy plugin správné heslo reálných dat vyzkouší, ale neoznačí za správné, například kvůli špatně fungujícímu ovladači grafické jednotky, čímž by mohl uživatel ztratit mnoho hodin či dní výpočetního času a mylně se domnívat, že správné heslo se ve vyzkoušené množině nenachází.

Přidělování identifikátorů (*hash-mode numbers*) probíhá sekvenčně, od vyhrazeného módu 00000 a s krokem 100 (díky tomuto kroku lze vytvářet skupiny podobných módů, jako například pro různé šifrovací módy Wireless Meter Bus). Rozsah 90000 až 100000 je vyhrazen pro volné použití, například testování modulů a vývojáři Hashcatu zajišťují, že moduly v tomto rozsahu nebudou v oficiálním repozitáři a nastanou tak kolize [17].

Nakonec je potřeba zmínit problematiku tzv. *soli* (anglicky *salt*). Hashcat nabízí dvě varianty, `salt_t` a `esalt`. `salt_t` je datová struktura fixní délky vhodná pro obecné použití. Obsahuje hlavní buffer o velikosti 256 bytů a několik dalších (z nichž některé jsou z hlediska Hashcat pluginu *read-only*, tzn. jen pro čtení). Naproti tomu, struktura a velikost `esalt` je ponechána na rozhodnutí autora pluginu. Pokud modul používá esalt, musí specifikovat její velikost pomocí funkce `module_esalt_size`. Využít esalt je doporučeno v případě, kdy plugin pracuje s inicializačními vektory nebo šifrovanými daty (ne hashovanými daty). V případě použití `esalt` je potřeba nastavit `salt_t` na náhodnou hodnotu s dostatečnou entropií, jelikož podle ní Hashcat načtené hash kategorizuje a řadí.

Systém Fitcrack



Obrázek 3.2: Uživatelské rozhraní programu Fitcrack. Převzato z [2]

Fitcrack je z hlediska této práce nadstavbou nad nástrojem Hashcat. Je postaven na platformě BOINC[8] a byl vyvinut výzkumnou skupinou NES@FIT na Fakultě informačních technologií na VUT v Brně [2]. Fitcrack umožňuje propojit více počítačů do sítě a rozdělit práci mezi ně a nabízí webové uživatelské rozhraní pro její správu. Díky kompatibilitě s nástrojem Hashcat by mělo být možné využít vzniklý plugin pro obnovu klíčů Wireless Meter Bus bez výrazných úprav i v tomto nástroji. Fitcrack nabízí grafické rozhraní v podobě webové stránky, které je ukázáno na obrázku 3.2.

3.3 Nástroj John the Ripper

Jednou z alternativ Hashcatu je nástroj John the Ripper (JtR), jehož vývoj začal již v roce 1996. John the Ripper je Open Source nástroj s licencí GNU GPL, autorem je Alexander Peslyak. Podporuje různé operační systémy, běh na CPU, GPU a také FPGA (*Field-programmable gate array*). Jednou z jeho výhod je automatická detekce šifry použitého hashovacího algoritmu z formátu hash. John the Ripper podporuje čtyři módy útoku: *wordlist mode*, *single crack mode*, *incremental mode*, a *external mode*.) [4] [14]

Pravidla (*rules*) používaná ve slovníkových útocích jsou plně kompatibilní s těmi v programu Hashcat. Na rozdíl od Hashcatu však John the Ripper pravidla zpracovává procesorem, což je značně pomalejší než zpracování v kernelu (typicky na grafické jednotce), které využívá Hashcat.

Po posouzení možných alternativ a jejich výhod pro tuto práci bylo rozhodnuto, že práce bude implementována jako plugin pro Hashcat. Hlavními důvody jsou jeho velká rozšířenost, kompatibilita s nástrojem Fitcrack a detailně zpracovaný článek o vytváření pluginu od autora Hashcatu[9]

[8] BOINC je platforma pro distribuci výpočtů mezi zapojené počítače, https://boinc.berkeley.edu/
[9] https://github.com/hashcat/hashcat/blob/master/docs/hashcat-plugin-development-guide.md

Kapitola 4

Návrh nástroje pro obnovu klíčů

Cílem této kapitoly je seznámit čtenáře s procesem návrhu vytvářeného nástroje, jeho částmi, a s podstatnými rozhodnutími v návrhu.

Dohodou s vedoucím práce bylo rozhodnuto, že výsledný nástroj bude implementován jako plugin pro program Hashcat. Tím je dána struktura navrhovaného nástroje, který se bude skládat z více **modulů**, zpracovávajících šifrovaná data před samotným začátkem procesu obnovy a jednoho **kernelu** (česky *jádra*), jehož úkolem je provádět samotné výpočetně náročné operace dešifrování. Modul a kernel dohromady vytváří **plugin**.

Před zahájením práce na samotném pluginu pro Hashcat byla ověřena funkčnost a proveditelnost celého konceptu jednoduchým programem, pracovně nazvaným *PoC*, tedy *Proof of Concept* (volně přeloženo jako „ukázka proveditelnosti“). Ten byl vypracován v jazyce C s použitím kryptografických funkcí z knihovny OpenSSL. Díky tomu, že je naprogramován klasickým sekvenčním způsobem, bylo na něm možno zkoušet a ověřovat nápady a postupy, které byly následně zahrnuty do pluginu. Má také referneční funkci – lze jím ověřit, že plugin nachází pouze správné klíče. Tato funkce byla v pozdější fázi převedena na implementaci v jazyce Perl.

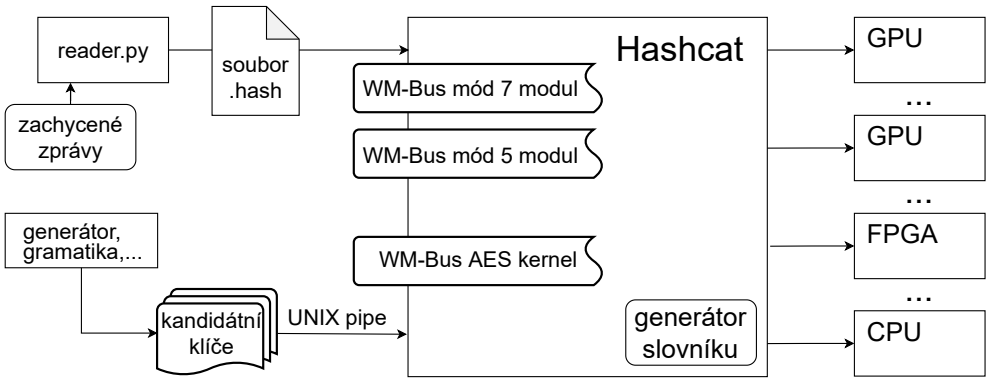
Jednou ze slepých vývojových větví bylo ověřování CRC z linkové vrstvy pro ověření správnosti dešifrování. Vzhledem k tomu, že výpočet CRC je prováděn ze zašifrovaných dat, ukázala se tato metoda jako nepoužitelná.

Šifrování a dešifrování algoritmem AES-128 obecně bylo v průběhu celé práce porovnáváno s online nástrojem *AES – Symmetric Ciphers Online*[1].

Ve složce `hc_wmbus_plugin` jsou umístěny zdrojové kódy pluginu pro Hashcat, ve složce `PoC` testovací implementace v C, zdrojový kód vlastních generátorů kandidátních klíčů je ve složce `wlist_geners`, ve složce `testing_data` se nachází testovací data a složka `hashcat-master` je určena pro zdrojové kódy Hashcatu. V některých složkách jsou také specifické soubory *Makefile* pro kompilaci dané části práce.

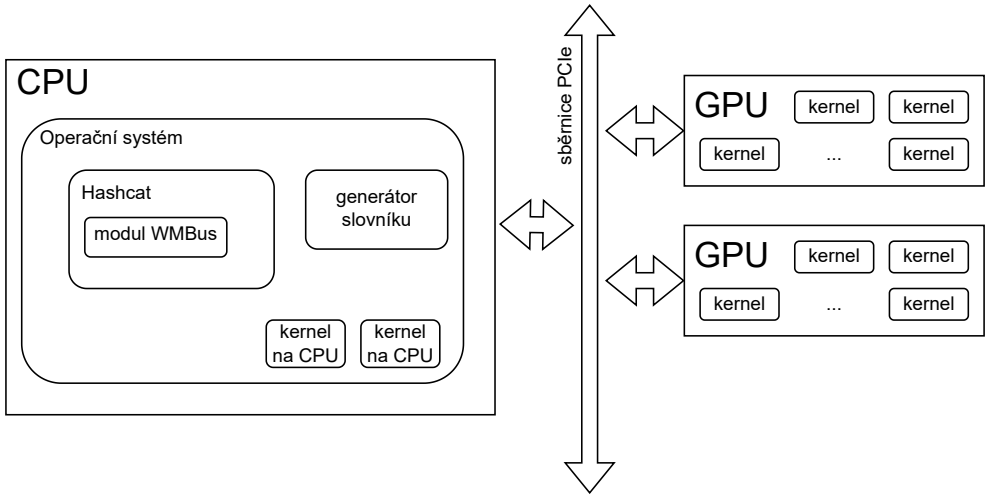
Pro licencování výsledné práce byla vybrána **licence MIT**, především pro shodnost s licencí programu Hashcat a jeho již existujících pluginů a také pro její jednoduchou formulaci. Text použité licence je přibalen ke zdrojovým souborům.

[1] http://aes.online-domain-tools.com/



Obrázek 4.1: Diagram navrhovaného řešení z hlediska vývojáře.

Celkový koncept navrhovaného řešení lze vidět na diagramech 4.1 a 4.2. Diagram 4.1 zobrazuje navrhované řešení z logické úrovně vývojáře a potažmo také uživatele. Rozložení navrhovaného řešení z hlediska hardware je znázorněno na obrázku 4.2 a ukazuje, která část bude spuštěna na kterém typu výpočetní jednotky. Zachycuje modelovou situaci, kdy je navrhované řešení spuštěno na počítači se dvěma grafickými jednotkami, připojenými přes sběrnici PCI Express, a procesor má k disposici dvě volná jádra, na kterých také mohou být spuštěny kernely.



Obrázek 4.2: Diagram navrhovaného řešení z hlediska hardware.

4.1 Návrh generátoru slovníku

Generátor slovníku (anglicky *wordlist generator*) je program (nebo součást nástroje pro obnovu hesel), který po spuštění vypíše klíče k vyzkoušení. Požadované vlastnosti navrhovaného generátoru slovníku jsou:

- **rychlost**. Generátor musí být alespoň tak rychlý, aby nezpomaloval proces obnovy klíčů. To znamená, že musí být rychlejší než program provádějící dešifrování a porovnávání. Tento požadavek platí i pro metody, kterými jsou klíče předávány.

- **možnost nastavit začáteční pozici**. Díky tomu bude možné proces zastavit a poté spustit od předešlého bodu. S touto vlastností souvisí informování o posledním vypsaném klíči před ukončením.
- **variabilita**. Možnost specifikovat různá omezení klíče, například jen na hodnoty v rozsahu latinky, nebo masky, které budou rozgenerovány na konkrétní hodnoty.
- **volba kódování**. Klíče šifry AES–128 jsou binární a je žádoucí, aby generátor slovníku umožňoval pro testovací účely přepínat mezi výpisem v binární podobě a tzv. *hex-ASCII*.

Nabízí se více možností, jak návrh generátoru slovníku pojmout. První, nejjednodušší, je využít již existující možnosti pro generování klíčů v rámci programu Hashcat. Jejich výhodou je rychlé a optimalizované předávání klíčů do části programu, která provádí dešifrování, již hotová implementace a výhodou také je, že potenciální uživatel je již s nimi pravděpodobně seznámen.

Druhou navrhovanou možností je vytvořit vlastní generátor slovníku. V rámci logického strukturování práce je koncipován jako samostatný program, který při spuštění obdrží klíč, od kterého má začít a následně jen sekvenčně vypisuje kandidátní klíče na standardní výstup (*stdout*). Vhodnou, ne však nutnou, vlastností navrhovaného generátoru slovníku je informovat uživatele o posledním vygenerovaném klíči. Vzhledem k tomu, že standardní výstup je posílán do části provádějící dešifrování, je potřeba tuto informaci vypsat na standardní chybový výstup (*stderr*).

4.2 Návrh pluginu pro Hashcat

Jak již bylo zmíněno, Hashcat definuje pro své pluginy závaznou strukturu, zjednodušeně sestávající z kernelu a modulu. Nejvýznamnějším zdrojem při návrhu a implementaci pluginu byla oficiální příručka od autorů Hashcatu, *Hashcat Plugin Development Guide* [17].

Pro vytvářené moduly byla zvolena čísla 99905 a 99907, pro bezpečnostní mód 05, respektive 07. Jelikož v době vývoje byl používán prozatímní prefix 999, používají se dále v této práci pro označení těchto modulů pouze poslední dvě číslice.

Po prostudování zvyklostí v Hashcatu bylo rozhodnuto, že klíče a šifrovaná data budou očekávána v tzv. *hex-ASCII* [7] formátu, což znamená, že budou zapsána *hexadecimálně* (v šestnáctkové soustavě), ale samotné hexadecimální číslice budou ASCII znaky. Výhodou tohoto přístupu je usnadněná čitelnost člověkem a snadné rozlišování speciálních znaků (jako například konec řádku), nevýhodou tohoto přístupu je zdvojnásobení objemu zpracovávaných dat[2].

Před návrhem kernelu a modulu je potřeba se rozhodnout pro jednu z tzv. *solí* (anglicky *salt*), což je v rámci programu Hashcat datová struktura, kterou jsou předávána data z modulu do pluginu. Pro návrh Hashcat pluginu jsou k disposici 2 možnosti, `salt_t` a `esalt`. V této práci je použita esalt, jelikož je vhodnější k předávání dat jako jsou inicializační vektory a šifrovaná data. Esalt má podobu struktury z jazyka C, kterou může autor pluginu definovat dle svých potřeb. Pro navrhovaný plugin musí obsahovat několik rámců šifrovaných dat a jejich inicializační vektory.

[2] hodnoty 0-255 (8 bitů) jsou zakódovány do dvou hexadecimálních číslic, z nichž každá je jeden ASCII znak, tedy $2 \times 8 = 16$ bitů.

Další z datových struktur, používaných v modulu i kernelu, je tzv. *digest* (do češtiny volně přeloženo jako „ochutnávka“). Ten má pevnou velikost 4×32 bitů[3] a je slouží k uchování části hash, která je následně v kernelu porovnávána s hodnotou vytovřého hash. V případě shodnosti obou hodnot je daný klíč označen jako správný pro daný hash.

Kenrel

Přestože již kernel pro šifru AES–128–ECB v rámci Hashcatu existuje (číslo modulu 26401[4]), není v této práci použit. Důvodem je jeho přílišná obecnost a také fakt, že provádí šifrování dat a následně porovnání se zašifrovanými daty, zatímco v případě této práce je potřeba data dešifrovat a poté provést kontrolu validity (nikoli porovnání s jinými daty).

Návrh kernelu v této práci je vázán pevně určenou strukturou kernelu pro Hashcat. Podle doporučení v [17] je založen na prostudování již existujících kernelů, hlavně těch, které mají blízko k navrhovanému: 26401, pro *AES–128–ECB* (základní struktura) a 02501, *WPA–EAPOL–PMK* (problematika *esalt*).

Kernel je navržen jako tzv. *fast kernel*, na základě rychlosti již existujícího kernelu pro AES–128, jehož rychlost se pohybuje v desítkách Mhash/s (milionů hash za sekundu)[5] a předpokladu, že rychlost výsledného kernelu bude řádově stejná.

Vzhledem k fixní délce hledaného klíče, 16 bytů, má smysl implementovat pouze optimalizovaný kernel. Ten se dále dělí na dva samostatné optimalizované kernely, `m` (*multi-*) a `s` (*single-*), první pro crackování více hash současně, druhý pro pouze jeden. Dále se kernely dělí na typy 0, 1 a 3, které se liší typy útoků. Kernel typu 3 je používán pro útok maskou, který nedává pro tuto práci smysl – Hashcat neumožňuje nastavit masku na potřebných 16 bytů a tudíž není implementován.

Na začátku činnosti kernelu je potřeba zpracovat přijatá data. To zahrnuje sdílených proměnných a vytvoření konečné podoby ověřovaného klíče, například aplikováním jednoho z pravidel nebo složením ze dvou částí. Dále je potřeba přetypovat předaný esalt na definovanou strukturu. Po nastavení šifrovacího klíče a dešifrování dat, není, na rozdíl od typické struktury Hashcat kernelu, spuštěno makro pro ověření úspěšnosti dešifrování, ale provedeno vlastní porovnání, a v případě úspěchu jsou postupně dešifrovány další zprávy. Pouze v případě, že jsou všechny dodané zprávy dešifrovány správně, je spuštěno makro, aby Hashcat právě zpracovávané heslo vyhodnotil jako správné.

Modul

Pro navržené řešení jsou potřeba dva moduly, které budou mít společný kernel. Jejich struktura je dána rozhraním, které Hashcat pro moduly poskytuje.

V modulu je nutné implementovat funkci, která klíč v případě jeho nalezení vypíše uživateli se šifrovanými daty. Pro výstup byl navržen formát `prvních 16 B první zprávy : první 2 byty IV : klíč`. Z těchto informací lze jednak dešifrování replikovat, jednak jsou dostatečně unikátní pro jednoznačnou identifikaci hash. Zbylé části zpráv a IV nejsou vypisovány, čímž je ušetřeno místo v datové struktuře esalt, které by jinka bylo nutné pro jejich uložení a výastup programu není zaplavován zbytečně dlouhým textem.

[3] Jelikož těchto 128 bitů je dostatečně velká hodnota pro zajištění unikátnosti a oproti použití celé hodnoty hash šetří přenosové pásmo mezi hostitelem a kernely.

[4] https://github.com/hashcat/hashcat/blob/be75e4b4eae6e294cfea1f5caecd86976ad53b6d/OpenCL/m26401_a0-optimized.cl

[5] na počítači, kde probíhal vývoj, viz kapitolu 6.1.

Testovací implementace

Mimo kernel a modul je nepovinnou součástí pluginu také testovací implementace v jiném jazyce, typicky v některém vysokoúrovňovém jazyce, jako Perl nebo Python. Jejím účelem v rámci testovacího rámce Hashcatu je vytvářet data, kterými bude modul testován a provádět vyhodnocení úspěšnosti.

4.3 Optimalizace algoritmu

Během návrhu bylo zjištěno, že některé části algoritmu a předávaná data jsou pro účely obnovy klíčů zbytečná. Cílem této podkapitoly je identifikovat a navrhnout lepší (primárně rychlejší) řešení.

Nejpodstatnější změna oproti prvotnímu návrhu proběhla v samotném dešifrování. Jelikož kernel vyhodnocuje úspěšnost podle prvních dvou bytů dešifrovaných dat, stačí, aby byl dešifrován pouze první blok dat (16 bytů). AES–128 je bloková šifra a v případě módu AES-CBC (*Cipher Block Chaining*), použitého v bezpečnostním módu 5 Wireless Meter Busu je výsledek šifrování každého bloku dat ovlivněn předchozími bloky, což zpomaluje šifrování a dešifrování, jelikož s jednotlivými bloky nelze počítat paralelně, jako tomu je v módu AES-ECB (*Electronic Codebook*). Ušetřením výpočetně nejnáročnější části lze značně urychlit celý proces.

Další výraznou optimalizací je změna pořadí operace XOR a samotného dešifrování:

$$2F2F_H = XOR(\text{AES–128}(data), IV)$$

je díky komutativitě operace XOR ekvivalentní s

$$XOR(IV, 0x2F2F_H) = \text{AES–128}(data)$$

Použitím tohoto principu lze provést XOR jen jednou na inicializačním vektoru a konstantě $2F2F_H$ a výsledek následně porovnávat s jednotlivými výsledky volání AES–128. Tím je ušetřena operace XOR pro každou invokaci kernelu. Přestože se inicializační vektory pro jednotlivé zprávy liší, první 2 byty jsou vyplněny identifikátorem výrobce, který je pro daný měřící přístroj konstantní.

Zrychlení lze docílit také zmenšením struktury předávaných dat z modulu do kernelu (struktura esalt), jejíž velikost je konstantní. Rozhodnutím omezení počtu šifrovaných zpráv na 5 lze omezit objem dat, pokaždé předávaných do kernelu. Přitom pravděpodobnost tzv. *false positive* (označení klíče jako správného, ačkoli správný není) je

$$\frac{1}{2^{16\times 5}} = \frac{1}{2^{80}}$$

(5 zpráv, pro každou z nich je pravděpodobnost správného dešifrování 2^{16}, což je pravděpodobnost, že začíná konstantou $2F2F_H$). Velikost množiny klíčů pro AES–128 je 128 bitů, takže statisticky v ní lze očekávat více než 1 klíč, který bude označen za správný. To je však vzhledem k velikosti množiny a v současnosti dosahovaným rychlostem dešifrování únosné za cenu zrychlení. V případě, že Hashcat nalezne klíč, je ponecháno na uživateli, aby ověřil jeho správnost na zbylých zprávách (například spuštěním s jinými pěti zprávami, či jiným nástrojem). Rozhodnutí použít právě 5 zpráv je kompromisem mezi co nejmenší pravděpodobností výskytu false positive a minimalizací přenášených dat.

Druhou optimalizací, provedenou na struktuře esalt, je uložení jen jednoho inicializačního vektoru. Přestože podstatou inicializačního vektoru je jeho proměnlivost, první dva

byty mají sémantiku výrobce zařízení. Lze předpokládat, že uživatelem dodané zprávy pochází z jednoho měřícího přístroje a tudíž mají první dva byty jejich IV identickou hodnotu. Toho je využito k dalšímu omezení velikosti esalt.

Výsledkem těchto optimalizací je, že esalt bude obsahovat jen 5×16 bytů šifrovaných dat, 2 byty inicializačního vektoru, upravené operací XOR s prefixem $2F2F_H$ a informaci o počtu dodaných zpráv.

Pro navrhovaný plugin musí obsahovat prvních 16 bytů šifrovaných dat[6] a první 2 byty inicializačního vektoru.

[6] jelikož AES provádí šifrování po 16bytových blocích

Kapitola 5

Implementace nástroje pro obnovu klíčů

V této kapitole je čtenáři představena cesta, která vedla od návrhu z předešlé kapitoly k funkčním, spustitelným pluginům pro Hashcat. Jsou v ní diskutovány implementační detaily a zajímavé úseky kódu tak, aby čtenář nabyl znalostí o fungování vytvářeného pluginu a generátorů slovníku. Je rozdělena na čtyři podkapitoly, první uvádí celkový pohled na vytvořený plugin, další odpovídají třem logickým částem vytvořené práce.

5.1 Integrace do nástroje Hashcat

Výsledné dva pluginy jsou implementovány jako šest kernelů, dva moduly a dvě referenční implementace v jazyce Perl. Dále byly vytvořeny dva generátory slovníku.

```
python3 ../wlist_gener/wlist.py 0102030405060708090A0B0C0D0E0E00 | ../hashcat-master/hashcat --potfile-disable --self-test-disable -w 4
       --hex-charset -m 99905 ../testing_data/bh13_68_hash
hashcat (v6.2.5) starting

[wlist] Starting on key 0102030405060708090a0b0c0d0e0e00
OpenCL API (OpenCL 3.0 ) - Platform #1 [Intel(R) Corporation]
============================================================
* Device #1: Intel(R) HD Graphics 520 [0x1916], 4672/9460 MB (2047 MB allocatable), 24MCU

OpenCL API (OpenCL 2.0 pocl 1.8 Linux, Release, RELOC, LLVM 13.0.0, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl project]
============================================================================================================================
* Device #2: pthread-Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, skipped

hashcat-master/OpenCL/m99900_a0-optimized.cl: Pure kernel not found, falling back to optimized kernel
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 32

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1428 MB

Starting attack in stdin mode

5923c95aaa26d1b2e7493b013ec4a6f6:9315:0102030405060708090a0b0c0d0e0f11
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 99905 (Wireless Meter Bus, Mode 5 (data:IV:data:IV:...))
Hash.Target......: 5923c95aaa26d1b2e7493b013ec4a6f6:9315
Time.Started.....: Mon May 9 23:40:22 2022 (1 sec)
Time.Estimated...: Mon May 9 23:40:23 2022 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Base.......: Pipe
Speed.#1.........: 6766.7 kH/s (36.49ms) @ Accel:256 Loops:1 Thr:128 Vec:4
Recovered.Total..: 1/1 (100.00%) Digests
Progress.........: 786432
Rejected.........: 0
Restore.Point....: 0
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[0102030405060708090a0b0c0d0e0e00] -> $HEX[0102030405060708090a0b0c0d1a0dff]
Hardware.Mon.#1..: N/A
Started: Mon May 9 23:40:16 2022
Stopped: Mon May 9 23:40:25 2022
[wlist] last tried: 0x102030405060708090a0b0c0d1a1798 (2.327496700494558e-31 %)
```

Obrázek 5.1: Výstup spuštění pluginu s ukázkovými daty.

Za digest, hodnotu předávanou z modulu do kernelu, která slouží k porovnávání s generovanými hash, bylo dosazeno prvních 16 bytů první šifrované zprávy. Dle [17] je toto vhodná metoda pro pluginy pracující se symetrickým šifrováním, jelikož šifrovaný text má dostatečně velkou entropii.

Pro předávání dat mezi modulem a kernelem je použita datová struktura na obrázku 5.2. Tuto strukturu je nutné definovat v kódu každého modulu a každého souboru s kernely a vývojář pluginu musí zajistit, že jsou shodné. To je z důvodu, že kód modulu a kód kernelu jsou zkompilovány samostatně překladačem jazyka C, respektive OpenCL, a Hashcat nemůže zajistit provázání, které by jednou definovanou strukturu zpřístupnilo oběma částem.

```
#define MAX_CAPTURES 5
// 4 B -> 128 b,
// because only 1st block of data matters (for the 0x2F2F prefix)
#define MAX_CAPTURE_SIZE 4

typedef struct wmbus_aes {
    u32 encrypted_data[MAX_CAPTURES][MAX_CAPTURE_SIZE];

    // only first 2 Bytes of IV are needed
    // those consist of manufacturer ID,
    // which should be same among specified messages
    u32 IV;

    u32 frames_cnt;

} wmbus_aes_t;
```

Obrázek 5.2: Struktura používaná pro předávání dat jako tzv. esalt

Mimo moduly a kernely potřebné pro funkcionalitu výsledného pluginu byly vytvořeny také dvě testovací implementace v jazyce Perl, jedna pro každý modul. Kód těchto implementací je založen na té módu 26401[1]. Obsahují tři funkce:

- `module_constraints` specifikuje omezení pro délky kandidátních klíčů a generovaných kryptografických solí,
- `module_generate_hash` na základě hesla a soli vytvoří hash (tedy obrácený proces k činnosti kernelu), na kterém může následně testovací framework Hashcatu ověřovat správnou funkčnost pluginu. V případě Wireless Meter Bus jsou kromě klíče a soli potřeba navíc data k zašifrování (plaintext), což je řešeno dosazením konstanty $2F2F_H$,
- `module_verify_hash` přijme řádek výstupu Hashcatu, ve kterém je hash, inicializační vektor a klíč, který plugin označil za správný, a pomocí funkce `module_generate_hash` ověří, že daný klíč je skutečně správný.

Kód jiných kernelů, modulů a testovacích implementací je velmi řídce komentován, tudíž je pro udržení konzistence vytvořený kód záměrně také komentován sporadicky (komentáře byly přidány hlavně v částech, ve kterých se od nich liší). Naproti tomu kód Proof of Concept v jazyce C je komentován mnohem více.

Práce byla vyvíjena s nejnovější verzí programu Hashcat[2] a není kompatibilní poslední vydanou verzí `6.2.5` z 21. listopadu 2021.

5.2 OpenCL kernel

Jelikož byl vyvíjený kernel navržen jako tzv. rychlý (fast) a optimalizovaný (optimized), bylo potřeba jej implementovat ve více verzích. Ty se liší v těchto vlastnostech:

- single – multi: single kernel zpracovává jen jeden hash a porovnání provádí v registrech, multi kernel na konci provede porovnání se všemi hash, pro které dosud nebyl

[1] https://github.com/hashcat/hashcat/blob/master/tools/test_modules/m26401.pm
[2] v době psaní commit `8d6622ce80cb82b64dfdfe24458797d4088c9e46`

nalezen klíč, prohledáním binárního stromu. Oba kernely jsou implementovány v jednom souboru, liší se písmenem `s` nebo `m` v názvu kernelu, například: `m99900_[s|m]04`.

- mód útoku (attack mode): existují tři verze, 0, 1 a 3, ale byly implementovány pouze 0 a 1, jelikož kernel typu 3 pracuje s maskou, která nemůže mít velikost celých 16 bytů, potřebných pro AES-128. Každá je implementována v odpovídajím souboru s názvem: `m99900_a[0|1|3]-optimized.cl`. Liší se zpracováním kandidátních hesel. Kernel módu 0 přijímá hotová hesla, například přečtená ze slovníku a v případě útoku pomocí pravidel (anglicky rule–based attack) ve smyčce aplikuje pravidla. V módu 1 kernel pracuje se základním (anglicky *base*) heslem, ke kterému v iteracích smyčky přidává druhou část. Kernely módu 3 provádí v rámci iterací rozgenerovávání masky na konkrétní hodnoty.

- Maximální délka kandidátního hesla: podle ní se kernely dělí na tři kategorie: 04, 08 a 16, kde číslo odpovídá maximální délce hesla ve čtyřbytových slovech, kterou kernel zpracovává. Jsou odlišeny číslem na konci názvu kernelu, například `m99900_s[04|08|16]`. Podle příručky [17] mají kernely pro módy útoku implementovat pouze typ 04, jelikož implementace jiných variant nepřináší výrazné zrychlení. Pro mód 3 je potřeba implementovat všechny 3 typy, ale jelikož jsou z větší části stejné, je tato stejná část vložena do funkce, kterou všechny tři kernely volají. Toto řešení je použito v mnoha existujících kernelech[3].

Celkem tak bylo vypracováno šest kernelů. Přestože velká část kódu všech kernelů je shodná, jsou z důvodu maximalizace výkonnosti implementovány samostatně.

Ve vytvořených kernelech je téměř výhradně používán datový typ `u32`, což je celočíselný typ o velikosti 32 bitů bez znaménka. Používání jiných datových typů, i menších, například `u8`, vede ke zhoršení výkonu, jelikož registry současných grafických jednotek jsou právě 32bitové a jiné typy je nutné emulovat [17].

Pro dešifrování dat algoritmem AES–128 jsou použity funkce `aes128_set_decrypt_key` a `aes128_decrypt`, které jsou implementovány v rámci Hashcatu v souboru `inc_cipher_aes.cl`. Jelikož kernel přijímá kandidátní klíče ve formě hex–ASCII, je potřeba je v rámci něj převést do binární podoby, k tomu slouží funkce `hex_to_u32`, která je založena na funkci `hex_to_u8` ze souboru `m02501-pure.cl` a upravena.

Algoritmus 1 zobrazuje postup, aplikovaný na každém kandidátním klíči. Pro zřejmost hodnoty proměnné IV je zde zmíněna operace XOR s konstantou $2F2F_H$, která je v implementaci z důvodu optimalizace provedena již v modulu.

[3] Například kernely v souborech 27900_a3-optimized.cl, m11500_a3-optimized.cl, m27900_a3-optimized.cl a m26401_a3-optimized.cl v adresáři OpenCL.

Algoritmus 1: Algoritmus dešifrování, provedený na každém jednotlivém kandidátním klíči.

Input: hex–ASCII key, IV, encrypted messages
Output: is the key the correct one?

```
1  IV = IV ⊕ 2F2F_H
2  key = hex_to_u32 ( hex–ASCII key )
3  decrypted = aes128_decrypt ( first encrypted message, key )
4  if first 2 bytes of decrypted and IV are equal then
5      for every encrypted message, except the first do
6          decrypted = aes128_decrypt ( encrypted message, key )
7          if first 2 bytes of decrypted and IV do not equal then
8              return False
9          end
10     end
11     return True
12 end
13 return False
```

Tento algoritmus může být v rámci jednoho spuštění kernelu proveden vícekrát, například pro různá pravidla, která kandidátní klíč modifikují. To je řešeno smyčkou *for*, jejíž hlavička má typicky podobu

$$\text{for (u32 il\_pos = 0; il\_pos < IL\_CNT; il\_pos += VECT\_SIZE)}$$

V ní jsou provedeny potřebné modifikace a poté výše zmíněný algoritmus.

Na konci kernelu je použito makro `COMPARE_S_SIMD`, respektive `COMPARE_M_SIMD`, které provede porovnání čtyř 32bitových dešifrovaných hodnot s očekávanými a v případě shodnosti označí právě ověřovaný klíč za správný. Jelikož je tento princip zamýšlen především pro hashovací algoritmy[4], nikoli šifrovací, je použití těchto maker v práci upraveno tak, že pokud jsou klíčem správně dešifrovány všechny dodané zprávy, je makru předána šifrovaná podoba 1. zprávy.

5.3 Moduly

Byly implemontovány dva moduly, jeden pro bezpečnostní mód 5 a jeden pro bezpečnostní mód 7 (viz kapitolu 2.3). Oba moduly používají stejný kernel, hlavní rozdíl je, že modul pro mód 5 pracuje s inicializačním vektorem, zatímco modul pro mód 7 nikoli. Následující popis implementace, s výjimkou popisu práce s inicializačním vektorem, je tudíž platný pro oba moduly.

Nejdůležitější funkcí modulu je dekódovat soubor obsahující šifrovaná data. To se děje tak, že pro každý řádek vstupního souboru je zavolána funkce `module_hash_decode`. Jejím úkolem je parsovat vstupní řádek (k tomu je k disposici *tokenizer*, který jej dokáže rozdělit podle oddělovacích znaků a ověřit některé vlastnosti) a naplnit datové struktury, které budou předávány kernelu, případně také hlásí chyby ve vstupním souboru. Tato funkce, dle optimalizace popsané výše, provádí operaci XOR nad inicializačním vektorem a konstantou $2F2F_H$.

[4] Tento problém je diskutován v kapitole `module_dgst_size()` v [17].

Specifikem implementovaných modulů je, že počet *tokenů* není pevně určen (moduly typicky vyžadují pevnou strukturu, např. data v hex–ASCII, oddělovač a poté kryptografickou sůl, tedy dvě položky oddělené konkrétním znakem). Naproti tomu, v případě vytvářených modulů, může uživatel zadat libovolný počet zachycených zpráv. Tento problém je řešen tak, že jsou nejdříve spočítány výskyty separátoru[5], z jejichž počtu je následně spočítán počet tokenů a teprve předán zpracovávací funkci (tokenizer). Hledání oddělovačů je limitováno na prvních pět, respektive deset[6], další zprávy totiž kernel nepoužívá (viz podkapitolu o optimalizaci 4.3).

Druhou podstatnou funkcí v modulu je `module_hash_encode`, která formátuje hash pro zobrazení uživateli. V případě modulu pro mód 5 jsou za hash přidány první dva byty inicializačního vektoru, jelikož jsou podstatné pro dešifrování. Funkce `module_hash_encode` je volána, když se Hashcatu podaří nalézt správný klíč a má být, spolu s hash, vypsán uživateli.

Další implementované funkce informují Hashcat o vlastnostech modulu. Pro vyvíjený modul byly použity následující:

- `module_attack_exec` – určuje, zda je použit *slow hash*, nebo *fast hash*,
- `module_esalt_size` – velikost datové struktury esalt v bytech,
- `module_hash_category` – kategorie modulu (jako například síťové, databázové šifrovací algoritmy, algoritmy pro šifrování dokumentů,...), pouze pro informační účely.
  - byla použita kategorie `HASH_CATEGORY_NETWORK_PROTOCOL`.
- `module_hash_name` – zobrazovaný název modulu,
- `module_kern_type` – použitý kernel,
- `module_opti_type` – použité optimalizace, specificky v této práci:
  - `OPTI_TYPE_ZERO_BYTE` – optimalizace pro klíče, které jsou z části nulové.
- `module_opts_type` – volby (*options*) pro daný modul, specificky v této práci:
  - `OPTS_TYPE_PT_GENERATE_LE` – maska generuje hesla v little endian.
  - `OPTS]_TYPE_PT_HEX`,
  - `OPTS]_TYPE_ST_HEX`.
- `module_pw_max` – maximální délka klíče,
- `module_pw_min` – minimální délka klíče,
  - minimum i maximum je nastaveno na 32 znaků.
- `module_salt_type` – typ datové struktury salt,
- `module_st_hash` – hash pro self test,
- `module_st_pass` – heslo pro self test,

[5] Pro oddělovač (separátor) je použit výchozí znak, kterým je dvojtečka

[6] V módu 5 má jedna zpráva formát `data:IV`, počet oddělovačů je tudíž dvojnásobný.

- `module_dgst_pos0` – `module_dgst_pos3` – jsou nastaveny na výchozí hodnoty.

Poslední funkcí je `module_init`, která nastavuje do datové struktury `module_ctx_t` ukazatele na konkrétní implementované funkce, čímž tvoří rozhraní mezi programem Hashcat a modulem. Za neimplementované funkce je dosazena hodnota `MODULE_DEFAULT`.

5.4 Generátor slovníku

V rámci práce vznikly dva generátory slovníku: jeden v jazyce Python a jeden v jazyce C. Struktura obou je velmi podobná, liší se co do rozšiřující funkcionality a také svou rychlostí (o tomto viz kapitolu 6), proto bude popsána na implementaci generátoru v jazyce Python, která je kratší a čitelnější (obrázek 5.3). Na obrázku 5.4 jsou uvedeny typické příklady spouštění jednotlivých generátorů.

Generátor slovníku po spuštění přijme přes poziční argument při volání počáteční pozici (*offset*) a případně ji použije. Pokud není zadána, je implicitně nastavena na nulový klíč (všechny bity klíče nulové). Před začátkem samotného generování vypíše na standardní chybový výstup, *stderr* (jelikož *stdin* je typicky přesměrován do Hashcatu nebo jiného programu) informaci o počáteční pozici (z toho důvodu, že často bývá specifikována ve skriptu a nezadává ji přímo uživatel při spuštění). Informace pro uživatele jsou uvozeny textem `[wlist]`, aby bylo zřejmé, odkud jsou vypsány. Poté již následuje samotná hlavní smyčka, ve které program iteruje přes množinu klíčů a vypisuje jednotlivé kandidátní klíče na standardní výstup, oddělené znakem nového řádku (*newline*). Vypisované klíče jsou zarovnány na 32 hexadecimálních číslic, případné nuly jsou přidány z levé strany.

Posledním prvkem programu je try–catch blok, do kterého je hlavní smyčka zabalena. Ten zajišťuje, že v případě přerušení činnosti programu bude uživateli vypsána (opět na standardní chybový výstup) informace o posledním vypsaném klíči.

```
try:
    # first positional arg defines starting offset
    # expects hex number
    START = int(argv[1], 16)
except IndexError:
    START = 0

MAX_KEY = (2 << 127) - 1

i = START
stderr.write("[wlist] Starting on key %032x\n" % i)

try:
    for i in range(START, MAX_KEY):
        print("%032x" % i)

except (BrokenPipeError, KeyboardInterrupt):
    stderr.write("[wlist] last tried: {} ({} %)\n"
        .format(hex(i), ((i - START) / (MAX_KEY - START)) * 100 ))
    pass
```

Obrázek 5.3: Zkrácený kód generátoru slovníku v jazyce Python

Implementace v jazyce C používá místo try–catch bloku registraci obsluhy signálů `SIGINT` a `SIGTERM`. Navíc v ní lze přepínat mezi hex–ASCII a binárním výstupem. To je kvůli rychlosti implementováno pomocí preprocesorové konstrukce if–else. V případě binárního výstupu nejsou jednotlivé vypisované klíče nijak oddělovány a předpokládá se, že program přijímající výstup jej bude dělit po 16 bytech.

Generátor v jazyce C navíc obsahuje 3 funkce, jejichž účelem je omezit monžinu kandidátních klíčů: `same_bytes` vypisuje klíče, ve kterých mají všechny byty stejnou hodnotu, `same_two_bytes` funguje na stejném principu, ale pracuje s dvojicemi bytů a funkce `range` iteruje přes specifikovaný rozsah pro každý byte klíče. Jednotlivé funkce lze zvolit odkomentováním preprocesorového makra na začátku programu.

V rámci práce byly často využity také již existující generátory slovníku, jmenovitě *Maskprocessor*[7] (který byl, alespoň v roce 2019, podle autorů nejrychlejší existující generátor slovníku vůbec[18]) a generátor vestavěný v programu Hahscat, který je z důvodu rychlosti implementován částečně v kernelu.

Maskprocessor umožňuje spuštěním s následujícími argumenty nastavit všech 16 bytů jako masku, v tom případě však neumožňuje nastavit počáteční pozici. V této práci je typicky použita množina znaků `-custom-charset1="?dabcdef"`, která obsahuje číslice 0 až 9 a A až F.

```
# zacne na klici 00000000000000000000000000000000
python3 wlist.py
# zacne od specifikovaneho klice
python3 wlist.py 00000000000000000000ffffffff321

./wlist_gener


./mp64.bin -r 32 --custom-charset1="?dabcdef"
    "?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1"

# ukazka propojeni s~Hashcat
python3 wlist.py 0 | sudo hashcat -m 99901 testing_data/kaden.dump
```

Obrázek 5.4: Příklady spuštění jednotlivých generátorů

Generátory vytvořené v rámci této práce nabízejí malé možnosti omezování množiny kandidátních klíčů, primárně proto, že byly vyvinuty pro experimenty ve fázi, kdy práce nebyla zakomponována jako plugin do programu Hashcat a vznikla potřeba generátoru k testování. Pokročilejší metody generování, jako masky, jsou ponechány na nástrojích v rámci Hashcat, jelikož nemá smysl reimplementovat již dobře fungující nástroje a není to cílem této práce.

[7] https://github.com/hashcat/maskprocessor/

Kapitola 6

Vyhodnocení a testování

Tato kapitola se zabývá vyhodnocením výsledku práce. Zaměřuje se na dva nejpodstatnější aspekty práce – **výkonnost**, či rychlost, a **verifikaci**, ověření, že vytvořený nástroj nalezne správný klíč, pokud se ve specifikované množině klíčů vyskytuje, a v opačném případě jej nenalezne. Od výkonnosti se odvíjí dosažení požadovaného výsledku, totiž nalezení šifrovacího klíče, v přiměřeném čase na aktuálně dostupném hardware.

Výkonnost nástrojů pro obnovu klíčů je udávána v jednotce `hash/s`, což značí počet vypočtených hash (v případě této práce vyzkoušených klíčů) za sekundu. Tato jednotka je dále kombinována se standardními SI prefixy G (giga-, 10^9), M (mega-, 10^6) a k (kilo-, 10^3).

Data použitá pro testování (slovníky i hash) se nacházejí ve složce `testing_data`.

Následující podkapitoly krátce shrnují, co bylo vyhodnocováno, v jakém prostředí a jakými metodami a nakonec uvádí rozbor a interpretaci získaných dat.

6.1 Použitý hardware

Pro účely vývoje a průběžného testování práce byl použit laptop Lenovo Thinkpad L460[1], s procesorem `Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz`, zahrnujícím integrovanou grafickou jednotku `Intel(R) HD Graphics 520`. V laptopu je k disposici celkem 12 GiB operační paměti RAM ve dvou slotech. Vývoj a testování probíhaly v operačním systému `Manjaro Linux` s Linux kernelem verze 5.15.28.

Druhým počítačem, na kterém proběhlo testování práce, byl desktop v laboratoři O204 v Centru výpočetní techniky na FIT VUT. Tento počítač disponuje grafickou kartou Nvidia GeForce GTX 980[2], procesorem Intel Corporation Xeon E3-1200 a 8 GiB operační paměti. Testování proběhlo v operačním systému CentOS Linux 7.9.2009, a verifikace také ve Winows 10. Na tomto počítači nemohla být práce spuštěna na procesoru, pravděpodobně kvůli chybějícím ovladačům nebo absenci běhového prostření OpenCL.

Pro přehlednost a stručnost jsou dále tyto počítače označovány zkráceně – „ThinkPad“ pro ThinkPad L460 a „0204“ pro počítač v laboratoři O204.

[1] https://psref.lenovo.com/syspool/Sys/PDF/ThinkPad/ThinkPad_L460/ThinkPad_L460_Spec.PDF
[2] https://www.nvidia.com/en-us/geforce/gaming-laptops/geforce-gtx-980/

6.2 Metody měření

První nabízející se cestou je využití vestavěné možnosti (*command line switch*) Hashcatu pro **benchmark** (do češtiny volně přeloženo jako „měření výkonnosti“). Její největší předností je spolehlivost výsledků, jelikož měří výkonnost v kontextu celého Hashcatu, tedy je nejblíže skutečné situaci. Další její výhodou je již existující implementace v ustáleném nástroji, což dodává jejím výsledkům důvěryhodnosti.

Další možnost úzce souvisí s první zmíněnou. Je jí **sledování výpisů** Hashcatu, v nichž je udávána aktuální rychlost zpracování pro jednotlivá zařízení a celková rychlost (součet rychlostí všech zařízení) Oproti 1. možnosti se liší vlivem generátoru slovníku na výsledek.

Pro měření rychlostí mimo Hashcat byla použita metoda, kdy po ukončení (typicky přerušení – *interrput* – signálem SIGINT) generátoru slovníku byla na standardní chybový výstup (*stderr*) vypsána **informace o posledním vygenerovaném klíči**. Při znalosti doby běhu (získané pomocí programu `time` vestavěného v systémech UNIX) lze dopočítat rychlost. Tato metoda je málo přesná kvůli *bufferování* (dočasnému ukládání) dat mezi generátorem a programem pro ověřování klíčů a dalším jevům, vyplývajících z přítomnosti operačního systému, které vytvářejí z pohledu práce nedeterminismus.

Další z použitých metod pro operační systémy na bázi UNIX je měření propustnosti *rour* (angl. *pipes*), kterými jsou předávány kandidátní klíče z generátoru do programu pro ověřování klíčů. Touto metodou lze zjistit, která část je rychlejší, než druhá, a o kolik. Pro tuto metodu byl prožit program `tqdm`[3].

Poslední, v tomto případě špatně aplikovatelnou metodou, je *profiling*, česky **profilování**, kódu. Hlavními důvody pro její špatnou použitelnost jsou I) malá dostupnost nástrojů, kde každý výrobce hardware vyvíjí svůj, úzce zaměřený, nástroj, II) nezdokumentované použití s nástrojem Hashcat (autor nedokázal nalézt zdroje k tématu profilování pluginů pro Hashcat a vzhledem k utilizaci hardwaru plně v režii Hashcatu usuzuje, že náročnost zprovoznění není adekvátní vyhodnocovací fázi této práce). Potenciální výhodou použití profilování jsou detailnější vyzískaná data, mimo jiné v podobě analýzy časové náročnosti jednotlivých bloků kódu, což jiné zde zmíněné metody nedokáží. Možným řešením je kód kernelu v OpenCL upravit a spustit jako kód v jazyce C v procesoru, kde je použití profilovacích nástrojů jednodušší.

V případě, že nebylo testováno konkrétní výpočetní zařízení nebo vliv nastavené výkonnosti, byla použita všechna dostupná zařízení a výkonnost byla nastavena na nejvyšší hodnotu, 4.

Kromě přímého měření bylo také použito sledování využití operační paměti, které probíhalo vypisováním aktuálního stavu využití paměti s periodou 2 sekundy.

6.3 Výsledky a vyhodnocení testování

V této podkapitole jsou visualizovány a interpretovány výsledky provedeného měření a testování. Metody měření popsané v předešlé podkapitole byly zakomponovány do skriptu `tests.sh`, který byl následně spuštěn na obou referenčních počítačích. Skript i jeho výstup z obou počítačů je přiložen na paměťovém médiu ve složce `tests`.

Celková doba vykonávání testovacího skriptu byla 25 minut 2 sekundy pro počítač v laboratoři v O204 a 126 minut 56 sekund pro ThinkPad.

[3] https://tqdm.github.io/

Verifikace vytvořených pluginů

V tabulce 6.1 jsou zobrazeny výsledky verifikace vytvořených pluginů. Byly použity dva způsoby testování, self–test a testovací rámec v rámci Hashcatu, `test.sh`. Výsledky platí pro všechna využitelná zařízení daného počítače. Pro PC v laboratoři 0204 a OS Windows byl spuštěn pouze self–test, jelikož se nepodařilo na nich zprovoznit testovací rámec.

Test	Modul	a	Kenrel	ThinkPad	O204	O204, Windows 10
self-test	05	0	single	✓	✓	✓
self-test	05	0	multi	✓	✓	✓
self-test	05	1	single	✓	✓	✓
self-test	05	1	multi	✓	✓	✓
self-test	07	0	single	✓	✓	✓
self-test	07	0	multi	✓	✓	✓
self-test	07	1	single	✓	✓	✓
self-test	07	1	multi	✓	✓	✓
test.sh	05	0	single	✓	?	?
test.sh	05	0	multi	✓	?	?
test.sh	05	1	single	✓	?	?
test.sh	05	1	multi	✓	?	?
test.sh	07	0	single	✓	?	?
test.sh	07	0	multi	✓	?	?
test.sh	07	1	single	✓	?	?
test.sh	07	1	multi	✓	?	?

Tabulka 6.1: Výsledky verifikačních testů.

Rychlost generátorů slovníku

Tento test porovnává rychlosti dvou vytvořených generátorů slovníků a jednoho převzatého, Maskprocessor. Každý generátor byl spuštěn třikrát, na 1 sekundu, 10 sekund a minutu. Bylo zjištěno, že rychlost všech je přibližně konstantní po celou dobu běhu a dále, že generátor implementovaný v jazyce C je oproti tomu v jazyce Python asi šestrkát rychlejší. Překvapenmím je rozdílnost rychlostí generátorů *wlist_gener* a *Maskprocessor* mezi referenčními počítači, kde na počítači v laboratoři O204 dosahovaly srovnatelných rychlostí, zatímco na počítači ThinkPad byl generátor *wlist_gener* oproti *Maskprocessor* výrazně pomalejší. Tabulky 6.3 a 6.2 ukazují naměřené rychlosti na obou referenčních počítačích. Všechny generátory byly nastaveny na hex–ASCII výstup se 32 byty na řádek[4].

Doba spuštění	wlist.py	wlist_gener	Maskprocessor
1 s	45.7	267	417
10 s	45.3	270	411
1 minuta	45.7	267	409

Tabulka 6.2: Naměřené rychlosti generátorů slovníku – ThinkPad. Rychlosti jsou v [MB/s].

[4] A jedním navíc pro znak nového řádku, celkem 33 bytů.

Doba spuštění	wlist.py	wlist_gener	Maskprocessor
1 s	48.6	319	334
10 s	49.7	319	349
1 minuta	48.7	325	348

Tabulka 6.3: Naměřené rychlosti generátorů slovníku – 0204. Rychlosti jsou v [MB/s].

Benchmark v programu Hashcat

V tomto testu byla využita funkce programu Hashcat, tzv. *benchmark* (česky *výkonnostní test*) ke změření dosahovaných rychlostí vytvořených pluginů. Tabulka 6.4 zobrazuje získané hodnoty a pro porovnání také hodnoty pro plugin 26401 (šifra AES–128 ECB), ze kterého tato práce čerpala.

Měření bylo postupně provedeno se všemi možnými hodnotami nastavení výkonnosti (v programu Hashcat přepínač `-w`) pro oba vytovřené pluginy a zvlášť pro spuštění na procesoru (CPU), grafické jednotce (GPU) a CPU i GPU najednou.

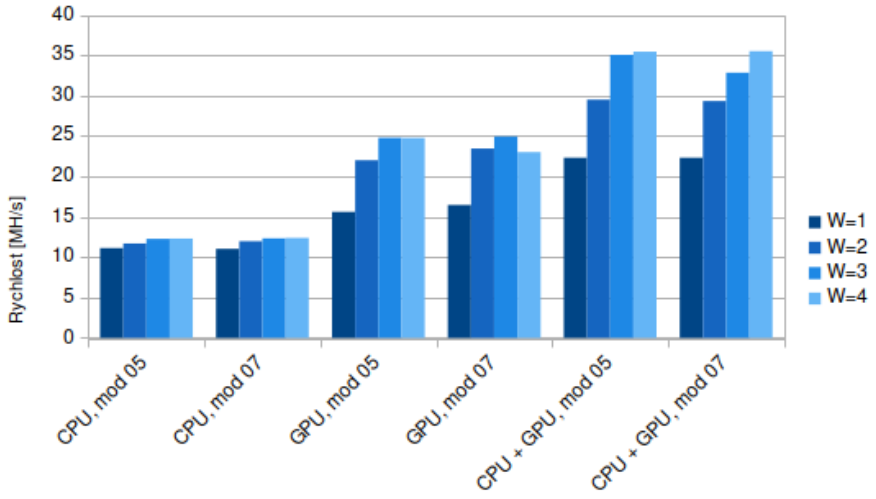
Benchmarkem byly změřeny dvě veličiny, rychlost (anglicky *speed*) a prodleva (anglicky *delay* nebo *lag*). Při spuštění na procesoru a grafické jednotce zároveň nelze měřit prodlevu, jelikož se liší pro jednotlivá zařízení.

Zařízení	M	W	ThinkPad		O204	
			R [kH/s]	P [ms]	R [MH/s]	P [ms]
GPU	-05	1	15606.4	1.33	488.4	2.08
GPU	-05	2	22008.8	8.10	517.9	7.98
GPU	-05	3	24785.2	60.52	518.2	64.49
GPU	-05	4	24769.5	241.00	529.1	505.80
CPU	-05	1	11130.2	1.46	–	–
CPU	-05	2	11703.4	11.07	–	–
CPU	-05	3	12265.4	85.28	–	–
CPU	-05	4	12307.1	340.51	–	–
GPU + CPU	-05	1	22330.5	–	492.6	–
GPU + CPU	-05	2	29521.4	–	514.6	–
GPU + CPU	-05	3	35054.9	–	528.5	–
GPU + CPU	-05	4	35452.1	–	527.3	–
GPU	-07	1	16471.7	1.19	484.4	2.10
GPU	-07	2	23460.1	7.66	513.0	8.05
GPU	-07	3	24946.3	60.35	526.5	63.50
GPU	-07	4	23016.8	254.38	526.6	508.21
CPU	-07	1	11036.8	1.48	–	–
CPU	-07	2	11979.1	10.74	–	–
CPU	-07	3	12337.4	84.80	–	–
CPU	-07	4	12387.3	338.30	–	–
GPU + CPU	-07	1	22326.4	–	484.6	–
GPU + CPU	-07	2	29339.3	–	513.1	–
GPU + CPU	-07	3	32847.6	–	526.4	–
GPU + CPU	-07	4	35511.4	–	526.0	–
GPU	26401	4	64881.3	381.61	1191.2	449.56
CPU	26401	4	19381.6	214.04	–	–

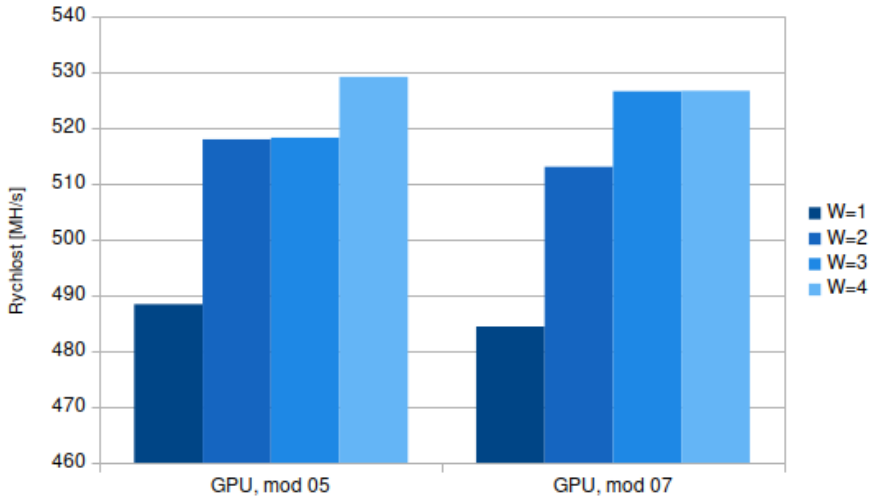
Tabulka 6.4: Výsledky benchmarku v rámci programu Hashcat. „M“ v hlavičce značí modul, „W“ výkonnost „R“ rychlost a „P“ prodlevu.

Získaná data o rychlostech byla vynesena do grafů 6.1 a 6.2. Lze z nich vyčíst, že dosahovaná rychlost roste přibližně lineárně (odchylky jsou způsobeny využíváním zařízení i jinými procesy v operačním systému, jimž nelze zamezit zcela) s nastavenou hodnotou výkonnosti. Také z nich vyplývá, že nastavená výkonnost má větší vliv na rychlost grafických jednotek než procesoru.

Grafy 6.1 a 6.2 také potvrzují domněnku, že rychlosti pluginů –05 a –07 se, při zanedbání šumu, neliší. To odpovídá faktu, že výpočetně náročná činnost je prováděna v kernelu, který mají oba moduly stejný.

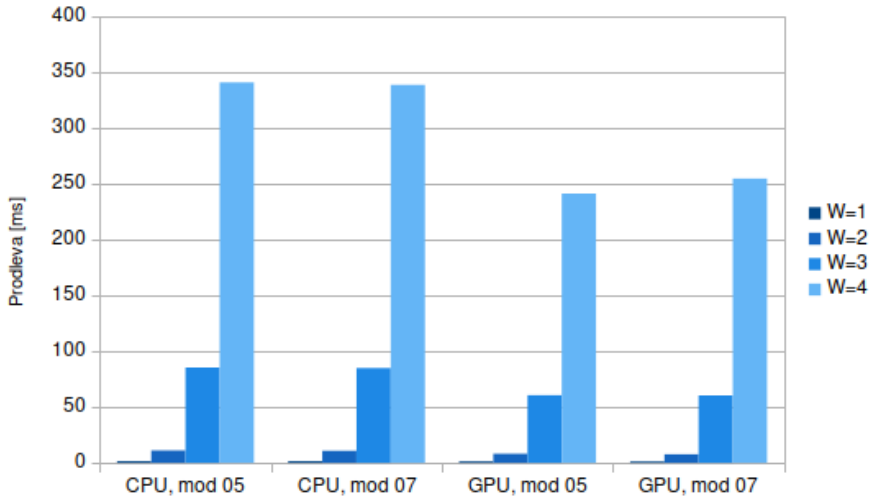


Obrázek 6.1: Porovnání rychlosti pro různá nastavení výkonnosti – ThinkPad.

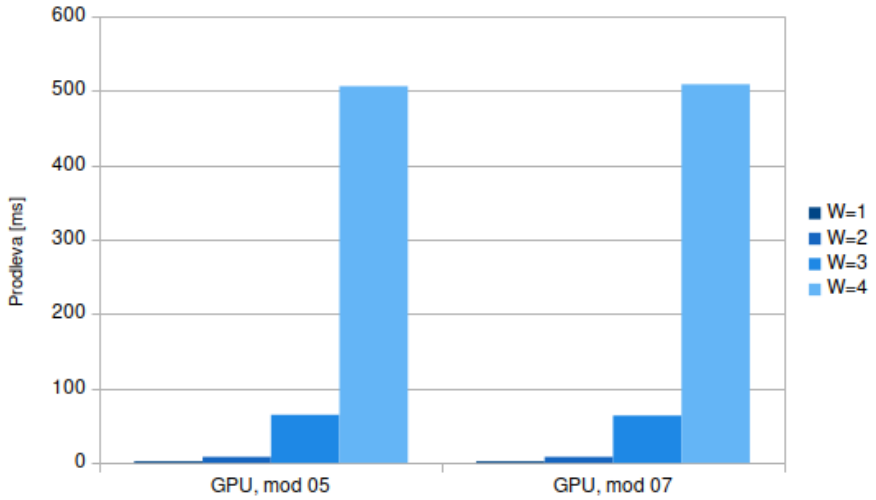


Obrázek 6.2: Porovnání rychlosti pro různá nastavení výkonnosti – O204.

Data o délce prodlevy jsou visualizována v grafech 6.3 a 6.4. Je v nich vidět přibližně exponenciální charakter závislosti. Při nastavení výkonnosti na úroveň 4 je prodleva pozorovatelná uživatelem, což ale vzhledem k typickému použití programu Hahscat není podstatné.



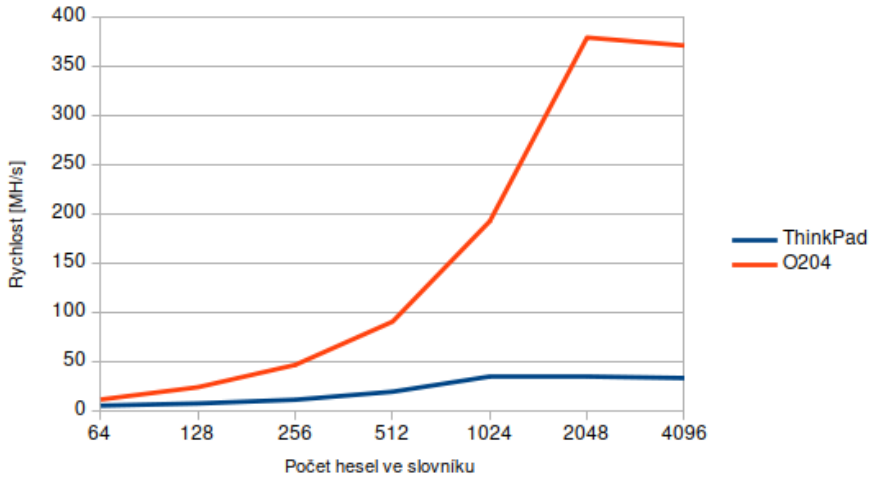
Obrázek 6.3: Porovnání prodlevy pro různá nastavení výkonnosti – ThinkPad.



Obrázek 6.4: Porovnání prodlevy pro různá nastavení výkonnosti – O204.

Porovnání rychlostí pro různé velikosti slovníku

Tento výkonový test měřil vliv velikosti (počtu položek) slovníku na výslednou rychlost. Slovník byl vygenerován předem a vzhledem k jeho velikostem mohl být načten programem Hashcat najednou. Pro prodloužení doby každého spuštění byl slovník kombinován s maskou šesti hex–ASCII číslic. Graf 6.5 ukazuje dosahované rychlosti pro vyzkoušené velikosti slovníku. Vyplývá z něj, že tato závislost má přibližně exponenciální průběh, ale od určité velikosti slovníku se ustálí a dále pokračuje konstantně. To je způsobeno vlastnostmi hardware, proto se tato hranice liší pro referenční počítače.

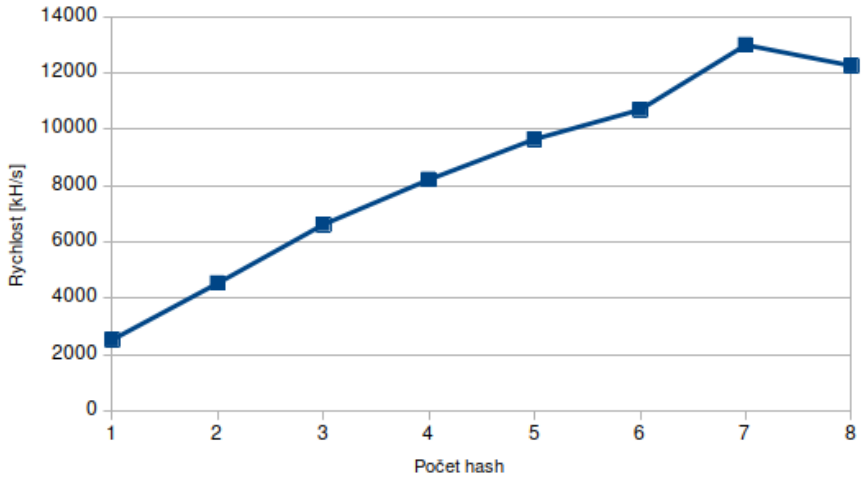


Obrázek 6.5: Rychlosti při různých velikostech slovníku.

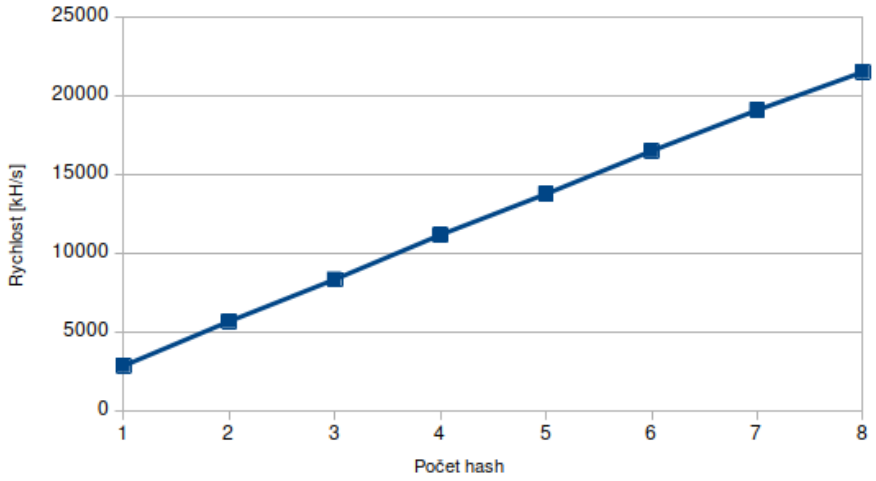
Porovnání rychlostí pro různý počet hash

V tomto výkonovém testu byla měřena závislost dosahované rychlosti na počtu dodaných hash (v případě této práce šifrovaných dat). Za každý hash bylo použito 5 náhodně vygenerovaných WM-Bus zpráv s IV, především pro minimalizaci šance, že v průběhu bude nalezen klíč a tím budou data o rychlosti zkreslena. Každé spuštění trvalo 30 sekund.

Byly provedeny testy pro 1 hash až 8. Při specifikaci jednoho hash je spuštěn single kernel, při dalších multi kernel. Podle visualizace dat v grafech 6.6 a 6.7 je tato závislost přibližně lineární, což nesouhlasí s logaritmickým průběhem použitého vyhledávání v binárním stromu. To je pravděpodobně způsobeno malým počtem vzorků a pří jejich navýšení by graf konvergoval k logaritmickému průběhu.



Obrázek 6.6: Rychlosti při různých počtech hash (zpráv s různými klíči) – ThinkPad.

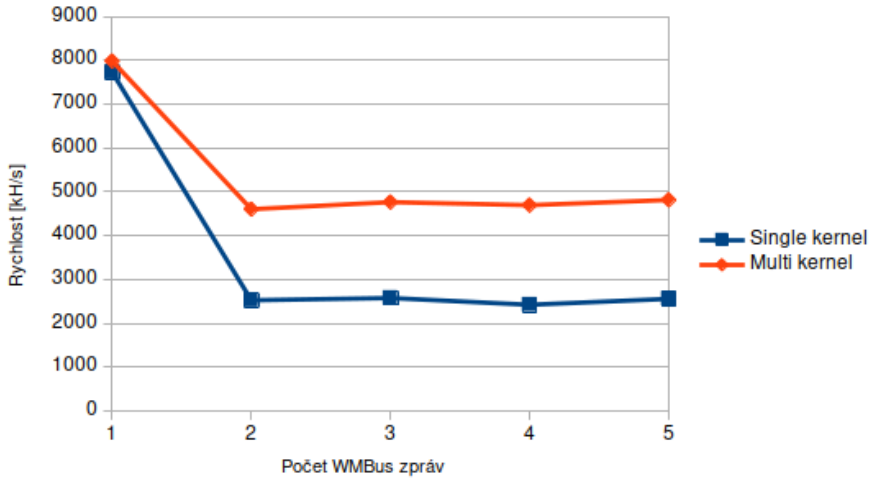


Obrázek 6.7: Rychlosti při různých počtech hash (zpráv s různými klíči) – O204.

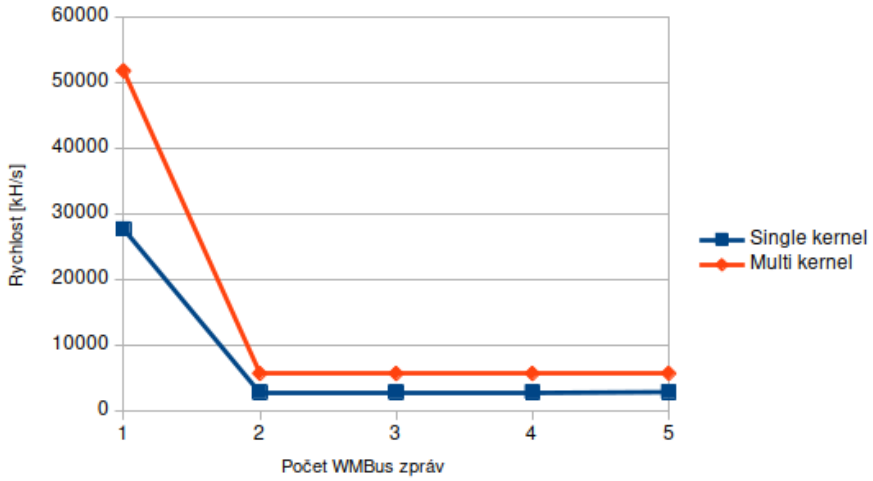
Porovnání rychlostí pro různý počet zpráv

Cílem tohoto testu bylo zjistit vliv počtu WM-Bus zpráv v jednom hash na rychlost. Test byl proveden pro single i multi kernel, v případě multi kernelu byly dodány dva hash. Při testování byl použit mód 05. Z výsledných grafů 6.8 a 6.9 lze vyčíst, že kromě případu použití jedné WM-Bus zprávy je rychlost konstantní.

Výrazně vyšší rychlost pro jednu WM-Bus zprávu je způsobena tím, že klíč, který je označen za správný, se vyskytuje statisticky jednou za 2^{16} pokusů. Z toho důvodu byl „správný“ klíč nalezen dříve, než mohlo dojít k ustálení a věrohodnému změření rychlosti programem Hashcat. Při zanedbání této chyby lze závislost považovat za lineární.



Obrázek 6.8: Rychlosti při různých počtech WM-Bus zpráv – ThinkPad.



Obrázek 6.9: Rychlosti při různých počtech WM-Bus zpráv – O204.

Výkonnost testovacího programu PoC

V tomto testu byla sledována rychlost programu PoC, který byl implementován v jazyce C před samotnou prací. Měření proběhlo pro různý počet vláken programu. Každé spuštění trvalo 30 sekund, po nichž byl program ukončen a vypsán poslední ověřovaný kandidátní klíč. Pro vstup kandidátních klíčů byl použit `wlist_gener` v hex–ASCII režimu. Naměřené hodnoty zobrazuje tabulka 6.5.

Počet vláken	ThinkPad	O204
1	4.192	8.799
3	4.299	9.365
8	4.292	9.429

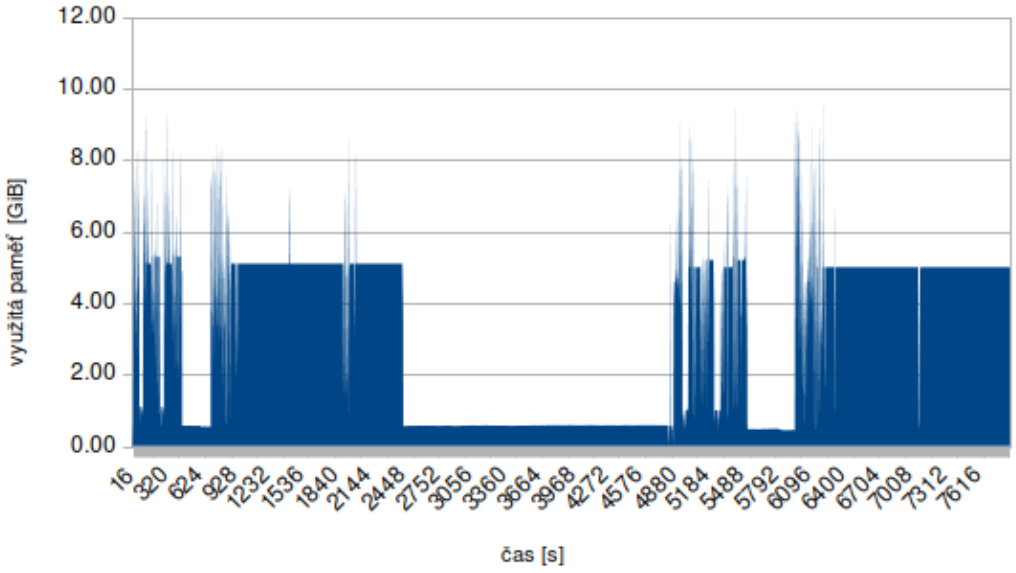
Tabulka 6.5: Rychlost programu PoC pro různý počet vláken. Uvedené rychlsti jsou v [MB/s].

Využití operační paměti v průběhu testování

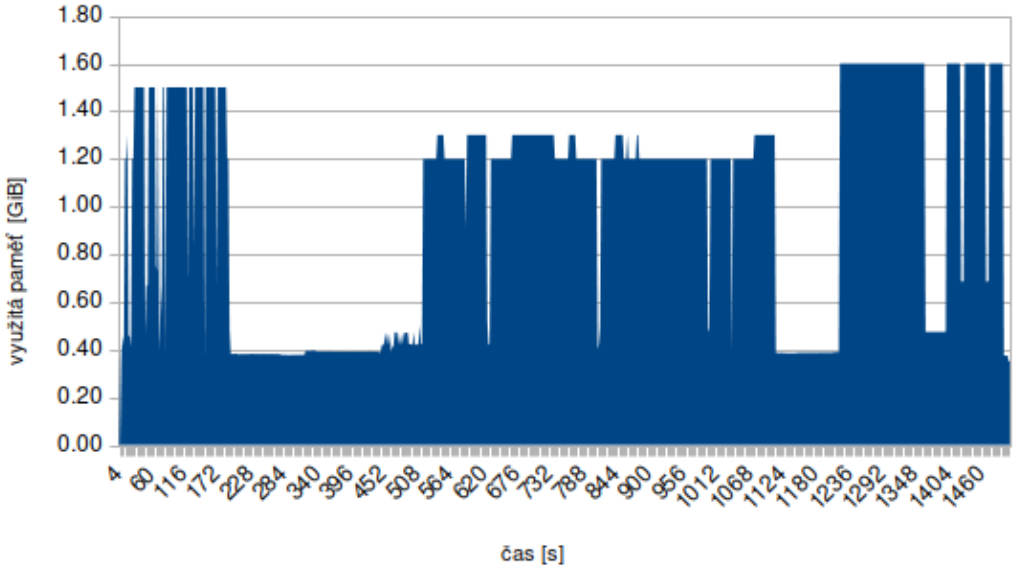
Poslední sledovanou veličinou je využitá operační paměť. Ta byla průběžně zaznamenávána v průběhu testování. Jedná se o paměť procesoru (v terminologii OpenCL a Hashcat se nazývá *host memory*), nikoli paměť dostupnou grafické jednotce.

Grafy 6.10 a 6.11 zobrazují průběh využití paměti. Hodnota 0.6 GiB pro ThinkPad, respektive 0.4 GiB pro PC v laboratoři O204, odpovídá využití samotným systémem. Krátké výkyvy (špičky) jsou způsobeny prodlevou mezi začátkem práce generátoru slovníku a časem, kdy Hashcat zinicializoval hardware, provedl ladění a self–test a teprve začal kandidátní klíče odebírat a zpracovávat.

Vyšší využití paměti na počítači ThinkPad oproti PC v O204 může být způsobeno tím, že integrovaná grafická jednotka v referenčním počítači nemá vlastní paměť.



Obrázek 6.10: Využití operační paměti v průběhu testování – ThinkPad.



Obrázek 6.11: Využití operační paměti v průběhu testování – O204.

Vyhodnocení

Výše popsanými testy bylo ověřeno, že vytvořené pluginy jsou funkční, nezávisle na software a hardware platformě. Dále byly vyhodnoceny a porovnány rychlosti jednotlivých částí a při

Při vůbec nejvyšší dosažené rychlosti v průběhu testování, 528.5 Mhash/s, by průchod celým prostorem klíčů pro 128 bitů šifry AES-128 trval přibližně 2×10^{20} let.

Kapitola 7

Závěr

Cílem této práce bylo rozšířit některý z již existujících nástrojů pro obnovu hesel o podporu obnovy klíčů šifrovaných zpráv protokolu Wireless Meter Bus. Tento cíl byl splněn.

V rámci studování teorie proběhlo seznámení se s protokolem Wireless Meter Bus z bezpečnostního hlediska v kapitole 2 a s nástroji pro obnovu hesel (kapitola 3). Zvláštní pozornost byla věnována programu Hashcat, který byl zvolen pro implementaci.

Na základě nabytých znalostí byl navržen algoritmus, kterým lze obnovit klíč šifrovaných zpráv protokolu Wireless M-Bus. Popisem návrhu se zabývá kapitola 4. Ten byl následně implementován pro existující nástroj – program Hashcat, což je popsáno v kapitole 5 – Implementace. Nakonec byly vyvinuté pluginy verifikovány a otestovány z hlediska výkonnosti v kapitole 6.

Praktickým výstupem práce jsou dva vyvinuté pluginy pro program Hashcat, které odpovídají dvěma bezpečnostním módům protokolu WM-Bus – 05 a 07. K těmto pluginům byly vytvořeny referenční implementace, jedna v jazyce C jakožto prototyp a dvě v jazyce Perl, které umožňují provádět validaci pluginů testovacím rámcem programu Hashcat. Dále byly pro účely testování vytvořeny dva generátory slovníků, jeden v jazyce Python a druhý v jazyce C.

Vzniklé pluginy jsou funkční pro testovací data a v nejbližší době budou nasazeny k dlouhodobému spuštění na fakultním počítači. Jejich cílem bude obnovit klíče zpráv vysílaných měřícími přístroji, které má fakulta k disposici k testování jejich bezpečnosti. V případě úspěchu tím může být dokázána nedostatečná bezpečnost zařízení, která jsou v současné době dostupná na trhu.

Dále plánuji vytvořené pluginy zveřejnit pomocí pull requestu do repozitáře programu Hashcat. Tím budou dány k disposici veřejnosti, jíž momentálně nástroj s tímto účelem není dostupný, přestože ověření bezpečnosti klíče je jedním z požadavků na bezpečnostní analýzu při nasazení protokolu Wireless M-Bus.

Existuje několik směrů, ve kterých lze práci dále posunout. Jsou jimi především profilování, na základě kterého by bylo možno kód dále optimalizovat a přidání podpory pro další bezpečnostní módy.

Díky této práci jsem především poznal Hashcat z hlediska vývojáře pluginu a získal tak přehled o jeho vnitřním fungování a částečně také o silně paralelním programování obecně. Také jsem rád za možnost prozkoumat komunikační protokol mimo obvyklý TCP/IP zásobník.

Literatura

[1] *Example_hashes [hashcat wiki]* [online]. [cit. 27. 04. 2022]. Dostupné z: https://hashcat.net/wiki/doku.php?id=example_hashes.

[2] *Fitcrack – Distributed password cracking system* [online]. [cit. 27. 12. 2021]. Dostupné z: https://fitcrack.fit.vutbr.cz/.

[3] *Hashcat - advanced password recovery* [online]. [cit. 27. 12. 2021]. Dostupné z: https://hashcat.net/hashcat/.

[4] *John the Ripper password cracker* [online]. [cit. 27. 12. 2021]. Dostupné z: https://www.openwall.com/john/.

[5] *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Standard ISO/IEC 7498–1. International Organization for Standardization, 1994. Dostupné z: https://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[6] *Komunikační systémy pro měřidla – Část 1: Výměna dat*. Norma ČSN EN-13757-1. Česká agentura pro standardizaci, srpen 2015.

[7] *Komunikační systémy pro měřidla – Část 7: Doprava a bezpečnostní služby*. Norma ČSN EN-13757-7. Česká agentura pro standardizaci, říjen 2018.

[8] *Komunikační systémy pro měřidla – Část 4: Bezdrátová komunikace M-Bus*. Norma ČSN EN-13757-4. Česká agentura pro standardizaci, listopad 2019.

[9] *OMS-Group* [online]. 2022 [cit. 22. 01. 2022]. Dostupné z: https://oms-group.org/en/.

[10] AB9IL.NET. *RTL2832 (RTL-SDR) Software Defined Radio* [online]. [cit. 20. 01. 2022]. Dostupné z: https://www.ab9il.net/software-defined-radio/rtl2832-sdr.html.

[11] BOGDANOV, A., KHOVRATOVICH, D. a RECHBERGER, C. Biclique Cryptanalysis of the Full AES. 2011. Dostupné z: https://eprint.iacr.org/2011/449.pdf.

[12] BRUNSCHWILER, C. Wireless M-Bus Security. *Whitepaper Black Hat USA*. 2013. Dostupné z: https://www.compass-security.com/fileadmin/Datein/Research/Praesentationen/blackhat_2013_wmbus_security_whitepaper.pdf.

[13] DWORKIN, M., BARKER, E., NECHVATAL, J., FOTI, J., BASSHAM, L. et al. *Advanced Encryption Standard (AES)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2001-11-26 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197. Dostupné z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[14] Hranický, R. *Digital Forensics: The Acceleration of Password Cracking*. Brno, CZ, 2022. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/phd-thesis/890/.

[15] Martin, E. *Home automation: reading my IZAR WMBus PRIOS hot water smart meter* [online]. 2020 [cit. 22. 01. 2022]. Dostupné z: https://www.zewaren.net/wmbus-izar-meter.html.

[16] Schneier, B. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley, 2015. ISBN 9781119096726. Dostupné z: https://books.google.co.uk/books?id=VjC9BgAAQBAJ.

[17] Steube, J., T, M., philsmd a Williams, R. *Hashcat Plugin Development Guide* [online]. [cit. 26. 03. 2022]. Dostupné z: https://github.com/hashcat/hashcat/blob/master/docs/hashcat-plugin-development-guide.md.

[18] Steube, J. a Valdés, L. *Maskprocessor* [online]. 2019 [cit. 19. 04. 2022]. Dostupné z: https://github.com/hashcat/maskprocessor/blob/4a1ddeac7de3242903148bc57732818b5ffca3e2/README.md.

[19] Wikipedia contributors. *Meter-Bus — Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 22. 01. 2022]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Meter-Bus&oldid=1058487557.

[20] Woolley, C. *Introduction to OpenCL*. 2011 [cit. 27. 03. 2022]. Dostupné z: https://season-lab.github.io/PFP/materiale/NVIDIA-intro_to_opencl.pdf.

[21] Česko. *Zákon, kterým se mění zákon č. 458/2000 Sb., o podmínkách podnikání a o výkonu státní správy v energetických odvětvích a o změně některých zákonů (energetický zákon), ve znění pozdějších předpisů, a další související zákony*. Sbírka zákonů České republiky, 14. září 2021. ISSN 1211-1244. Dostupné z: https://www.merenionline.cz/images/2021-LEGISLATIVA/sb0160-2021.pdf.

Příloha A

Zkrácený obsah souboru Readme.md

Následuje obsah souboru Readme.md z přiloženého paměťového média, zkrácený o výpis adresářové struktury.

Bakalářská práce - OBNOVA SDÍLENÝCH KLÍČŮ PROTOKOLU WIRELESS M-BUS

- autor: Ondřej Mikula
- VUT FIT, Brno
- ak. r. 2021/2022
- licence: MIT (viz license.txt)

V tomto adresáři se nacházejí všechny kódy, texty a data vytvořené, převzaté, či využité, v rámci bakalářské práce.

V adresářích se spustitelným kódem se nacházejí soubory Makefile. Ty umožňují kompilaci a uvádějí několik příkladů užití.

V adresáři hashcat-master se nachází naklonovaná aktuální (k 23. 04. 2022, commit f0037d93684eb72b8d4ae5c173ef7e43621efb20) verze programu Hahscat. Obsah tohoto adresáře není upraven a mělo by být možné jej v případě potřeby nahradit novější verzí tohoto programu.

Adresář hc_wmbus_plugin obsahuje kód pluginu pro Hashcat.

Nastavitelné parametry v Makefile:

- HC_FOLDER: složka se zdrojovým kódem Hashcat, kam se kód nakopíruje a spustí.
- WLIST_GENER: příkaz, který spouští generátor slovníku. Jsou k disposici možnosti implementace v C, Pythonu a Maskprocessor, případně zcela jiný. Neovlivňuje ukázky, které používají masky.
- DATA_FOLDER: složka se vzorky dat, se kterými jsou spouštěny ukázky běhu.

Důležité rules v Makefile:

- build: překopíruje soubory pluginu do adresáře Hashcatu a provede kompilaci.
- build_kernel: provede kompilaci kernelu. Vzhledem k tomu, že nelze explicitně vynutit kompilaci určitého kernelu, je smazán obsah složky se zkompilovanými kernely, čímž bude při příštím spuštění zkompilován.
- benchmark: spustí benchmark (výkonový test) integrovaný v Hashcatu nad pluginem.
- test: spustí testovací skript Hashcatu nad pluginem.
- self_test: spustí Hashcat bez parametru –self-test-disable, čímž provede self-test pluginu. V ostatních rules je self-test vypnut.
- zbylé rules spouští Hashcat s pokusnými daty. Z nich stojí za zmínku experiment_kaden a experiment_stul, které spouští Hashcat s reálnými zachycenými daty měřícího přístroje.

Wordlist generátor

V adresáři wlist_gener jsou dvě vzniklé implementace generátoru slovníku, wlist.c a wlist.py a Makefile pro překlad. Dále obsahuje složku maskprocessor s přejatým kódem generátoru Maskprocessor (z https://github.com/hashcat/maskprocessor, není upraven).

Adresář tools

obsahuje tři pomocné nástroje:

- reader je upravená verze Python skriptu od Ing. Libora Polčáka, Ph.D., který binární soubor se zachycenými daty převede na vstupní formát pro Hashcat.
- V ascii_encode jsou dvě utility, jedna překóduje data na hex-ASCII, druhá z nich obráceně.

Adresář testing_data

obsahuje soubory s pokusnými daty. kaden.dump a 2021-06-25-stůl.dump obsahují reálně zachycené zprávy, poskytnuté vedoucím práce. Zbylá jsou uměle vytvořená, nebo převzatá z příkladů.

Adresář tests

v adresáři tests je uložen skript, který provádí validaci a výkonové testování. V podadresáři results jsou uloženy výsledky spuštění, na základě kterých vznikla kapitola Vyhodnocení a testování v textu práce.

Adresář doc

obsahuje textovou část BP ve zdrojové podobě v jazyce TeX a přeloženou do souboru PDF.