VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SBĚR INDIKÁTORŮ KOMPROMITACE Z OPERAČNÍCH SYSTÉMŮ

COLLECTING INDICATORS OF COMPROMISE FROM OPERATING SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JIŘÍ PROCHÁZKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MATĚJ GRÉGR, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Procházka Jiří, Bc.**
Program: Informační technologie Obor: Počítačové sítě a komunikace
Název: **Sběr indikátorů kompromitace z operačních systémů**
Collecting Indicators of Compromise from Operating Systems
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s problematikou detekce indikátorů kompromitace z operačních systémů a nástroji, které se pro tento účel používají. Zaměřte se primárně na operační systém Linux.
2. Navrhněte nástroj, který otestuje operační systém, zda-li není kompromitovaný.
3. Implementujte navržený nástroj a navrhněte propojení se systémem provozovaným Národním úřadem pro kybernetickou a informační bezpečnost.
4. Otestujte nástroj v produkční síti a zhodnoťte dosažené výsledky.

Literatura:

- Newman, R. *Computer Security: Protecting Digital Resources*. Jones and Bartlett Publishers, Inc., 2009, USA.
- Cox, K; Gerg, Ch. *Managing security with Snort and IDS tools*. O'Reilly Series. ISBN 0-596-00661-6.
- Danyliw, R., *The Incident Object Description Exchange Format Version 2*, RFC 7970, DOI 10.17487/RFC7970, November 2016.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

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

Vedoucí práce: **Grégr Matěj, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 31. října 2018

Abstrakt

Tato práce se věnuje návrhu a implementaci aplikace pro sběr indikátorů kompromitace ze systémů. Součástí práce je seznámení se s pojmem indikátor kompromitace a popis běžně používaných kategorií. Práce obsahuje shrnutí existujících nástrojů věnujících se podobné problematice. V práci je zahrnut také výčet několika existujících formátů pro zápis a sdílení indikátorů kompromitace a výběr formátu, který výsledná aplikace používá. V práci jsou zhodnoceny i výsledky z testování, které bylo prováděno lokálně a na infrastruktuře kybernetického cvičení.

Abstract

Focus of this thesis is on the design and implementation of an application for gathering indicators of compromise from the systems. In the thesis, there is an introduction to the term indicator of compromise and description of commonly used categories. Next, there is a summary of existing tools with a similar focus. In the thesis, there is a list of some existing formats for sharing of indicators of compromise and selection of format which resulting application uses. After the implementation, application was tested both locally and on infrastructure of cyber exercise.

Klíčová slova

indikátory kompromitace, digitální forenzní analýza, it bezpečnost, kybernetický incident

Keywords

indicator of compromise, digital forensics, it security, cyber incident

Citace

PROCHÁZKA, Jiří. *Sběr indikátorů kompromitace z operačních systémů*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Matěj Grégr, Ph.D.

Sběr indikátorů kompromitace z operačních systémů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Matěje Grégra, PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Procházka
19. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu práce panu doktoru Grégrovi za odborné vedení, rady a náměty, které mi během konzultací poskytl.

Obsah

Kapitola 1

Úvod

Tématem této práce je návrh a implementace nástroje pro sběr indikátorů kompromitace ze systémů běžících na operačním systému Linux. Součástí teoretické části je seznámení se s pojmem indikátor kompromitace, vyjmenování a popis nejběžnějších kategorií indikátorů kompromitace a seznámení se s nástroji, které s nimi pracují. Obsahuje také popis formátů, které se pro popis indikátorů kompromitace používají a výběr formátu, který je použit ve výsledné aplikace. Poslední částí teoretického bloku je návrh aplikace. Technická část práce obsahuje popis implementace nástroje a testování. Testování bylo prováděno na lokální infrastruktuře a také na infrastruktuře jednoho kybernetického cvičení a byla zde testována nejen funkčnost aplikace, ale také její výkon a použitelnost v praxi.

Výstupem této práce je nástroj, který správcům umožní jednoduše a rychle provést skenování sítě na přítomnost zaslaných indikátorů kompromitace. Tato aplikace však nemá za cíl zcela nahradit vyšetřování, ale pomoci především správcům IT infrastruktury dotčených subjektů a zaměstnancům NÚKIB[1] získat v co nejkratším čase alespoň základní přehled o stavu sítě na daném subjektu. Nástroj bude zaměřený na stanice s operačním systémem Linux. Důvodem tohoto zaměření je především jejich majoritní podíl na serverových aplikací napříč soukromou i veřejnou sférou. Lze ji však rozšířit tak, aby fungovala i na jiných operačních systémech. Je napsaná v jazyce Python 3, což je multiplatformní jazyk.

Důvodem, proč byla tato práce napsána a nástroj implementován, je snaha o zrychlení reakce ze strany IT správců, kteří spravují systémy KII[2] a VIS[3], podle definice uvedené v § 2 zákona o kybernetické bezpečnosti (zákon č. 181/2014 Sb.). Pokud se na těchto systémech detekuje kybernetický bezpečnostní incident, správce je povinen nahlásit tuto skutečnost NÚKIB a ten poté může v rámci vyšetřování požádat i jiné subjekty o dodatečnou kontrolu na jejich infrastruktuře. Taková kontrola probíhá zasláním tzv. indikátorů kompromitace subjektům, jejichž správci potom na systémech ve spravovaných sítích provádějí kontrolu na výskyty těchto indikátorů. Ne všichni správci však mají znalosti, čas a lidské zdroje, což může negativně ovlivnit vyšetřování a také degradovat úroveň kybernetické bezpečnosti v České republice.

V první kapitole této práce se čtenář seznámí s pojmem indikátor kompromitace a s výběrem nejčastějších kategorií. U jednotlivých kategorií je také napsáno, kde se s nimi správce může na operačním systému setkat, přičemž pokrývá nejběžnější situace. Další kapitola obsahuje seznámení se s existujícími nástroji, které se v oblasti digitální forenzní analýzy používají pro práci s indikátory kompromitace. Výběr pokrývá programy různých zaměření

[1] Národní úřad pro kybernetickou a informační bezpečnost
[2] Kritická informační infrastruktura
[3] Významný informační systém

a různých rozsahů. V další kapitole se čtenář seznámí s existujícími formáty, které slouží pro sdílení indikátorů kompromitace. Součástí této kapitoly je také výběr a popis formátu, který bude aplikace používat. Poté následuje kapitola obsahující návrh samotné aplikace, na základě informací získaných během rešerše.

V technické části práce je kapitola popisující implementace nástroje. Ta se věnuje návrhu jednotlivých komponent aplikace, především návrhům tříd a jejich vzájemného propojení. Další kapitola je zaměřená na testování v testovacím prostředí a v prostředí infrastruktury kybernetického cvičení. Během testování byly použity různé množiny testovacích dat, i topologie o různých velikostech. Během testování se ověřovala funkčnost aplikace a také doba, za jakou vyhledávání indikátorů kompromitace provede. Měřen byl také vliv optimalizací na výkon aplikace. Součástí této kapitoly jsou také grafy z výsledků tohoto testování. Závěr se věnuje zhodnocení celé práce, jejím kladům a záporům a možnostem, jakými lze aplikace využít, a především rozšířit v budoucnosti aby našla širší uplatnění v produkčních sítí.

Kapitola 2

Indikátory kompromitace

Pod pojmem indikátor kompromitace (z anglického Indicator of Compromise, dále v textu uváděno také pod zkratkou IoC) si lze představit data, jejichž přítomnost na systému značí vysokou pravděpodobnost jeho kompromitace. Různé zdroje se k samotné definici staví rozdílně. RSA[1] pojem IoC definuje jako vniknutí do systému, které lze identifikovat pomocí malware signatur, IP adres nebo domén spojených s některou z command and control kampaní[2][15]. Mezi další definice, se kterými se lze setkat, patří například ta, která IoC definuje jako části forenzních dat, které lze najít v záznamech logů nebo v souborech, které identifikují potenciálně škodlivou aktivitu na systému nebo v síti. Slouží jako „cestička z drobků chleba“ pro administrátory a analytiky, kteří se snaží na základě těchto nálezů detekovat škodlivou aktivitu v co nejranější fázi útoku [14]. Obecně objevení jakéhokoliv IoC (za předpokladu, že zdroje IoC ze kterých čerpáme, jsou důvěryhodné) už samo o sobě značí, že systém mohl být kompromitován. Detekce těchto indikátorů však může být komplikovaná z několika důvodů. Prvním je relativně malý počet otevřených zdrojů - organizace buď z detekovaných incidentů nedělají seznamy IoC nebo je nezveřejňují a používají je pouze pro svou interní potřebu. Jako další důvod lze zmínit složitý proces vyhledávání IoC.

Cílem této práce je navrhnout a implementovat nástroj, který bude schopný na systému detekovat přítomnost indikátorů kompromitace. Součástí je také návrh vlastního formátu, pomocí kterého budou jednotlivé indikátory kompromitace popsané. Z toho důvodu je vhodné se seznámit s běžnými typy IoC a se způsoby, jak je lze na systému detekovat. Na základě těchto informací a na základě požadavků na výslednou aplikaci lze pak vytvořit formát, ve kterém budou indikátory uloženy a sdíleny mezi subjekty. Po návrhu formátu je proveden a popsán návrh aplikace, která na základě poskytnutého seznamu indikátorů kompromitace prohledá cílový systém a ověří jejich přítomnost.

Pro tuto práci jsem vybral tyto kategorie IoC - IP adresu, URL, doménové jméno, hash souboru a cestu souboru. Tyto kategorie jsou indikátorů kompromitace jsou popsány v následujících podkapitolách. Tyto kategorie jsem zvolil proto, že jsou ty nejběžnější, se kterými se lze v reálném prostředí setkat. Součástí původního návrhu byla také kategorie pro YARA pravidla. Ta jsem zvolil, protože umožňují komplexnější vyhledávání pomocí vzorů nad daným vzorkem dat, kterým může být téměř cokoliv - log soubory, záznam síťového toku dat, obraz paměti, binární soubory, apod. Od implementace zpracování a vyhledávání IoC této kategorie jsem ale nakonec upustil z důvodů popsaných v podkapitole 2.5

[1] Americká firma zabývající se síťovou a počítačovou bezpečností, viz https://www.rsa.com

[2] Označuje strategii útočníků v oblasti kyberbezpečnosti, kdy jeden centrální server patřící útočníkovi ovládá a řídí napadené stanice.

2.1 IP adresa

IP adresa slouží k jednoznačné identifikaci síťového rozhraní v počítačové síti. Aktuálně rozlišujeme dvě implementace protokolu IP a to IPv4 a IPv6. Z důvodu malého počtu adres pro IPv4 je tento protokol postupně nahrazován protokolem IPv6. Pro škodlivý kód se používá nejčastěji jako adresa útočníkova serveru (občas nazývaný jako Command & Control server), který je vystavený do internetu a ze kterého lze buď stahovat škodlivá data nebo naopak na něj lze data z kompromitovaného stroje odesílat.

IPv4 adresa se zapisuje ve formátu 4 dekadických čísel v rozsahu 0 - 255, oddělených tečkami, např. 192.168.254.1. Velikost jedné IP adresy je 32 bitů. IPv6 adresa se zapisuje v hexadecimálním formátu a její velikost je 128 bitů. Při zápisu se jednotlivá slova oddělují dvojtečkami, např. 2a00:1450:4014:80d::200e. Protokol IP je nejrozšířenější protokol v počítačových sítí a také je to jeden z nejběžněji používaných indikátorů kompromitace obecně.

```
[vagrant@localhost ~]$ ss -tlnpa
State       Recv-Q Send-Q          Local Address:Port                        Peer Address:Port
LISTEN      0      128                         *:111                              *:*
LISTEN      0      128                         *:22                               *:*
LISTEN      0      100                 127.0.0.1:25                               *:*
ESTAB       0      0                   10.0.2.15:22                       10.0.2.2:61937
LISTEN      0      128                        :::111                             :::*
LISTEN      0      128                        :::22                              :::*
LISTEN      0      100                       ::1:25                              :::*
```

Obrázek 2.1: Ukázka výstupu nástroje ss.

```
root@ubuntu-xenial:/home/vagrant# netstat -tulnpa
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      26840/mysqld
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1363/sshd
tcp        0      0 10.0.2.15:35698         91.189.91.26:80         TIME_WAIT   -
tcp        0     36 10.0.2.15:22            10.0.2.2:61975          ESTABLISHED 22134/sshd: vagrant
tcp        0      0 10.0.2.15:48374         91.189.88.161:80        TIME_WAIT   -
tcp6       0      0 :::80                   :::*                    LISTEN      24052/apache2
tcp6       0      0 :::21                   :::*                    LISTEN      23067/vsftpd
tcp6       0      0 :::22                   :::*                    LISTEN      1363/sshd
udp        0      0 0.0.0.0:68              0.0.0.0:*                           955/dhclient
```

Obrázek 2.2: Ukázka výstupu nástroje netstat.

Možnosti, kde se IP adresa může na systému objevit jsou četné. Může se objevit v lozích systému nebo aplikací, v konfiguračních souborech, v seznamu navázaných spojení nebo třeba v pravidlech firewallu. Prakticky největší smysl hledat výskyty IP adresy dává v navázaných síťových spojeních. Ta se na systému Linux dají zobrazit například pomocí nástrojů `netstat` nebo `ss`. Tyto nástroje umí zobrazit navázaná síťová spojení i proces, který dané síťové spojení navázal. Oba tyto nástroje jsou běžně dostupné pro všechny Linuxové distribuce a často bývají na systémech přítomné od počáteční instalace systému. Ukázky výstupů obou nástrojů je možné vidět na obrázcích 2.1 a 2.2. Ani jeden z těchto nástrojů však nedrží historii navázaných síťových spojení a tudíž pokud je dané síťové spojení ukončeno, nedá se pomocí těchto nástrojů dohledat. Abychom se pokusili co nejvíce zmírnit riziko, že nám hledané síťové spojení spojené s útočníkovou IP adresou unikne, je vhodné také prohledat log záznamy systému a aplikací, které na něm běží. Tyto záznamy se

nejčastěji vyskytují ve složce `/var/log` a názvy, respektive cesty jednotlivých systémových a aplikačních logů se mohou lišit na základě použitého operačního systému. Log záznamy se mohou kromě textového formátu ukládat také v binární podobě. Jako příklad lze uvést systémy používající démona pro správu systému Linux `systemd`, kde se logovací démon jmenuje `journald` a jeho záznamy jsou na systému ukládány v binární podobě. V původním nastavení však tyto záznamy nejsou persistentní a ukládají se do složky `/run/log/journal` a po restartu systému se mažou. Avšak i v těchto binárních formátech je možné jednotlivé řetězce vyhledávat, ačkoliv pak nemusí být na první pohled zřejmé, čeho se daný záznam týká. Pak je v kompetenci správce, aby si daný záznam zobrazil v nástroji tomu určenému a nález ověřil. Konfigurační soubory, kde se IP adresa může také nacházet, bývají nejčastěji uloženy ve složce `/etc` nebo v některé z jejích podsložek. Zde zpravidla všechny soubory bývají textovém formátu, proto v nich lze snadno vyhledávat nástroji jako je například `grep`. V neposlední řadě je také vhodné zkontrolovat možné spustitelné soubory, zda-li není přímo v nich napevno IP adresa zapsaná. Případně lze prohledat i přímo paměťový prostor běžících procesů, ke kterému se lze dostat pomocí speciálního adresáře `/proc`, ve kterém názvy podsložek odpovídají PID[3] běžících procesů.

2.2 Hash souboru

Hashovací funkcí (někdy též nazývanou jako kontrolní součet) nazýváme jednosměrnou funkci, která převádí vstupní data na číslo. Nejčastěji se s ní můžeme setkat v informatice při úkonech jako je zajištění integrity dat, ověřování kryptografických klíčů, certifikátů či elektronických podpisů nebo pro vyhledávání v tabulkách. Hashovací funkce má tyto hlavní vlastnosti [16, 17]:

1. Jakkoliv dlouhý vstup musí generovat vždy stejně dlouhý výstup.
2. I malou změnou vstupních dat způsobíme velkou změnu výstupních dat.
3. Z výstupu je nemožné složit původní vstup (neexistuje korelace mezi vstupem a výstupem).
4. Je vysoce nepravděpodobné, že funkce pro dva různé vstupy vygeneruje dva stejné výstupy.

Mezi nejběžnější hashovací funkce patří algoritmy z rodiny Message-Digest algorithm, kam patří i algoritmus MD5. Tato funkce vytváří výstup o délce 128 bitů. Ačkoliv se dnes z bezpečnostních důvodů použití této funkce nedoporučuje, stále se jedná o jeden z nejrozšířenějších algoritmů pro hashování. Druhou známou rodinou hashovacích funkcí jsou algoritmy s názvem Secure Hash Algorithm, známé pod zkratkou SHA. Součástí této rodiny jsou algoritmy SHA-1, SHA-224, SHA-256, SHA-384 a SHA-512. Délka výstupu prvního jmenovaného je 160 bitů, u zbylých délku výstupu v bitech značí číslo v názvu algoritmu.

Na operačních systémech z rodiny GNU/Linux existuje několik možností, kde se s touto kategorií IoC můžeme setkat. Nejčastější je hash souboru, který se používá pro ověření integrity. Můžeme tak počítat a porovnávat hashe souborů, které se nachází na zkoumaném systému. Tento postup se často používá pro ověření, že soubory na systému nebyly pozměněny nebo nahrazeny. Cesty na systému lze rozdělit na několik kategorií a to podle pravděpodobnosti, se kterou se na dané cestě škodlivý soubor může objevit. Ostatní možné lokace

[3] Process Identification Number

výskytu hashe na systému nemá cenu zkoumat, vzhledem k poměru složitosti a zisku. Pro optimalizaci tohoto hledání lze také přeskakovat větší soubory. V praxi jsou totiž hledané soubor nejčastěji malware, který je až v 97% případů menší než 1 MB [10]. Důvodem malé velikosti je především snaha útočníků infikovat stroj co nejdříve, což by jim při stahování velkého souboru na systém komplikovalo situaci.

Pro výpočet hashe souboru lze využít nástroje běžně dostupné na operačních systémech Linux jako je například `md5sum` nebo lze některé z běžně dostupných skriptovacích jazyků jako je Python nebo Perl.

2.3 Cesta souboru

Dalším běžným indikátorem kompromitace je cesta souboru. Ta obvykle značí umístění škodlivého souboru, kterým může být například skript nebo spustitelný binární soubor. Kontrola existence souboru na zadané cestě je jednoduchá a výpočetně i časově nenáročná, proto je vhodné ji zahrnout do kategorií, které budeme vyhledávat.

Zápis cesty souboru na operačních systémech GNU/Linux lze rozdělit na dvě varianty:

1. Pokud cesta začíná znakem /, pak se jedná o absolutní cestu a vyhledává se od kořenové složky systému.

2. Pokud cesta nezačíná znakem /, pak se jedná o relativní cestu a vyhledává se od aktuální složky.

Kontrola existence souboru podle cesty je jednoduchá, postačí například volání programu `ls` a kontrola návratového kódu. Nebo lze použít funkci `test` jazyka Bash s přepínačem `-f` pro testování existence soubor ve formátu. Tyto metody ale neumožňují ověřit existenci souboru, který byl smazán. V takovém případě však lze tuto informaci dohledat pouze za pomoci některého z komplexnějších nástrojů pro forenzní analýzu zaměřených na extrakci dat a metadat ze souborových systémů. Tento proces je náročný jak časově, tak výpočetně a je vhodné jej provádět na obrazu disku, nikoliv přímo na fyzickém disku a to z důvodu zachování originálního disku jako například důkazního materiálu. Jelikož nástroj, který je předmětem této práce má sloužit pouze pro rychlé skenování hledaných indikátorů kompromitaci, není potřeba se zabývat touto analýzou do detailu a postačí test existence existujících souborů. Důslednější forenzní analýza disku by proběhla až v případě pozitivního nálezu některého z IoC.

2.4 URL a doménové jméno

URL[4] slouží ke specifikaci umístění zdrojů dat. URL nebo doménové jméno může stejně jako IP adresa sloužit škodlivému kódu pro připojování na domovský server za účelem stažení nebo nahrání dat. Výhodou použití doménového jména či URL, oproti IP adrese, je například to, že v případě že útočník přijde o konkrétní IP adresu, může použít jinou a pouze pozměnit DNS záznam dané domény. Protože lze zápis doménového jména chápat jako podmnožinu URL, tak v této práci sloučím doménové jméno a URL v jednu kategorii.

Formát zápisu doménového lze popsat jako posloupnost řetězců oddělených tečkami. Platí, že poslední část označujeme jako doménu 1. řádku (anglicky top-level domain), předposlední jako doménu 2. úrovně, apod. V doménových jménech se můžou objevit znaky

[4] Uniform Resource Locator

anglické abecedy, číslice nebo pomlčky. Zápis URL je obsáhlejší, kromě doménového jména navíc může obsahovat informaci o protokolu a rovněž také údaj o lokaci námi požadovaného zdroje. Místo doménového jména se může použít i IP adresa. Syntaxe formátu URL vychází z obecné syntaxe formátu URI[5].

Výskyty URL a doménového jména na systému lze najít v lozích systému a aplikací nebo v konfiguračních souborech aplikací. Svým způsobem se totiž jedná o kategorii podobnou jako je IP adresa, jelikož samotné doménové jméno se na IP adresu překládá pomocí protokolu DNS[6]. URL ani doménové jméno se ale nemůže vyskytovat například v seznamu navázaných síťových spojení. Zde tedy dává smysl vyhledávat výskyty pouze v souborech na disku nebo v paměti systému.

2.5 YARA pravidla

YARA [12] je nástroj sloužící k identifikaci a klasifikaci vzorků škodlivého kódu. YARA pravidla jsou zápisy, pomocí kterých se popisuje škodlivý kód. Skládá se z řetězců, podmínek a metadat, které slouží k popisu a klasifikaci. Ukázku takového pravidla můžeme vidět na výpisu 2.1.

```
1 rule AsciiExample {
2     strings:
3         $ascii_string = "hello"
4
5     condition:
6         $ascii_string
7 }
```

Výpis 2.1: Ukázka jednoduchého YARA pravidla

Pravidlo v ukázce 2.1 nad daným vzorkem dat vyhledává řetězec "hello". Z ukázky je patrno, že pravidlo se dělí na dvě hlavní části - řetězce (strings) a podmínky (conditions). Řetězců může být v pravidle několik a nemusí jít pouze o ASCII reprezentaci, ale jednotlivé znaky mohou být zadány i například hexadecimálně. YARA pravidla umožňují i použití tzv. wildcards, offsetů, rozsahů a mnoho dalšího. Všechny tyto konstrukce jsou přehledně popsány v dokumentaci [13]. V této práci jsem YARA pravidla chtěl využít pro skenování paměti nebo pro prohledávání obsahu souborů na systému. Ovšem psát tato pravidla je poměrně složité. V praxi se nejčastěji používají pro popis známých malware signatur a ty se vyznačují vysokou komplexitou. Klasické indikátory kompromitace však bývají jednoduché a na jejich popis stačí jednoduché konstrukce bez podmínek a popisu metadat. Rovněž problematické může být samotné ověřování na systému, protože vyžaduje instalaci speciálního software, který nebývá součástí běžných distribucí ani repozitářů. Klasický zápis YARA pravidel je víceřádkový řetězec, což by zkomplikovalo i výběr formátu pro zápis a sdílení indikátorů kompromitace. Ze všech kategorií zmíněných v této práci je navíc jediná, která se ve světě indikátorů kompromitace nepoužívá. Proto jsem se po pečlivém zvážení rozhodnul tuto kategorii neimplementovat.

[5] Uniform Resource Identifier
[6] Domain Name System

Kapitola 3

Srovnání existujících nástrojů

Obsahem této kapitoly je srovnání existujících nástrojů, které se věnují stejné, či podobné problematice. Nástroj, který by funkcionalitou odpovídal přesně tomu, jenž je cílem této práce jsem během výzkumu nenašel. Vybral jsem proto programy, z jejichž vlastností a funkcionalit bude možno čerpat při návrhu vlastní aplikace. Výběr byl konstruován tak, aby pokryl co nejširší škálu aplikací. Obsahuje proprietární i open source aplikace; vyvíjené velkými firmami i malými týmy respektive jednotlivci a aplikace různých architektur. Cílem srovnání je udělat si přehled o aktuálních trendech v oblasti forenzní analýzy počítačových systémů. Během srovnání jsem se snažil vyznačit nejdůležitější vlastnosti známých existujících aplikací, které pak bude eventuálně možno použít při návrhu vlastní aplikace. Na závěr srovnání je pak k dispozici tabulka pro rychlý přehled 3.1.

3.1 GRR

GRR[1] je nástroj vyvíjený americkou firmou Google. Jedná se o tzv. incident response framework a zaměřuje se na vzdálený sběr dat z běžících systémů. Nástroj je napsaný v jazyce Python a využívá architekturu typu server-klient. Klient běží na koncových stanicích, které máme v úmyslu zkoumat, v režimu agenta. Server agenty spravuje, komunikuje s nimi a sbírá a zpracovává z nich data. Serverová část se skládá z několika různých komponent - frontend, workers a UI server. Poskytuje grafické webové rozhraní pro uživatele a také otevřené API[2], pomocí kterého lze plánovat úlohy na klientech a zpracovávat sbíraná data [2].

Jedná se o multiplatformní nástroj, funguje na operačních systémech Windows, macOS a Linux. Serverovou část lze nainstalovat z balíčkovacího nástroje pro jazyk Python `pip`, z balíčku pro systémy Debian/Ubuntu, pomocí Docker obrazu vydaného oficiálně vývojáři nebo přímo kompilací zdrojového kódu. Všechny tyto postupy jsou podrobně zdokumentovány v repozitáři dokumentace nástroje[3]. Nástroj nabízí také zabezpečenou komunikaci mezi serverem a agentem.

Mezi nejdůležitější vlastnosti nástroje GRR patří: možnost analýzy paměti běžícího systému pomocí YARA pravidel, možnost vyhledávání a stahování souborů, vyhledávání v registrech Windows, přímý přístup k souborovému systému na úrovni operačního systému pomocí nástroje SleuthKit (zkráceně také TSK) a monitorování zdrojů klienta (využití

[1] Google Rapid Reponse
[2] Application Programming Interface
[3] https://github.com/google/grr-doc

procesoru a paměti, využití IO). Nástroj je aktivně vyvíjen, na jeho vývoji se podílí tým o třiceti členech. Nástroj je open source[4] a je vydán pod licencí Apache Licence 2.0.

3.2 IOC Finder

Aplikaci IOC Finder naprogramoval a udržuje Floyd Hightower. Jedná se o open source aplikaci, napsanou v jazyce Python, která slouží k extrakci různých druhů IoC z textu. Nástroj se inspiruje modulem pro jazyk Python `extract_iocs` od Moses Schwartz, který v současnosti není dále vyvíjen[5]. Kód je zveřejněn pod licencí MIT Licence. Jedná se o menší projekt, samotná aplikace má okolo 300 řádků kódu. Aplikace umí v textu nalézt následující druhy IoC: IP adresy (jak pro protokol IPv4, tak IPv6), e-mailové adresy, doménová jména, URL, hashe (MD5, SHA-1, SHA-256, SHA-512), cesty registrů operačního systému Windows, ASN[6], CVE[7], bitcoinové adresy (P2PKH, P2SH, Bech32) nebo Google Adsense Publisher a Analytics Tracker ID.

Hlavní využití tohoto nástroje spočívá ve funkci `find_iocs`, která má jeden povinný vstupní argument, který obsahuje řetězec, jenž chceme na výskyt IoC testovat. Navíc má tři volitelné argumenty, pomocí kterých může uživatel extrahovat domény z URL, e-mailové adresy nebo IP adresu z CIDR[8] rozsahu. Funkce vrací slovník, kde klíčem jsou jednotlivé kategorie a hodnoty jsou listy obsahující všechny výskyty pro danou kategorii. Aplikace je dostupná ke stažení v GitHub repozitáři[9]. Součástí repozitáře je i dokumentace nástroje.

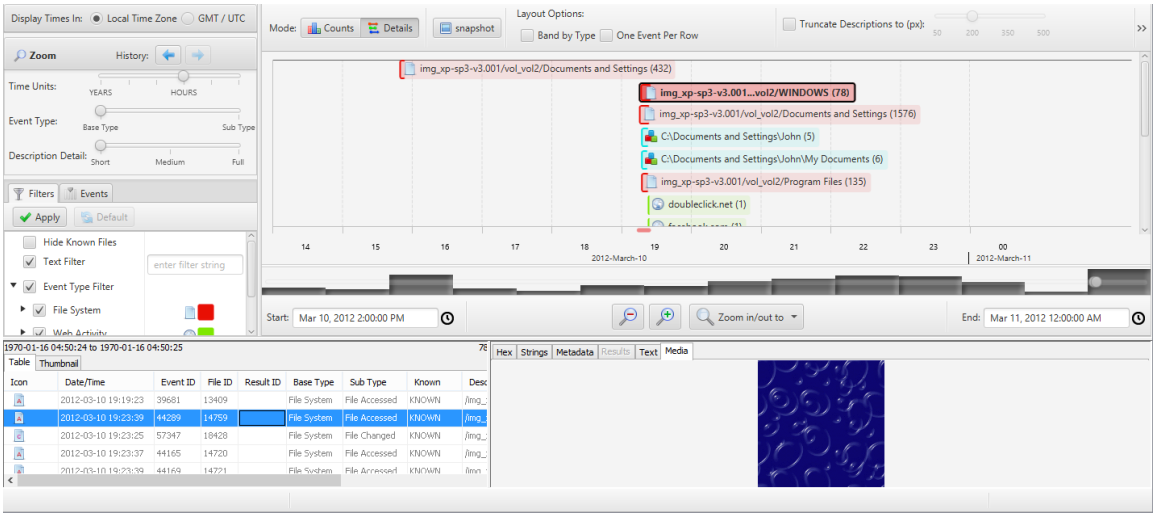
3.3 SleuthKit a Autopsy

Nástroje SleuthKit a Autopsy si kladou za cíl být vedoucím nástrojem v oblasti open source forenzní analýzy souborových systémů. Oba nástroje jsou multiplatformní a lze je provozovat na operačních systémech Windows, Linux a macOS. Jedná se o komunitní projekty, ačkoliv původní část nástroje SleuthKit vychází z nástroje zvaného The @stake Sleuth Kit (TASK). TASK rozšiřoval existující nástroje The Coroner's Toolkit (TCT) a TCTUTILs a přidal podporu pro souborové systémy typu FAT a NTFS. Autopsy byl původně vyvíjen jako grafické rozhraní nástrojů TCT a TCTUTILs, později však byl kompletně přepsán a nyní je určen pro nástroj SleuthKit [9].

SleuthKit lze využívat dvěma způsoby: jako knihovnu, napsanou v jazyce C nebo přímo nástroji určenými pro příkazový řádek. Jako knihovnu jej lze využít v jiných nástrojích pomocí implementovaných metod rozhraní API. Jeho použití je podrobně rozepsáno v příručkách uživatele na oficiálních stránkách aplikace[10]. Nástroje, určené pro příkazový řádek, umožňují uživateli provádět operace nad obrazy disku a jsou rozděleny do vrstev na nástroje pro správu souborového systému, nástroje pro správu svazků disku, nástroje pro správu obrazů disku a nástroje pro správu disku. Pro každou kategorii existují různé nástroje, např. v kategorii nástrojů pro správu souborového systému je nástroj `tsk_recover`, který slouží k extrakci alokovaných i nealokovaných souborů z obrazu disku; v kategorii obrazů disku, nástroj `img_stat` zobrazí detaily formátu obrazu. Nástroje jsou všechny popsány v doku-

[4] https://github.com/google/grr
[5] https://github.com/mosesschwartz/extract_iocs
[6] Autonomous System Number
[7] Common Vulnerabilities and Exposures
[8] Classless Inter-Domain Routing
[9] https://github.com/fhightower/ioc-finder
[10] http://wiki.sleuthkit.org/index.php?title=Library_User%27s_Guide

mentaci, která je dostupná online[11]. Většina nástroje je psaná v jazyce C, část je v jazyce C++ a část v jazyce Java. Kód je zveřejněný[12] pod licencemi IBM Open Source a Common Public Licence.



Obrázek 3.1: Ukázka grafického prostředí aplikace Autopsy.

Autopsy je platforma pro digitální forenzní analýzu a grafické prostředí nástroje SleuthKit (ukázku lze vidět na obrázku 3.1) a jiných nástrojů pro digitální forenzní analýzu. Nástroj se používá k analýze obrazů disku, lokálních disků nebo složek či lokálních souborů. Je psaná v jazyce Java a kód je dostupný v GitHub repozitáři[13] pod licencí Apache Licence 2.0. Nástroj umožňuje tvorbu časové osy událostí pro potřeby forenzní analýzy, filtrování výsledků podle hashe souboru, vyhledávání klíčových slov, extrahování užitečných dat (historie, záložky, cookies) z nejpoužívanějších prohlížečů (Chrome, Firefox, Internet Explorer), obnovu smazaných souborů a to včetně multimediálních a v neposlední řadě také vyhledávání IoC zadaných ve formátu STIX [11]. Nástroj umí pracovat s více jádry procesoru, běží na pozadí a informuje uživatele o výsledcích hned jak jsou dostupné. Stejně jako nástroj SleuthKit je i tento nástroj zdarma, přičemž je k němu možno přikoupit komerční podporu od vývojářů.

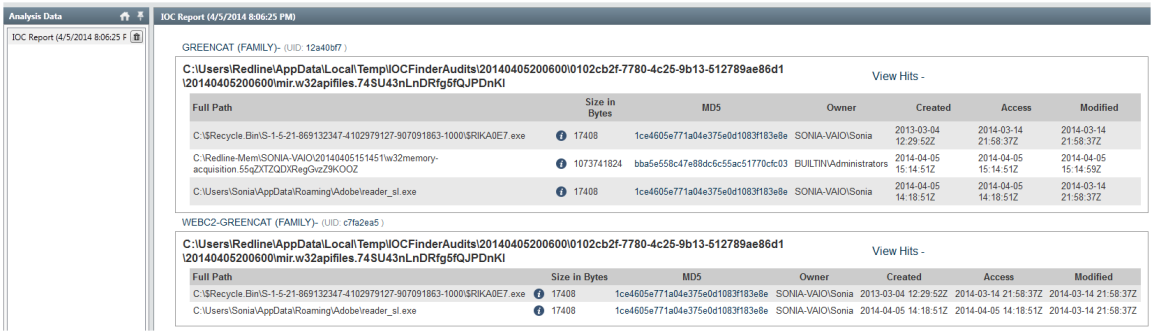
3.4 Redline

Redline je freeware nástroj od americké společnosti FireEye. Firma FireEye poskytuje hardware, software a služby sloužící k vyšetřování útoků v kybernetickém prostoru a také konzultace v oblasti kyberbezpečnosti. Nástroj Redline je proprietární, ale dostupný zdarma na oficiálních stránkách[14]. Nástroj podporuje jedinou platformu a tou je rodina operačních systému Windows, od verze XP až po nejnovější verzi Windows 10. Aplikace Redline umožňuje provádění auditu a sběr následujících druhů dat ze systému: informace o běžících procesech a nahraných řadičích v paměti, metadata souborového systému, data z registrů Windows, logy událostí, informace o síťové konfiguraci, informace o běžících a pozastave-

[11] http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview
[12] https://github.com/sleuthkit/sleuthkit/
[13] https://github.com/sleuthkit/autopsy
[14] https://www.fireeye.com/services/freeware/redline.html

ných službách, úlohách a historii webového prohlížeče. Kromě sběru těchto dat je dokáže i analyzovat a zobrazit, včetně možnosti filtrování podle požadovaných parametrů. Tyto informace pak lze zobrazit na časové ose, pomocí které si uživatel může udělat větší přehled o posloupnosti akcí během zkoumaného incidentu.

Nástroj samotný přímo umí provádět i analýzu indikátorů kompromitace. Aplikaci stačí poskytnout kolekci IoC, které chceme na systému hledat a nástroj sám ověří možné výskyty a v případě, že je některý z indikátorů kompromitace nalezen, je tato informace součástí závěrečné zprávy z analýzy (viz obrázek 3.2). Redline zpracovává IoC zadané v textovém serializačním formátu XML[15]. Pro správu indikátorů kompromitace firma Redline nabízí svůj další freeware nástroj IOC Editor, který je také dostupný pouze pro systémy Windows a lze jej stáhnout na oficiálních stránkách[16]. Schéma, které oba tyto nástroje používají, se nazývá OpenIOC 1.1 [5] a jeho popis je dostupný v repozitáři[17].



Obrázek 3.2: Ukázka závěrečné zprávy z analýzy indikátorů kompromitace v nástroji Redline.

3.5 Srovnání a závěr

Ze srovnání jsem vytvořil tabulku 3.1 pro rychlý přehled základních charakteristik a rozdílů. Z výše uvedených nástrojů je patrné, že každý nástroj se zaměřuje na trochu jinou oblast digitální forenzní analýzy a žádný z nich přímo neodpovídá nástroji, jehož návrh a vývoj je náplní této práce. Nejblíže takovému nástroji má aplikace Redline, ta je ale spustitelná pouze na operačních systémech Windows. Tento nástroj umí vyhledávat indikátory kompromitace na systému, poskytuje uživateli grafické rozhraní a po ukončení analýzy vytvoří přehlednou zprávu s výsledky. Během rešerše jsem zjistil, že se aplikace často věnují především diskům, případně obrazům disků. V oblasti digitální forenzní analýzy je však zapotřebí věnovat notnou dávku pozornosti také operační paměti počítače, jelikož se v ní mohou nacházet škodlivé procesy, případně jiná data vzniklá útočníkovým počínáním. Během výzkumu jsem také zjistil, že dvě ze zkoumaných aplikací používají již zavedené formáty indikátorů kompromitace. Autopsy používá formát STIX [11] a Redline používá formát vyvinutý stejnou firmou OpenIOC [5].

[15] Extensible Markup Language
[16] https://www.fireeye.com/services/freeware/ioc-editor.html
[17] https://github.com/mandiant/OpenIOC_1.1

Tabulka 3.1: Tabulka srovnání jednotlivých nástrojů

Název	Platformy	Primární účel	Otevřenost kódu	Jazyk
GRR	Windows, Linux, macOS	analýza disků a paměti	open source	Python
IOC Finder	Windows, Linux, macOS	vyhledání IoC v textu	open source	Python
SleuthKit	Windows, Linux, macOS	analýza diskových systémů	open source	C/C++
Autopsy	Windows, Linux, macOS	grafické rozhraní pro nástroje dig. forenzní analýzy	open source	Java
Redline	Windows	analýza disků a paměti	freeware	N/A

Kapitola 4

Výběr formátu pro sdílení indikátorů kompromitace

Během srovnávání různých nástrojů jsem narazil na dva existující formáty IoC, které nástroje využívaly a to OpenIOC [5] a STIX [11]. Při dalším výzkumu jsem objevil další formáty: MAEC [4], CAPEC [1], TAXII [3], Oval [6] a SCAP [8]. Do této kapitoly jsem zařadil srovnání a krátký popis vybraných existujících formátů, dále potom seznam požadavků na formát IoC, který bude výsledná aplikace používat, a rozhodnutí, ke které z variant jsem se v rámci práce uchýlil. Popis existujících formátů se nachází v podkapitole 4.1. Volba a důvody pro zvolení finálního formátu jsou pak uvedeny v podkapitolách 4.3 a 4.2. Cílem výběru je zvolení optimálního formátu, ve kterém budou indikátory kompromitace uloženy a který bude aplikace zpracovávat.

4.1 Existující formáty

Formáty MAEC, CAPEC, STIX, TAXII a OVAL jsou všechny pod správou americké neziskové společnosti MITRE Corporation[1]. Tato společnost se zabývá především výzkumem a mimo jiné také spravuje například systémy CVE[2] a CWE[3]. Formát SCAP provozuje americký NIST[4], který se zabývá standardy a vědou. Vydává také standardy pro šifrování a digitální podpisy.

Z formátů společnosti MITRE stojí za zmínku především STIX, jelikož ten slouží přesně pro sdílení indikátorů kompromitace. Ostatní formáty mají primárně jiné využití, například MAEC slouží k popisu malwaru, CAPEC slouží k popisu mechanismů útoku, TAXII pracuje na aplikační vrstvě a slouží jako doplněk formátu STIX určený pro komunikaci dat a OVAL slouží k popisu systému, který chceme v rámci incidentu analyzovat. Formát STIX je aktuálně ve verzi 2.0 a nabízí pestrou škálu informací, které do něj lze zakódovat. Jako příklad lze uvést možnost navazování vztahů mezi různými kategoriemi - vztah mezi indikátorem kompromitace a malwarem, vztah mezi identitou a hrozbou, možnost definovat kampaně a jiné. Formát STIX dokáže data reprezentovat ve dvou formátech: v serializačním textovém formátu JSON[5] a v jazyce Python. Nevýhoda tohoto formátu tkví především ve velkém množství dat, které s sebou jeden záznam nese. Příkladem lze uvést ukázku z ofi-

[1] https://www.mitre.org
[2] Common Vulnerabilities and Exposures
[3] Common Weakness Enumeration
[4] Národní institut standardů a technologie, https://www.nist.gov
[5] JavaScript Object Notation

ciální dokumentace, kde pro popis jednoho indikátoru kompromitace, v tomto příkladu škodlivá URL, je zapotřebí 46 řádkový zápis v jazyce JSON (k nahlédnutí v příloze B.1) a 31 řádkový zápis v jazyce Python (k nahlédnutí v příloze B.2).

Oproti tomu OpenIOC používá zápis v serializačním formátu XML. I tento formát s sebou nese velké množství režijních informací, jelikož se jedná o značkovací jazyk jako je například HTML[6]. Každou značku je tak potřeba začít a ukončit, což pro každou položku přidává další řetězce. Ukázku, jak zápis takového indikátoru kompromitace vypadá, je možno vidět v příloze B.3.

Formát SCAP používá také zápis ve formátu XML a tento zápis je ještě rozsáhlejší a komplexnější, než systém OpenIOC, proto jsem se rozhodl jej v této práci dále nerozebírat.

4.2 Požadavky na formát

Požadavky na výsledný formát vychází mimo jiné také ze způsobu, jakým bude výsledná aplikace využívána. Jelikož není cílem výsledné aplikace nahradit antivirové programy nebo existující komplexní nástroje pro digitální forenzní analýzu, ale poskytnutí možnosti rychlé kontroly stanic, není důležité, aby formát umožňoval komplexnější konstrukce popisu IoC, jakými jsou třeba logické operátory. Tyto komplexnější konstrukce a operace umožňují existující nástroje, které slouží k detailnější analýze postižených systémů. Nástroj, jehož návrh a implementace je cílem této práce, by tak měl fungovat na co nejjednodušším principu a k tomu mu postačuje i jednoduchý formát zápisu indikátorů kompromitace.

Hlavní požadavky formátu se dají shrnout následovně. Formát by měl být textový, lidsky čitelný a jednoduše upravovatelný. Měl by být co nejkompaktnější, zároveň by však měl umožňovat alespoň jednoduché popsání či označení indikátoru kompromitace. To může sloužit k rychlejší orientaci v případě nálezu v případě, že se v souboru budou objevovat indikátory kompromitace z více různých incidentů.

4.3 Výběr formátu

Při výběru formátu jsem zvažoval tři možnosti. První možností bylo využití některého z již existujících formátů, jako je OpenIOC nebo STIX. Druhou možností byl návrh vlastního formátu a třetí možností bylo použití některého z existujících serializačních formátů, jako je CSV, YAML nebo JSON. Vzhledem k požadavkům popsaným v podkapitole 4.2 jsem se nakonec uchýlil k třetí možnosti. Mezi hlavní důvody patří následující: všechny existující formáty, se kterými jsem se během rešerše seznámil, obsahují velké množství redundantních dat, které by nebyly využity při skenování systému a přidávaly by zbytečnou práci těm, kteří by seznamy indikátorů kompromitace vytvářeli. Práce navíc by spočívala nejen ve vyplňování dat, která nejsou nikde využita, ale navíc také ve složitém zápisu jednotlivých indikátorů kompromitace. Pro funkcionalitu aplikace pro kontrolu přítomnosti indikátoru kompromitace nám v zásadě postačí, aby indikátor kompromitace obsahoval samotnou hodnotu a kategorii, do jaké patří. Tímto způsobem lze pak v budoucnu nástroj rozšiřovat o další kategorie indikátorů kompromitace. Navíc jsem se rozhodl do formátu přidat také volitelnou informaci, která může sloužit jako poznámka pro zvýšení přehlednosti. Například u kategorie hash souboru tato poznámka může pomoci s identifikací, kde na první pohled nemusí být zřejmé, o který soubor ze zadané množiny se jedná. Tím jsou určené tři hlavní položky, které indikátor kompromitace definují: typ, hodnota, komentář. Ačkoliv

[6] HyperText Markup Language

YARA pravidla jsem se rozhodnul neimplementovat, pokusil jsem se alespoň najít a popsat způsob, jak by v budoucnu v případě potřeby mohly být implementovány.

Dalším krokem bylo vybrat patřičný serializační formát. Jelikož se jedná o jednoduchou datovou strukturu, zvolil jsem pro uchovávání formát CSV. Tento formát používá pro oddělování hodnot znak čárky, a jednotlivé položky jsou na samostatných řádcích. Hodnoty lze uzavřít do uvozovek, čímž umožňují výskyt čárky i v hodnotě. Problémem by bylo uložení hodnoty typu pravidla YARA, protože tento typ je víceřádkový řetězec. Jako řešení jsem navrhl zakódování YARA pravidla pomocí kódování `base64`. Ztratí se tím sice možnost přečíst účel YARA pravidla přímo v seznamu IoC, v tomto ohledu ale může pomoci hodnota poznámky, která může popsat YARA pravidlo slovně. Ukládání víceřádkového řetězce v textovém souboru by bylo problematické a celý systém by ztratil svou jednoduchost užívání. Jiné serializační formáty, jako jsou například YAML nebo JSON nejsou vhodné a to proto, že by pro tento účel nebyly využity datové typy, které nabízí (jako je například pole nebo slovník). Jednotlivé záznamy jsou vlastně jednoduché trojice, oddělené čárkami, na což formát CSV sedí nejlépe.

Jednoduchá databáze s indikátory kompromitace tak může vypadat následovně:

```
typ,hodnota,poznamka
MD5,058f25aeff61562c1c8064079ac43c51,malicious binary file
MD5,6c4b566fe80ec5971333fbcbd1d5eb94,malicious pam module
MD5,b7d7f1f67cd3d50558b487060707d744,malicious library
IP,127.0.1.1,malicious IPv4
IP,10.5.47.163,another malicious IPv4
IP,fe24:dead:beef:cafe::1337,malicious IPv6
URL,https://malicious.domain.com,
URL,another.malicious.domain.com,
path,/tmp/path/to/malicious/file,
path,/opt/another/path/to/malicious/file,
path,/root/yet/another/path/to/malicious/file,
```

Výpis 4.1: Ukázka zápisu indikátorů kompromitace ve formátu CSV

Kapitola 5

Návrh aplikace pro kontrolu přítomnosti indikátorů kompromitace

Tato kapitola se zabývá návrhem samotné aplikace. V předchozích kapitolách jsem popsal, které kategorie indikátorů kompromitace bude aplikace umět vyhledávat, udělal jsem souhrn existujících aplikací a formátů a vybral formát pro ukládání indikátorů kompromitace. Nyní je potřeba shrnout požadavky na aplikaci, shrnout poznatky, získané během srovnávání existujících aplikací a na základě těchto informací provést návrh. Podle tohoto návrhu pak proběhne implementace, která bude další částí této práce.

Cílem této kapitoly je vybrat programovací jazyk a provést návrh aplikace na základě získaných informací a požadavků.

5.1 Požadavky na aplikaci

Na úvod je nutno zmínit, že aplikace nemá sloužit jako komplexní antivirový systém, ale k rychlému skenování většího počtu systémů na přítomnost indikátorů kompromitace získaných od NÚKIB nebo z jiných zdrojů. Aplikace by měla mít možnosti konfigurace rozsahu skenování (jako je výběr kategorií) a nastavení hloubky prohledávání systému. Přihlašování na systémy by mělo probíhat pomocí běžně dostupného protokolu, jako je například SSH. Na konci skenování by měla aplikace uživateli vygenerovat přehlednou zprávu s výsledky skenování. Používání aplikace by mělo být jednoduché, aby uživatele zbytečně nezatěžovalo. Měla by měla fungovat s minimální uživatelskou konfigurací. Požadavkem také je, aby aplikace uměla zpracovávat indikátory kompromitace zadané ve formátu, který jsem vybral v kapitole 4.3.

5.2 Návrh vlastní aplikace

První volbou byl programovací jazyk, ve kterém bude aplikace implementována. Zvolil jsem programovací jazyk Python 3 a to z následujících důvodů:

- Je multiplatformní, takže jej lze používat na běžných operačních systémech kam patří Windows, Linux a macOS.

- Mám s ním největší zkušenosti z praxe.
- Je interpretovaný, není jej potřeba kompilovat a do kódu lze v případě potřeby dělat zásahy okamžitě.
- Je jednoduchý a přehledný.
- Nabízí pestrou škálu knihoven pro různé účely.

Nevýhodou je především výkon, vzhledem k povaze je však výkon zanedbatelný, protože hlavní výpočetně náročné operace se provádí na zkoumaných systémech, nikoliv lokálně.

Dále bylo potřeba vybrat typ architektury. Nabízely se dvě hlavní varianty. První byla použití centrálního serveru a agentů. Agenti by běželi na pozadí na všech systémech, ze kterých bychom sbírali data a na tyto agenty by se připojoval centrální server, který by agenty instruoval. Stejný princip používá například nástroj GRR (popsaný v podkapitole 3.1). Nevýhodou tohoto přístupu je ale zvýšená režie pro administrátory, protože musí nainstalovat a spravovat agenty na všech kontrolovaných stanicích. Druhou variantou bylo použití samostatné aplikace, která by se stanicemi navazovala spojení sama a to pomocí některého z běžných protokolů, které systémoví administrátoři používají, jako je SSH. Při použití tohoto přístupu by administrátorovi stačilo pouze vyplnit seznam hostů, na které by se aplikace měla připojit, vyplnit potřebné údaje pro přihlášení protokolem SSH a aplikace by se paralelně připojovala na stanice a prováděla na nich skenování. Pro mou aplikaci jsem zvolil druhou variantu, jelikož jedním z požadavkům na aplikaci je jednoduchost používání a instalace a správa agentů na systémech by tomuto požadavku neodpovídala. Komunikačním protokolem jsem zvolil SSH, jelikož to je nejčastěji používaný protokol pro vzdálenou správu počítačů s operačním systémem Linux.

Soubor, ve kterém budou definované systémy, na kterých bude prováděno skenování, se nazývá inventář. V inventáři je stanice identifikována buď IP adresou nebo doménovým jménem, přičemž platí, že na jednom řádku je definice právě jedné stanice. Kromě IP adresy nebo doménového jména může následovat i vlastní pojmenování stanice, pro snazší identifikaci. Oddělení IP adresy nebo doménového jména a vlastního názvu se provádí pomocí sekvence mezer nebo tabulátorů. Uživatel také může za IP adresu, respektive doménové jméno, zapsat i port, na kterém na daném serveru naslouchá služba SSH. Pro každý systém tak může specifikovat vlastní hodnotu portu. Pokud ji nespecifikuje, použije se buď výchozí hodnota nebo hodnota získaná z konfigurace.

Konfiguračním souborem bude textový soubor, který bude obsahovat hodnoty ve formátu `klíč=hodnota`. Jednotlivé položky budou odděleny znakem nového řádku. Uživatel bude mít možnost definovat jednotlivé položky konfigurace i pomocí přepínačů aplikace na příkazové řádce. Takto zadané hodnoty mají při vyhodnocování větší prioritu, než hodnoty v konfiguračním souboru. Uživatel bude moci nastavovat následující možnosti:

- Uživatelské jméno, pod kterým se uživatel bude připojovat.
- Cestu k SSH klíči, pomocí kterého se bude připojovat.
- Port, který bude pro připojení SSH použit.
- Cestu k souboru s inventářem.
- Cestu ke složce, do které budou vytvářeny závěrečné zprávy z testování.
- Cestu k souboru obsahujícího indikátory kompromitace

- Možnost výběru kategorií indikátorů kompromitace pro skenování (pokud ji uživatel nezadá, jsou standardně vybrány všechny kategorie).
- Rozsah skenování, který odpovídá hodnotám 1-3, kde 1 je nejméně rozsáhlý a 3 je nejvíce rozsáhlý (standardní hodnota je 1). Rozsah skenování definuje různé adresáře pro různé rozsahy, přičemž platí, že větší hloubka pokrývá adresáře z menších hloubek a rozšiřuje je o další. Čím rozsáhlejší skenování, tím delší dobu trvání vyhledávání lze očekávat.

Aplikace po ukončení skenování vytvoří soubor, ve kterém se bude nacházet závěrečná zpráva ze skenování.

Jednotlivé kroky spojené s použitím aplikace jsou následující:

1. Správce si aplikaci stáhne.
2. Stáhne si také indikátory kompromitace, které mu budou zaslány společně s aplikací.
3. Správce si na základě svojí infrastruktury, na které chce skenování provádět, vytvoří inventář.
4. V případě potřeby vytvoří konfigurační soubor nebo požadované parametry zadá přes příkazovou řádku.
5. Spustí aplikaci a vyčká, než aplikace skončí. Poté zkontroluje soubor se závěrečnou zprávou. V případě pozitivního nálezu správce tuto skutečnost oznámí příslušným autoritám.

Kapitola 6

Implementace

Cílem této kapitoly je popsat implementaci aplikace. Obsahuje popis jednotlivých částí aplikace, jejich vzájemné propojení a také úskalí, na která jsem během implementace narazil a způsoby, jakými jsem je vyřešil.

Aplikace je napsaná v jazyce Python 3 a jedinou požadovanou nestandardní knihovnou je `Paramiko` (viz 6.5.1). Neobsahuje žádné pokročilé konstrukce specifické pro tuto verzi jazyka Python a proto by v případě potřeby šla přepsat také do starší verze, tedy do verze Python 2. První polovina této kapitoly se věnuje jednotlivým třídám, které jsou součástí aplikace a mají na starosti zpracování souborů, vyhledávání a tvorbu závěrečné zprávy.. Druhá polovina se pak věnuje samotné implementaci vyhledávání, práci s SSH spojeními a zpracování indikátorů kompromitace.

Pro samotnou aplikaci je důležitá informace, že se jedná o nástroj se specifickým využitím a tím je poskytnout správcům IT infrastruktur jednoduchý a rychlý nástroj pro ověření kompromitace jejich sítí. Nástroj nemá sloužit jako antivirový program, není určen k tomu, aby se pouštěl pravidelně a aby jeho výsledky byly strojově zpracovány. Je důležité, aby jej zvládl nakonfigurovat a spustit administrátor v co nejkratším čase. Ten následně naváže komunikaci se subjektem a začne vést podrobnější digitální forenzní analýzu potenciálně infikované sítě. Nástroj byl vytvořen v reakci na zkušenosti s pomalou reakční dobou některých subjektů a klade si za cíl tyto doby zkrátit. Samotné seznamy indikátorů kompromitace také bude tvořit NÚKIB, přičemž se ve většině případů jedná o informace podléhající utajení podle zákone 412/2005 Sb. Zákon o ochraně utajovaných informací a o bezpečnostní způsobilosti[1]. Proto se nedá předpokládat, že by NÚKIB vydané indikátory kompromitace zveřejňovat, jelikož se může jednat o citlivé informace, jejichž vyzrazení by mohlo ohrozit zájmy České republiky. Správci si však mohou sestavovat vlastní seznamy indikátorů kompromitace, mají-li zájem a to díky jednoduchému formátu, který tato aplikace používá a tím je formát CSV.

6.1 Konfigurace aplikace

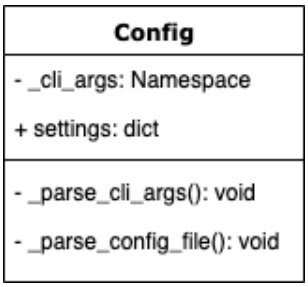
Aplikace pro fungování potřebuje inventář (seznam systémů, na kterých bude vyhledávání probíhat), seznam indikátorů kompromitace a konfiguraci zadanou v konfiguračním souboru nebo pomocí parametrů příkazové řádky.

Způsob zadávání parametrů se dá rozdělit na tři části: Tou první jsou původní hodnoty, které jsou nastavené podle nejběžnějšího způsobu použití a použijí se v případě, kdy je

[1] https://www.zakonyprolidi.cz/cs/2005-412

uživatel sám nespecifikuje. Mají tudíž nejmenší prioritu, při výběru. Druhou možností jsou hodnoty v konfiguračním souboru. Jako konfigurační soubor se považuje soubor pojmenovaný jako config, který je ve stejném adresáři jako je aplikace. Uživatel ale může pomocí přepínače `-c` nebo `--config-file` definovat jiné umístění konfiguračního souboru. Parametry v tomto konfiguračním souboru jsou zapisovány ve formátu `klíč=hodnota`. Pokud uživatel uvede nesprávnou hodnotu klíče, aplikace jej na to upozorní varovným hlášením na standardní chybový výstup a ukončí se, tím pádem nedojde k vyhledávání indikátorů kompromitace. Důvodem tohoto opatření je snížit riziko nesprávného spuštění nástroje, které by mohlo mít za cíl zbytečnou manipulaci se systémy, což by mohlo komplikovat potenciální digitální forenzní analýzu. Třetí možností je pomocí příkazové řádky při spouštění nástroje. Takto specifikované mají největší prioritu ze všech tří možností. Při spuštění aplikace se jako první věc provede právě načtení parametrů konfigurace ze všech možných zdrojů a nástroj pokračuje dále pouze v případě, že zpracování argumentů ze všech tří zdrojů proběhlo v pořádku.

Třída, která se zabývá načítáním celé konfigurace zabírá se nazývá `Config` a její definice se nachází v souboru `Config.py`. Jejími úkoly jsou načtení konfigurace ze souboru a z příkazové řádky, zpracování těchto dat a zachování priority tak, jak byla popsána v úvodu této sekce a také uložení těchto dat, aby s nimi v pozdější fázi mohly pracovat zbylé části aplikace. Třída obsahuje tři metody a to konstruktor, metodu `_parse_config_file` a metodu `_parse_cli_args`. Obsahuje také pomocný atribut `cli_args` a atribut `settings`. Konstruktor třídy volá obě metody třídy a volá se hned po spuštění programu. Jako první provede zpracování argumentů z příkazové řádky. To se provádí pomocí knihovny `argparse`[2]. Jako první se zpracovávají parametry zadané z příkazové řádky, až poté ze souboru. Poté se uloží v podobě datového typu slovník do atributu `settings`, odkud jej mohou vyčítat ostatní části aplikace. Za povšimnutí navíc stojí proměnná `DEBUG`, definovaná v tomtéž souboru, která pokud je nastavena na hodnotu `True`, vypisuje i ladící informace, jako je obsah zpracované konfigurace, inventáře a seznamu indikátorů kompromitace. Tyto informace vypisuje na standardní výstup. Navíc také do závěrečné zprávy ukládá i informace o nenalezených indikátorech kompromitace, to se v případě běžného nastavení aplikace do zprávy nezapisuje. Diagram třídy lze vidět na obrázku 6.1.



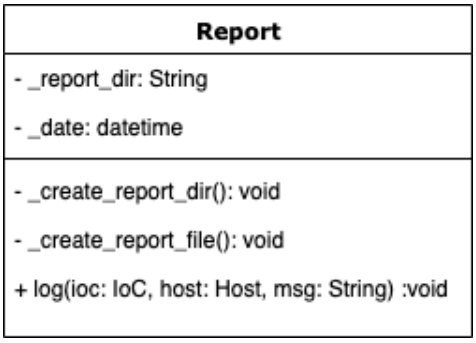
Obrázek 6.1: Diagram třídy Config.

6.2 Zpracování závěrečné zprávy

Po zpracování konfigurace se inicializuje třída `Report`, definovaná v souboru `Report.py`, která má na starosti vytvoření a zapisování do závěrečné zprávy. Tato třída obsahuje tři

[2] https://docs.python.org/3/library/argparse.html

metody a to `create_report_dir`, `create_report_file` a `log`. První jmenovaná metoda slouží pro vytvoření složky, do které se závěrečné zprávy, nebo-li reporty budou ukládat. Cesta může být zadána relativně nebo absolutně a uživatel, který aplikaci spouští musí mít práva danou složku vytvořit. Metoda `create_report_file` pak vytváří v této složce zprávu s názvem ve formátu `report_YYYY-MM-DD-HH:MM:SS.log`, kde `YYYY` je aktuální rok, `MM` je aktuální měsíc, `DD` je aktuální den, `HH` je aktuální hodina, `MM` je aktuální minuta a `SS` je aktuální sekunda. Zároveň do tohoto souboru vytvoří hlavičku pro jednotlivé sloupce, těmi jsou datum, označení systému, kategorie a hodnota nalezeného indikátoru kompromitace a zpráva, která obsahuje detailnější informace o nálezu. Takto jsou strukturovány jednotlivé záznamy vygenerované aplikací. Diagram této třídy lze vidět na obrázku 6.2



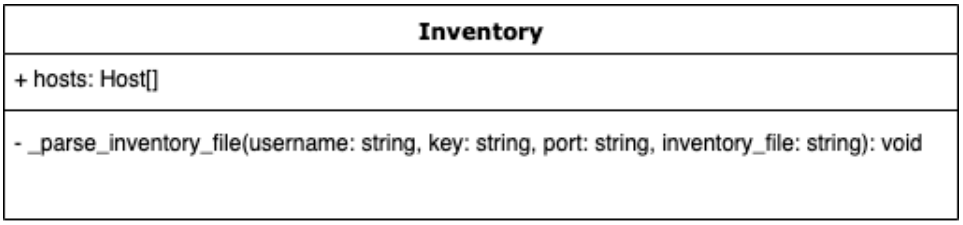
Obrázek 6.2: Diagram třídy Report.

6.3 Inventář

Dále dochází ke zpracování souboru inventáře, který obsahuje seznam IP adres (povinná část) a doménových jmen nebo poznámek (volitelná část) oddělené mezerou. Navíc může být specifikován SSH port a to tak, že se vloží za IP adresu oddělený pouze dvojtečkou. Takto zapsaný SSH port má větší prioritu než porty zadané jinými způsoby, ať už pomocí souboru nebo přes parametry příkazové řádky. Díky této funkcionalitě může uživatel pro každý systém specifikovat vlastní SSH port, pokud se mu v rámci topologie liší. Pro zpracování souboru inventáře slouží třída `Inventory`, definovaná v souboru `Inventory.py`. Metody této třídy jsou pouze konstruktor a metoda nazvaná `_parse_inventory_file`. Jediným atributem této třídy je atribut `hosts`, což je list objektů třídy `Host`. Metoda `_parse_inventory_file` používá následující parametry z konfigurace: název SSH uživatele, cesta k SSH klíči, SSH port a cesta k souboru s inventářem. Po otevření a zpracování obsahu souboru inventáře poté inicializuje pro každý systém objekt třídy `Host` a uloží jej do atributu `hosts`. V pozdější fázi aplikace se přes toto pole iteruje a vyhledávají se indikátory kompromitace. Diagram této třídy lze vidět na obrázku 6.3.

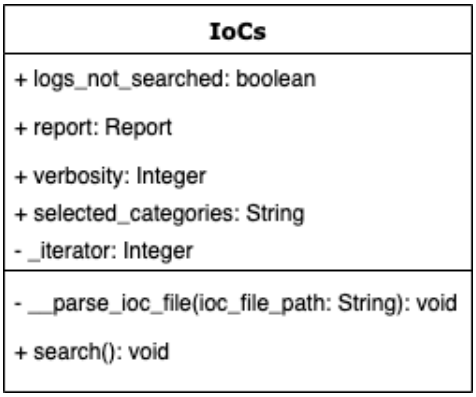
6.4 Seznam indikátorů kompromitace

Poslední částí zpracování vstupních dat je zpracování souboru obsahující indikátory kompromitace. V kapitole návrhu (viz 4.3) jsem vybral jako formát pro distribuci indikátorů kompromitace textový formát CSV, tzn. hodnoty oddělené čárkami. Pro zpracování souboru s indikátory kompromitace a pro následnou manipulaci s těmito daty slouží třída



Obrázek 6.3: Diagram třídy Inventar.

`IoCs`, definovaná v souboru `IoCs.py`. Tato funkce má kromě konstruktoru také nadefinované metody tzv. Python iterátorů, které slouží k tomu, aby se přes objekt dané třídy dalo iterovat. Navíc má také metodu sloužící pro otevření souboru s indikátory kompromitace a jejich zpracování a tou je metoda `_parse_ioc_file`. Tato metoda naplní poté atribut `array`, což je pole, objekty třídy `IoC`. Metoda `search` poté iteruje přes jednotlivé indikátory kompromitace a provádí jejich vyhledávání, více o tom procesu je rozepsáno v kapitole 6.6. Během zpracování se také kontroluje přítomnost následujících znaků v hodnotě indikátoru kompromitace: ;, |, &, ’, ". Pokud je daný znak nalezen, je program přerušen a na standardní chybový výstup se vypíše hláška pro uživatele informující o existenci nepovoleného znaku. Toto opatření bylo zavedeno, aby nebylo možné omylem ukončit provádění některého příkazu, protože vyhledávání indikátorů kompromitace se provádí voláním příkazů v jazyce Bash. Tyto znaky by pak mohly změnit chování volaného příkazu, což by vedlo k nevalidnímu výstupu a nesprávnému fungování aplikace. Diagram třídy IoCs lze vidět na obrázku 6.4.



Obrázek 6.4: Diagram třídy IoCs.

6.5 Připojování na systémy

Tato sekce je rozdělena na dvě části, v první je krátké seznámení s Python knihovnou `Paramiko`, která se v této práci používá pro připojování na systémy přes protokol SSH a pro volání a sběr výstupů příkazů, které se volají rovněž přes protokol SSH. V druhé části sekce je poté více rozepsáno jak je knihovna implementována a jaké možnosti to v rámci aplikaci nabízí.

6.5.1 Knihovna Paramiko

`Paramiko` je knihovna pro jazyky Python 2 a Python 3 a implementuje SSHv2 protokol a to jak klientskou část tak i serverovou část. Využívá rozšíření Pythonu v jazyce C pro zpracování nízkoúrovňové kryptografie. Existuje od roku 2003 a za tu dobu se stala nejpopulárnější knihovnou pro práci s protokolem SSH v jazyce Python [7]. V této práci se používá pro připojování se na systémy a na spouštění příkazů, sběr jejich výstupů a návratových hodnot. Používá se pouze klientská část, v praxi to znamená třída SSHClient. Použití této knihovny je velice jednoduché a pro uživatele intuitivní. Alternativou pro knihovnu `Paramiko` je pak knihovna `ssh2-python`[3], tu jsem nakonec nezvolil kvůli složitějšímu používání a menší podpoře od vývojářů. Jak je vidět na jednotlivých repozitářích, `Paramiko` má přes tři tisíce commitů od více než sto přispěvatelů, kdežto `ssh2-python` má přes sto commitů od pět přispěvatelů. Během implementace a testování nástroje jsem nezaznamenal žádné problémy, které by tato knihovna způsobovala a s jejím fungováním jsem byl spokojen.

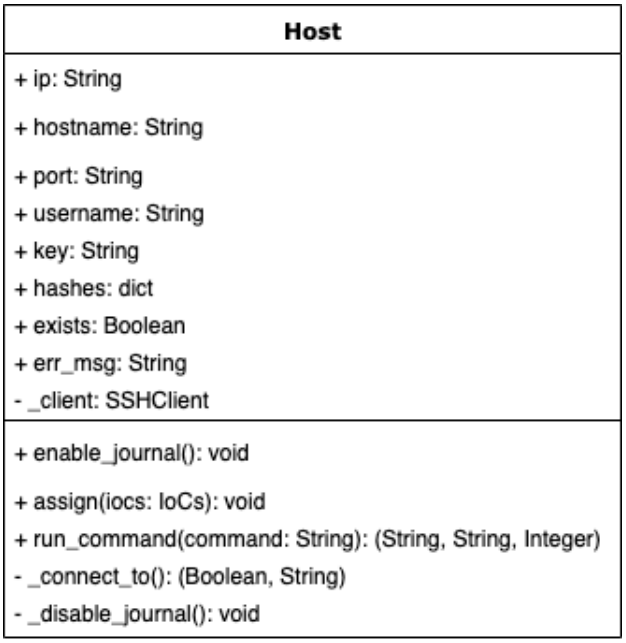
6.5.2 Implementace připojení v aplikaci

Připojování, odpojování, spouštění příkazů a ukládání pomocných dat řeší třída `Host`, definovaná v souboru `Host.py`. Třída má několik atributů a metod, viz třídní diagram na obrázku 6.5. Mezi důležité metody, z hlediska implementace patří metoda `connect_to`, která inicializuje objekt třídy `SSHClient` z knihovny `Paramiko` a načte SSH klíč, který uživatel definuje buď v konfiguračním souboru nebo pomocí parametru na příkazové řádce. Poté se pokusí připojit na server (parametry IP, port a uživatelské jméno bere z konfigurace). V případě neúspěchu se program neukončuje, uživateli se zobrazí varovná hláška o nedostupnosti daného systému. U systému na kterých běží démon `systemd` se navíc ještě kontroluje stav systémové služby `systemd-journald`. Pokud tato služba běží, je pozastavena a zapíná se znovu až po ukončení vyhledávání indikátorů kompromitace. Důvodem této funkcionality je fakt, že žurnál `systemd` ukládá logy systému do binárního souboru a v případě, že uživatel spustí aplikaci pro vyhledávání, zapíší se tyto události do systémového logu a při druhém spuštění poté nastával false positive výsledek. U textových logů je tohle vyřešeno ověřením, že nalezený záznam není náhodou pouze záznam spuštění příkazu pro hledání indikátoru kompromitace. U binárních logů toto není možné, proto jsem jako řešení implementoval dočasné vypnutí logovacího démona. Spouštění příkazu se provádí pomocí metody třídy `SSHClient exec_command`, která vrací objekty typu `Channel`, reprezentující standardní vstup, výstup a chybový výstup. Z toho objektu si aplikace uloží návratový kód příkazu a převede na řetězec s kódováním UTF-8 jak standardní výstup, tak i standardní chybový výstup, které potom tato metoda vrací v podobě datového typu tuple. Odpojení od systému je řešeno v destruktoru třídy, kde se volá metoda třídy `SSHClient close`, která korektně ukončí SSH spojení.

6.6 Vyhledávání indikátorů kompromitace na systémech

Poslední a nejdůležitější částí aplikace je samotné vyhledávání indikátorů kompromitace na systémech. Jedná se o nejsložitější část celé aplikace. Každá kategorie indikátoru kompromitace má přiřazenou vlastní třídu, ve které je vyhledávání implementováno. Všechny tyto třídy mají jednu společnou mateřskou třídu a tou je třída `IoC`. Tato třída i třídy jednotlivých kategoriích jsou všechny definovány v souboru `IoC.py`. Grafickou reprezentaci tříd

[3] https://github.com/ParallelSSH/ssh2-python



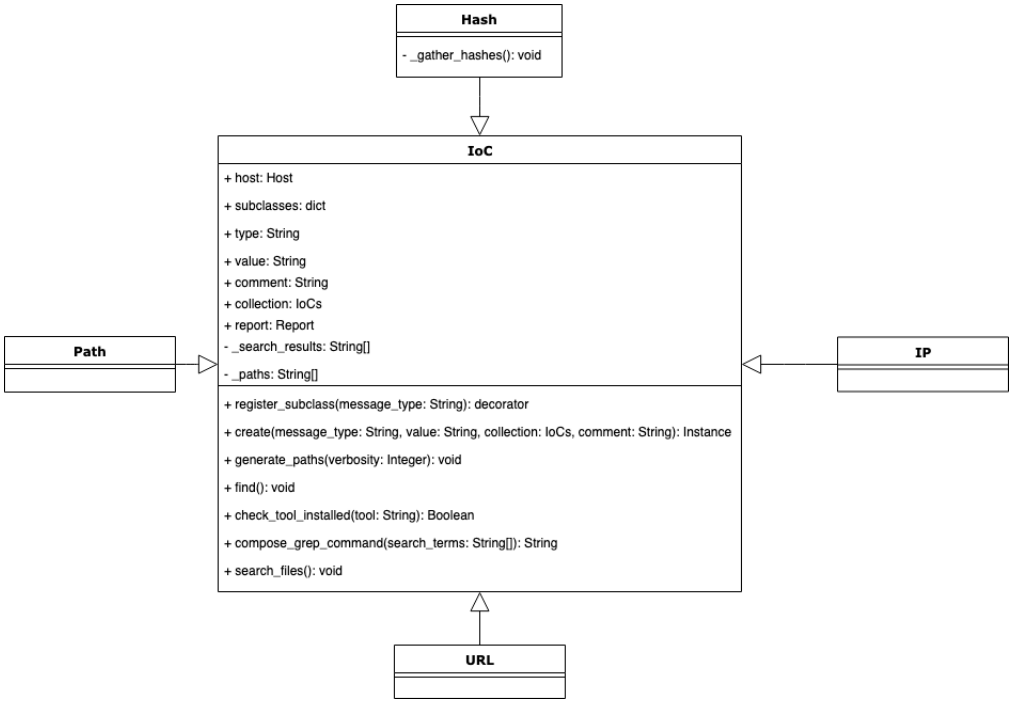
Obrázek 6.5: Diagram třídy Host.

a jejich vzájemných vztahů lze vidět na obrázku 6.6. Jednotlivým metodám vyhledávání různých kategorií indikátorů kompromitace se věnují podsekce níže.

Společná třída `IoC` funguje jako návrhový vzor Factory (česky též Továrna) a slouží ke správné inicializaci objektu podle kategorie indikátoru kompromitace. V praxi to znamená, že pokud ačkoliv v kódu inicializuji objekt třídy `IoC`, tato inicializace vrací objekt z jedné z následujících tříd: `Path`, `Hash`, `URL`, `IP`. Tuto funkcionalitu zajišťují dekorátor `register_sublclass` a metoda `create`, kde dekorátor slouží k zaregistrování kategorie indikátoru kompromitace jako vlastní třídy a metoda `create` poté na základě typu, předkládaného jako parametr, inicializuje a vrací tento objekt původnímu konstruktoru třídy `IoC`. Výhodou tohoto řešení je snadné rozšiřování kategorií indikátorů kompromitace, stačí nadefinovat novou třídu s dekorátorem a třída `IoC` se poté sama postará o to, aby se vytvářel správný objekt. Objekty této třídy v sobě udržují také referenci na objekt třídy `Report`, ta slouží pro zápis výsledků do závěrečné zprávy. Obsahují také referenci na objekt třídy `IoCs` a to kvůli optimalizaci vyhledávání v souborech, více o této optimalizaci je rozepsáno v podsekci 6.6.3. V neposlední řadě se pak stará o sestavení seznamu cest, ve kterých vyhledávání probíhá a to také z důvodu optimalizace celého procesu vyhledávání. Třída též obsahuje pomocné metody sloužící více podtřídám. Mezi ně patří metoda `check_tool_installed`, která ověřuje, zda-li je nástroj předložený parametrem nainstalován, dále `compose_grep_command` který se používá k vytvoření příkazu pro vyhledávání v souborech, více o něm v podsekci 6.6.3. Také metoda `search_files`, která implementuje samotné vyhledávání v souborech je více rozepsána v podsekci 6.6.3. Poslední metodou je pak metoda `run_command`, která na systému volá příkaz zadaný jako parametr. Důležitým poznatkem je, že všechny příkazy jsou volány jako administrátor, tudíž je před každý příkaz vkládán příkaz sudo. Diagram třídy je k vidění na obrázku 6.6.

Uživatel má možnost vybrat hloubku vyhledávání a to pomocí parametru `--verbose`. V případě, že nechá původní hodnotu, vyhledávání v souborech a počítání hashů souborů probíhá ve složkách `/var/log` a `/etc`. Pokud zvolí vyšší hloubku vyhledávání, prohledávání

probíhá i ve složkách `/home`, `/root`, `/var` a `/tmp`. Při volbě nejvyšší hloubky vyhledávání, jsou navíc zahrnuty i složky `/usr`, `/opt`, `/bin` a `/lib`. S větší hloubkou vyhledávání ale narůstá doba, jakou vyhledávání trvá. Pokud ale například uživatel chce vyhledávat pouze indikátory kompromitace typu cesta souboru a IP adresa a nezajímají ho hashe souborů, může vyhledávat pouze tyto kategorie IoC a to pomocí přepínače `--selected-categories`.



Obrázek 6.6: Diagram třídy IoC a jejich dceřiných tříd.

6.6.1 Cesta souboru

První a nejjednodušší kategorií indikátorů kompromitace je cesta souboru. Třída implementující tuto kategorii se nazývá `Path`. Hledání se provádí jednoduchou konstrukcí v jazyce Bash a to `[ -f '%s' ] || [ -L '%s' ]`. Za `%s` se dosadí zkoumaná cesta k souboru. V případě, že soubor na dané cestě existuje a to buď v podobě souboru nebo v podobě symbolického odkazu, návratová hodnota příkazu se rovná nule. V takovém případě se tato událost zapíše do závěrečné zprávy, v opačném případě se neděje nic. Důvodem, proč se používají dvě testovací konstrukce, jedna s argumentem `-f` a druhá s `-L` je takový, že Bash se během testování choval k symlinkům jinak na starších systémech Debian, než na ostatních systémech (více o testování v kapitole 7). Tato konstrukce byla ověřena jako funkční na všech systémech z testovacího prostředí. Druhou variantou, jak toto vyhledávání realizovat, by pak bylo zavolání například příkazu `ls` nebo příkazu `find`, možností je v tomto případě více.

6.6.2 Hash souboru

Druhou kategorií, je hash souboru, nebo-li jeho kontrolní součet. Odpovídající třída se nazývá `Hash`. Jako algoritmus hashovací funkce v této práci byl vybrán algoritmus MD5. Nástroj, běžně dostupný a instalovaný na operačních systémech Linux, který umožňuje

spočítat MD5 hash ze souboru se nazývá `md5sum` a tento nástroj je ve výsledné aplikaci použit. Proces hledání tohoto indikátoru kompromitace probíhá následovně: pro všechny vybrané složky, jejich podsložky (rekurzivně) a soubory v nich se spočítá MD5 hash a uloží do slovníku, který je uložen v atributu objektu třídy `Host`, kde klíčem je cesta a hodnotou je vypočtený MD5 hash. Po sesbírání všech hashů se projde celý slovník a hledá se shoda s hashem daného indikátoru kompromitace. V případě nálezu se tato skutečnost zapíše do závěrečné zprávy. V tomto záznamu je pak uvedena i absolutní cesta dotyčného souboru.

Vyhledávání tohoto typu indikátoru kompromitace je časově a výpočtově nejnáročnější ze všech zvolených kategorií. Proto je samotné počítání hashů souborů prováděno pouze jednou a to při prvním nálezu indikátoru kompromitace typu MD5. Jakékoliv další nálezy poté pak už pouze prohledávají slovník s hashi, který je uložený v objektu třídy `Host`, tudíž je společný pro všechny objekty třídy `Hash`. Tím se značně zrychlí celý proces vyhledávání především pro větší seznamy indikátorů kompromitace. Samotný proces spouštění výpočtu hashe souboru obstarává Linuxový nástroj `find`, který pro zadanou složku umožňuje rekurzivně vyhledat všechny soubory v ní a také nad nimi spustit příkaz, v tomto případě se jedná o příkaz `md5sum`. Jako další použitá optimalizace je omezení maximální velikosti souboru, pro které se kontrolní součty počítají. Při této optimalizaci jsem vycházel ze skutečnosti, že více než 97% malware je menší než 1 MB [10] a proto je původní hodnota maximální velikosti nastavena právě na ten 1 MB. Pro větší úrovně hloubky vyhledávání je pak tato hranice posunuta na 5 MB, 10 MB respektive 100 MB pro největší hloubku vyhledávání.

6.6.3 URL a doménové jméno

URL a doménové jméno, jsou v aplikaci sdruženy do jedné kategorie a odpovídá jim třída `URL`. Pro vyhledávání je použita metoda `search_files`. Vyhledávání v souborech bylo druhou nejvíce časově náročnou kategorií indikátorů kompromitace, a proto byly optimalizace provedeny i zde. Výsledkem optimalizace je, že se vyhledávání provádí pouze jednou, implementačně je to však řešeno jinak než u indikátorů kompromitace typu hash. Při prvním pokusu o hledání indikátoru kompromitace typu URL, doménového jména nebo IP adresy (vyhledávání IP adresy také používá tuto funkci, viz podsekce 6.6.4), sestaví se příkaz tak, aby vyhledávání mohlo být provedeno jediným příkazem. K tomu slouží metoda `compose_grep_command`, která vygeneruje příkaz `grep` se všemi potřebnými parametry. To zahrnuje všechny hledané indikátory kompromitace a všechny cesty složek, ve kterých se vyhledává. Cesty složek generuje konstruktor třídy `IoC` a všechny hledané indikátory kompromitace se získají díky referenci na objekt třídy `IoCs`, ze kterého je možné projít všechny indikátory kompromitace a vytvořit seznam pouze z těch typu IP a URL. Po sestavení a spuštění `grep` příkazu na systému se jeho výstup uloží a jednotlivé objekty tříd `URL` a `IP` poté prohledávají pouze tento výstup. Pro každý nalezený výskyt hodnoty indikátoru kompromitace je do závěrečné zprávy vytvořen záznam o nálezu, ve kterém je informace o souboru, kde byl nalezen a je zde i obsah celého řádku s nálezem.

6.6.4 IP adresa

Poslední kategorií je IP adresa, které odpovídá třída `IP`. IP adresa se vyhledává v souborech stejným způsobem jako URL a doménová jména, popsané v podsekci 6.6.3. Kromě toho navíc probíhá vyhledávání také ve výstupech příkazů `netstat` a `ss`. Cílem hledání i v těchto nástrojích je ověření, že s danou IP adresou není navázáno síťové spojení. Prohledávání v souborech pak může odhalit přítomnost škodlivé IP adresy v konfiguračních souborech

nebo v záznamech systému nebo aplikací. V případě nálezu výskytu IP adresy ve výstupu příkazů `netstat` nebo `ss` je tato skutečnost zapsána do závěrečné zprávy.

Kapitola 7

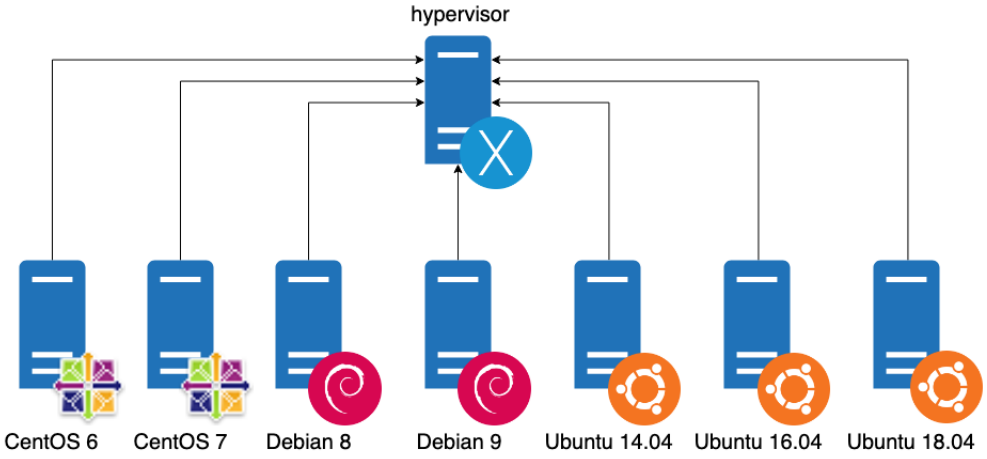
Testování

Testování aplikace lze rozdělit na dvě části. Tou první je průběžné testování aplikace v průběhu samotného vývoje a tou druhou testování po vývoji. Testování v průběhu vývoje se zaměřovalo především na ověření funkčnosti nástroje. Ověřovalo se jednak to, že nová funkcionalita funguje jak bylo zamýšleno, a zároveň že nepřestala fungovat žádná z již existujících funkcí aplikace. Dalším cílem lokálního testování bylo odhalovaní slabých míst aplikace zejména v oblasti rychlosti a výkonu. Pomocí tohoto testování se podařilo výrazně zrychlit fungování celé aplikace tak, aby byla použitelná i v reálném prostředí. Více informací včetně naměřených výsledků je v sekci 7.1. Aplikace byla po dokončení testována také na větším prostředí, konkrétně na infrastruktuře jednoho kybernetického cvičení. Během tohoto testování se ověřovala jak funkčnost nástroje, tak i výkon při vyhledávání ve větší infrastruktuře, než byla ta lokální. Výhodou využití tohoto prostředí je také to, že na strojích byly nainstalovány zranitelnosti a škodlivé programy, tudíž šlo vytvořit i reálnější seznam indikátorů kompromitace, než ten, který byl použit při lokálním testování. Více informací o topologii a výsledcích tohoto testování je rozepsané v sekci 7.2.

7.1 Lokální testování

Pro lokální testování bylo v první řadě potřeba vybrat způsob, jakým bude testování prováděno. Při výběru jsem kladl důraz na taková řešení, která by přinášela minimální režii, umožňovala by testování na různých distribucích operačního systému Linux a byla spolehlivá. Finální rozhodování probíhalo mezi použitím virtuálních strojů nebo kontejnerů. Nakonec jsem vybral virtuální stroje a to hlavně kvůli jejich věrnější reprezentaci fyzických strojů. Dalším krokem při budování testovacího prostředí byla platforma pro práci s virtuálními stroji. Opět byla cílem především jednoduchost a spolehlivost, kde obě tyto podmínky splňoval nástroj Vagrant[1]. Tento nástroj slouží k jednoduchému nasazování předpřipravených obrazů s různými operačními systémy a je primárně zaměřen na vývojáře, kteří při kolaborativní spolupráci mohou využívat totožné prostředí pro vývoj, testování a nasazování aplikací. Pomocí tohoto nástroje jsem si tak na svém lokálním počítači rozjel jednoduchou topologii se 7 stroji s následujícími Linuxovými distribucemi: CentOS 6, CentOS 7, Debian 8, Debian 9, Ubuntu 14.04, Ubuntu 16.04 a Ubuntu 18.04. Tímto výběrem jsem se snažil pokrýt většinu běžně používaných Linuxových distribucích a to jak na serverech tak i na pracovních stanicích uživatelů. Topologie je k vidění na obrázku 7.1.

[1] https://www.vagrantup.com/



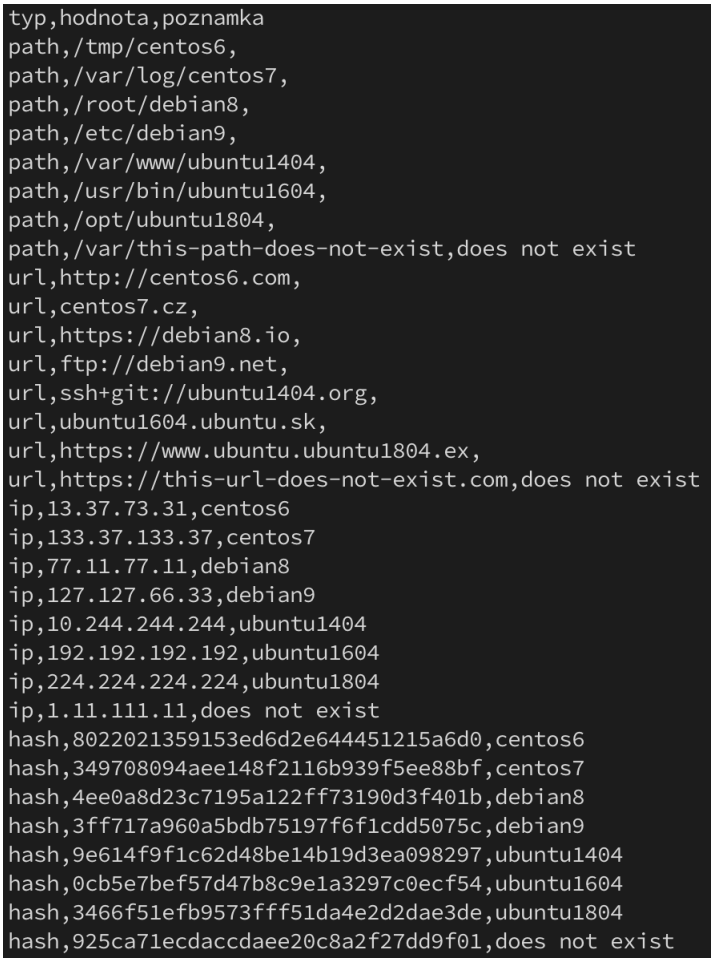
Obrázek 7.1: Topologie používaná pro lokální testování.

Na základě topologie vytvořené nástrojem Vagrant jsem sestavil inventář pro mou aplikaci, který je k vidění na 7.2. Důvodem, proč je všude na pozici IP adresy adresa lokálního stroje je ten, že Vagrant v základním nastavení nepoužívá veřejné rozhraní pro virtuální stroje, ale provoz tuneluje, včetně SSH komunikace. Proto se pro připojení používá localhost, s různými porty pro různé stroje. Na stroje potom byly nahrány nebo vytvořeny různé indikátory kompromitace, jejich soupis je možné vidět na obrázku 7.3. Snahou bylo pokrýt všechny operační systémy všemi typy kategorií i pro různě zvolené hloubky vyhledávání. Jsou zde obsaženy i neexistující záznamy a to kvůli ověření, že aplikace nehlásí false positive. Aplikace pak byla pravidelně s těmito daty pouštěna aby se ověřovala funkčnost, tedy že najde indikátory kompromitace, které najít má a nehlásí žádné false positive.

```
localhost:2200 centos6
localhost:2222 centos7
localhost:2201 debian8
localhost:2202 debian9
localhost:2204 ubuntu1404
localhost:2205 ubuntu1604
localhost:2206 ubuntu1804
```

Obrázek 7.2: Inventář používaný pro lokální testování.

Druhým cílem lokálního testování pak bylo měřit rychlost a výkon vyhledávání, najít slabá místa, které by se daly zoptimalizovat a následně ověřit efektivitu implementované optimalizace. Toto testování se provádělo v průběhu vývoje i po jejím dokončení, přičemž probíhalo i srovnání se staršími verzemi aplikace. Měření bylo vykonáno desetkrát, do tabulky byla poté zapsána průměrná hodnota měření. Různé byly také množiny testovacích dat, a to jak inventáře, tak i množina indikátorů kompromitace. Při testování jedné kategorie se použila množina s jedním výskytem, s deseti výskyty a se sto výskyty dané kategorie na systému.



```
typ,hodnota,poznamka
path,/tmp/centos6,
path,/var/log/centos7,
path,/root/debian8,
path,/etc/debian9,
path,/var/www/ubuntu1404,
path,/usr/bin/ubuntu1604,
path,/opt/ubuntu1804,
path,/var/this-path-does-not-exist,does not exist
url,http://centos6.com,
url,centos7.cz,
url,https://debian8.io,
url,ftp://debian9.net,
url,ssh+git://ubuntu1404.org,
url,ubuntu1604.ubuntu.sk,
url,https://www.ubuntu.ubuntu1804.ex,
url,https://this-url-does-not-exist.com,does not exist
ip,13.37.73.31,centos6
ip,133.37.133.37,centos7
ip,77.11.77.11,debian8
ip,127.127.66.33,debian9
ip,10.244.244.244,ubuntu1404
ip,192.192.192.192,ubuntu1604
ip,224.224.224.224,ubuntu1804
ip,1.11.111.11,does not exist
hash,8022021359153ed6d2e644451215a6d0,centos6
hash,349708094aee148f2116b939f5ee88bf,centos7
hash,4ee0a8d23c7195a122ff73190d3f401b,debian8
hash,3ff717a960a5bdb75197f6f1cdd5075c,debian9
hash,9e614f9f1c62d48be14b19d3ea098297,ubuntu1404
hash,0cb5e7bef57d47b8c9e1a3297c0ecf54,ubuntu1604
hash,3466f51efb9573fff51da4e2d2dae3de,ubuntu1804
hash,925ca71ecdaccdaee20c8a2f27dd9f01,does not exist
```

Obrázek 7.3: Indikátory kompromitace používané pro lokální testování.

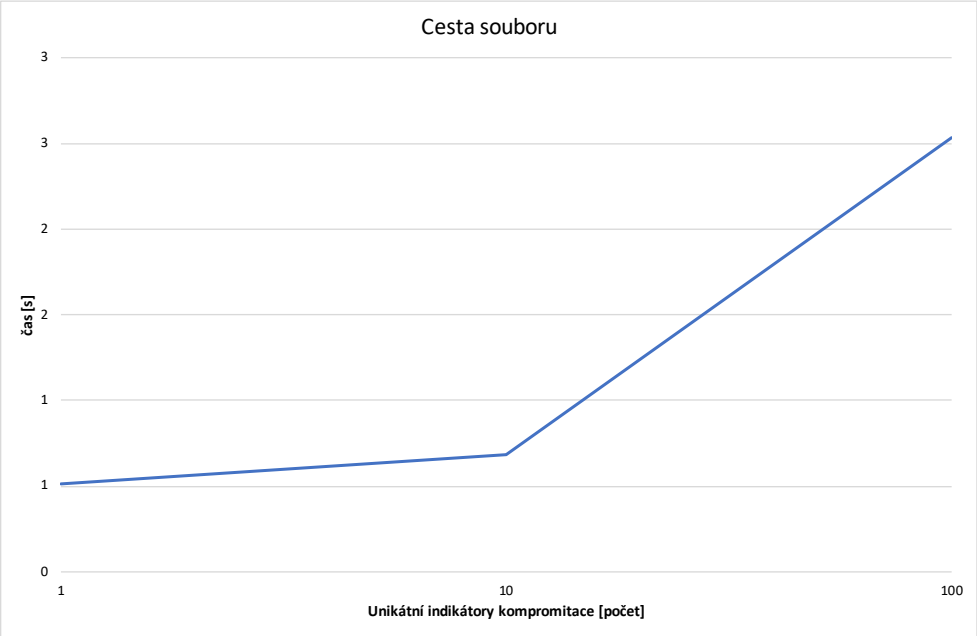
7.1.1 Popis testovacího prostředí

Testování bylo prováděno na laptopu s operačním systémem macOS 10.14.5 a následujícími parametry: procesor 2,2 GHz Intel Core i7, paměť 16 GB 2400 MHz DDR4, pevný disk 512 GB Flash Storage, grafická karta Radeon Pro 560X 4 GB. Virtuální stroje, na kterých bylo vyhledávání prováděno, měly následující parametry: 1 jádrový procesor, paměť 500 MB, disk 40 GB. V průběhu testování byly vypnuty všechny nepotřebné aplikace a služby, aby neovlivňovaly výsledky měření. Mezi jednotlivými měřeními byla pauza minimálně do doby, než se přestaly točit větráky na testovacím laptopu.

7.1.2 Testování rychlosti vyhledávání jednotlivých kategorií IoC

Path

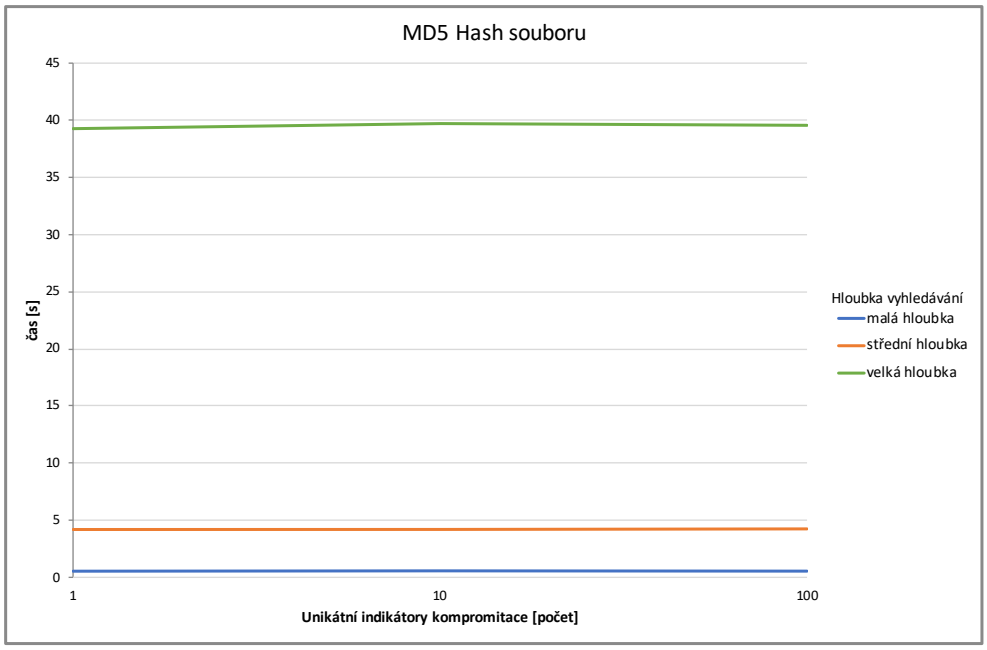
Prvním indikátorem kompromitace, na kterém testování rychlosti probíhalo, byla cesta souboru. Vyhledávání tohoto indikátoru kompromitace bylo implementováno nejdříve a tato implementace se po celou dobu vývoje neměnila. Výsledky měření je možno vidět na obrázku 7.4. Měření pro různé hloubky zde není relevantní, protože to nijak funkci neovlivňuje. Na této kategorii indikátoru kompromitace je také nejvíce patrný rozdíl při rostoucím počtu množiny indikátorů kompromitace tohoto druhu. Při použití jednoho indikátoru kompromitace byla průměrná doba trvání aplikace 0,513 sekund, při použití deseti pak 0,687 sekund a při použití sta pak dokonce 2,533 sekund. To je zapříčiněno tím, že pro každý indikátor kompromitace toho druhu se volá samostatný příkaz, který využívá vlastní session. U ostatních kategorií po provedení optimalizací se většinou příkaz volá nejvýše jednou. Vzhledem k očekávaným počtům indikátorů kompromitace a nárůstu času trvání aplikace však optimalizace vyhledávání tohoto typu není nutná.



Obrázek 7.4: Graf popisující vztah počtu indikátorů kompromitace typu cesta souboru a rychlost vyhledávání.

Hash

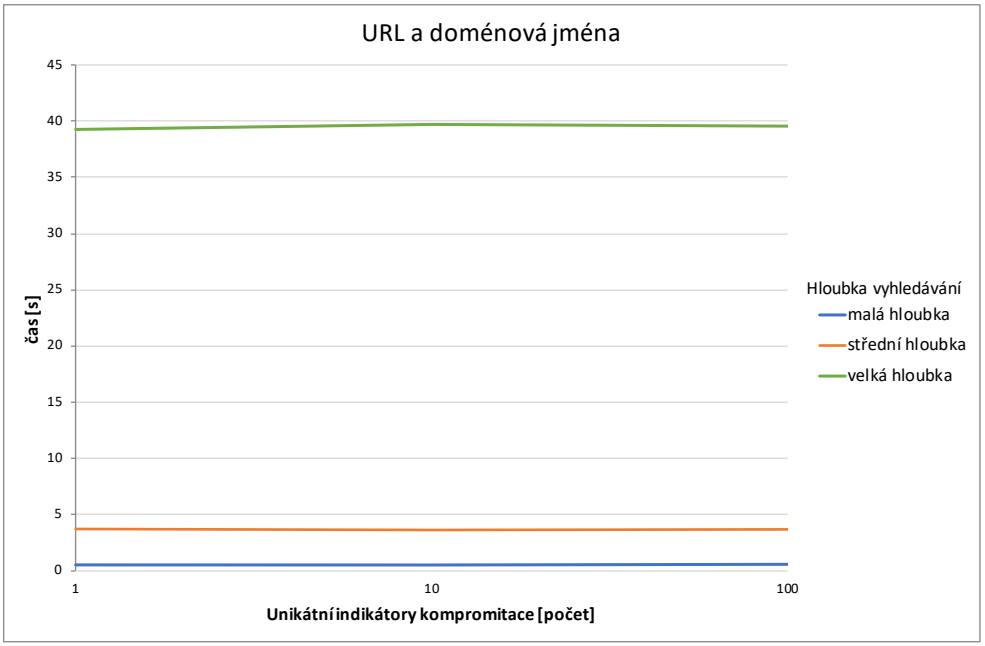
Druhou kategorií je MD5 hash. Zde původní implementace počítala hashe souborů v systému pro každý výskyt znova, což mělo drastický dopad na rychlost a stabilitu celé aplikace. Poté co byla funkce pro vyhledávání přepsána, aby hashe souborů počítala nejvýše jednou, časy se výrazně zlepšily, viz obrázek 7.5. Zde je z měření patrno, že největší vliv na dobu trvání má prohledávaná hloubka. Počet indikátorů kompromitace zde nemá téměř žádný vliv. Rozdíl by pravděpodobně bylo možné pozorovat až po použití řádově vyšších počtů indikátorů kompromitace, kdy by se k době vyhledávání připočetla režie vyhledávání ve slovníku. Bez optimalizace by toto vyhledávání trvalo řádově déle.



Obrázek 7.5: Graf popisující vztah počtu indikátorů kompromitace typu hash souboru a rychlost vyhledávání.

URL

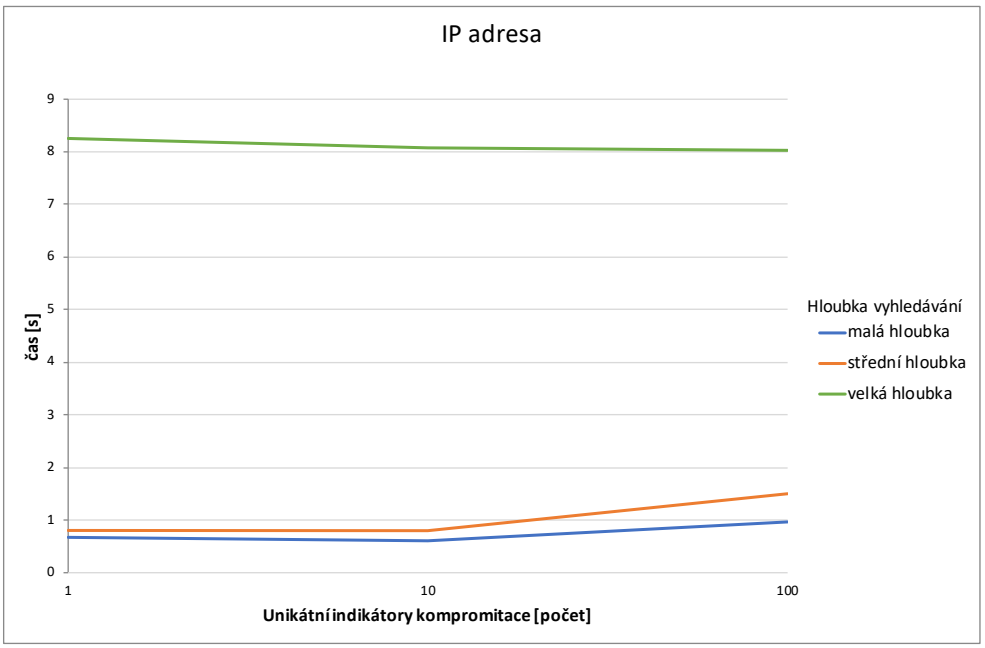
I u třetí kategorie, kterou je URL a doménové jméno došlo k výrazné optimalizaci rychlosti. Původní implementace prováděla vyhledávání pro každý indikátor kompromitace zvlášť, pro každou složku zvlášť. Nová verze agreguje všechny složky a všechny hodnoty, které má vyhledávat a tudíž k samotnému hledání dochází pouze jednou. Výsledek je poté zpracován v aplikaci. Vliv na rychlost aplikace je patrný na obrázku 7.6. Opět i zde lze pozorovat malý vliv počtu indikátorů kompromitace na délku vyhledávání. Samotný příkaz pro vyhledávání se totiž spouští pouze jednou a samotné výskyty se hledají ve výstupu volaného příkazu, tudíž se jedná pouze o vyhledávání v řetězci. Největší vliv na dobu vyhledávání zde opět má hloubka vyhledávání.



Obrázek 7.6: Graf popisující vztah počtu indikátorů kompromitace typu URL a doménové jméno a rychlost vyhledávání.

IP

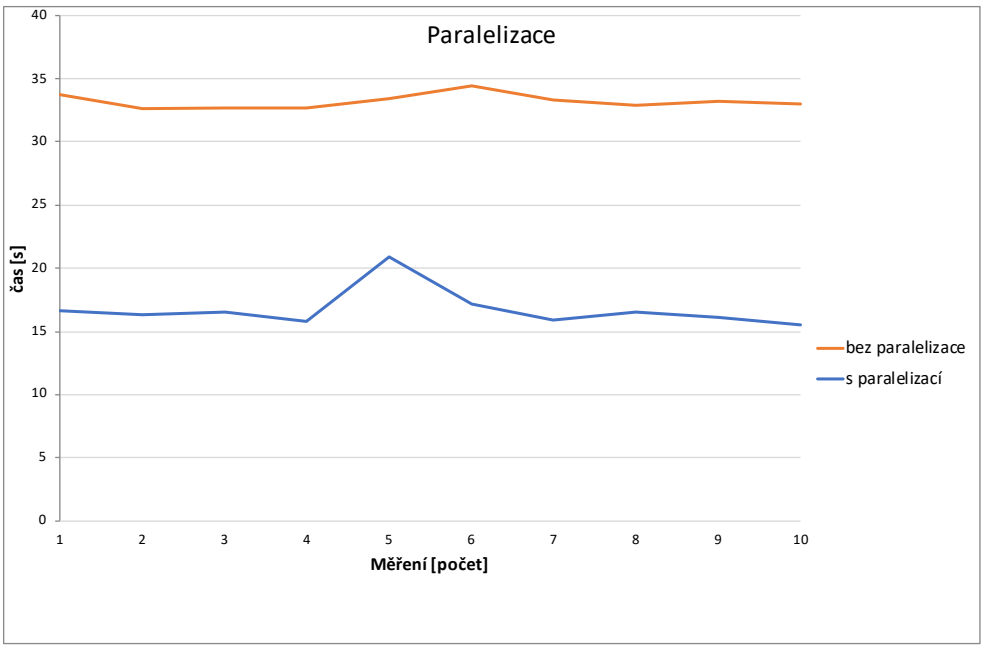
U poslední kategorie, kterou je IP adresa, platí, že se vyhledává v souborech společně s URL a doménovými jmény a navíc ještě ve výstupech nástrojů `netstat` a `ss`. Na všech testovacích strojích byly oba tyto nástroje nainstalovány. Zde lze vidět, že se pro větší počet indikátorů kompromitace také zvedá doba trvání. To je způsobeno neoptimálním vyhledáváním, kdy je pro každý indikátor kompromitace volán nástroj netstat i ss. Doba, kterou to přidává je však zanedbatelná. Jinak jsou časy podobné jako v případě vyhledávání URL. Výsledky lze pozorovat na grafu na obrázku 7.7.



Obrázek 7.7: Graf popisující vztah počtu indikátorů kompromitace typu IP a rychlost vyhledávání.

Paralelizace

Pro všechny tyto testy platí, že byly spuštěny už v době, kdy byla jedna z nejzásadnějších optimalizačních technik využita a to paralelizace funkce spouštění vyhledávání na systémech. První verze aplikace totiž postupovala sekvenčně - vyhledala všechny indikátory kompromitace na jednom stroji, poté na druhém, třetím, atd. Nová verze používá knihovnu threading a funkce, starající se o vyhledání IoC, se pouští v samostatném vlákně paralelně. Tato optimalizace měla na celkové zrychlení patrně největší dopad. Výsledky testování implementace s paralelizací a bez paralelizace je možné vidět na obrázku 7.8. Pro testování bylo použito 5 různých virtuálních strojů.

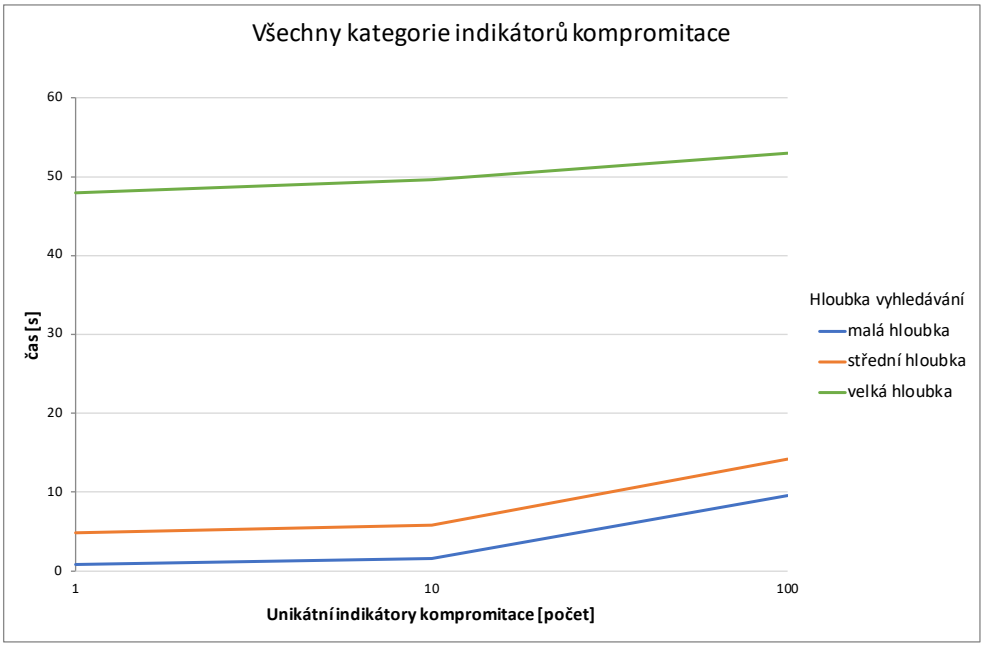


Obrázek 7.8: Graf ukazující vliv paralelizace na rychlost vyhledávání.

Dalším pokusem o optimalizaci byla paralelizace spouštění příkazů i v rámci jednoho systému. Tedy aby se všechny příkazy pro vyhledání všech indikátorů kompromitace spustily naráz. Využita k tomu byla stejná metoda, jako v případě paralelizace spouštění funkce vyhledávání na systémech. Zde jsem narazil na limity protokolu SSHv2, které v původním nastavení limitují maximální počet spojení z jednoho systému na deset. Knihovna `Paramiko` používá pro spouštění každého příkazu nové vlastní spojení a tedy při spuštění více příkazů se otevírá i více spojení, což v momentě větší počtu indikátorů kompromitace mělo za následek havárii aplikace. Naštěstí však i bez použití této optimalizace dosahovala aplikace velmi dobrých výsledků ve všech testech.

Vše

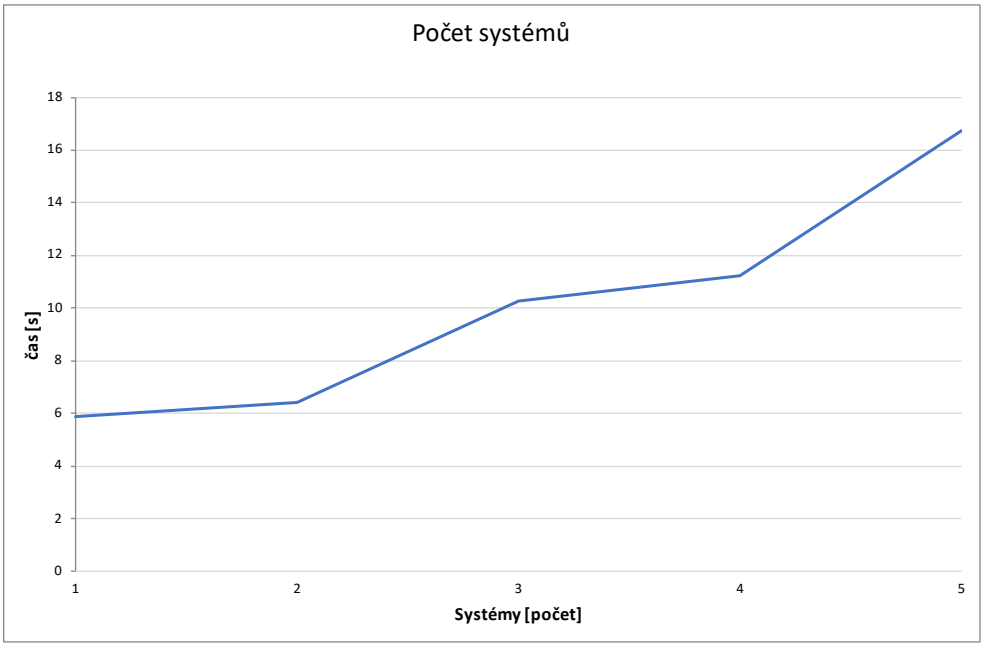
Na obrázku 7.9 je k vidění srovnání doby trvání při použití všech kategorií indikátorů kompromitace, přičemž od každého typu je použit buď jeden, deset nebo sto různých indikátorů kompromitace. Celkový počet unikátních indikátorů kompromitace pro každý test tedy je čtyři, čtyřicet nebo čtyři sta. Při použití čtyř set indikátorů kompromitace s největší hloubkou vyhledávání byl průměrný čas vyhledávání 52 sekund. Je však potřeba brát v potaz, že vyhledávání probíhalo na virtuálním stroji běžícím lokálně, na SSD disku. Na vzdálených fyzických strojích, které používají klasické plotnové disky tak doba vyhledávání bude tak delší.



Obrázek 7.9: Graf popisující vztah počtu indikátorů kompromitace a rychlost vyhledávání.

Počet systémů

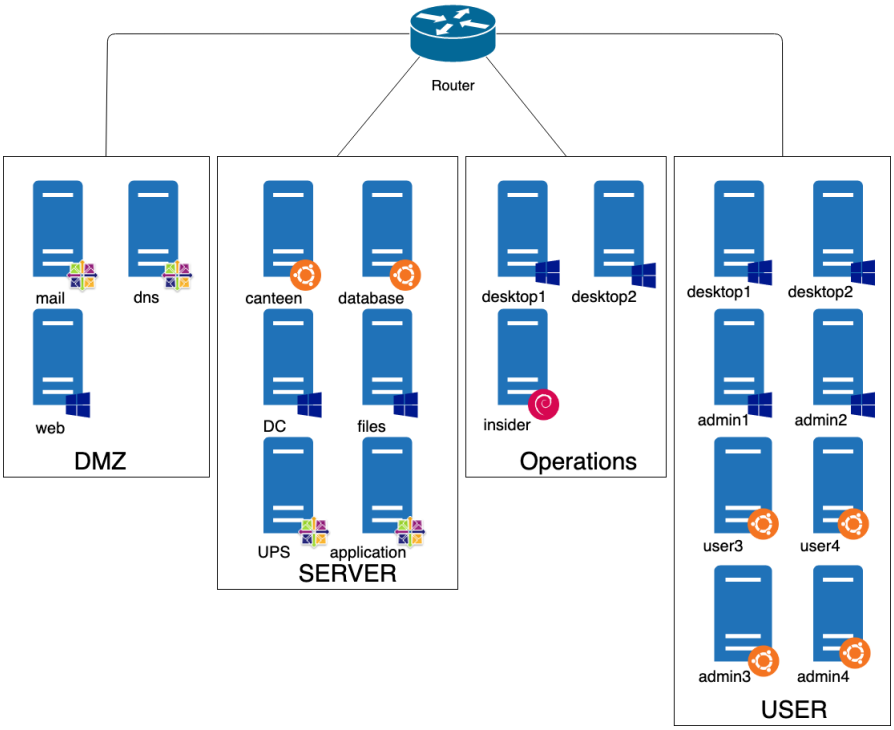
Cílem posledního měření bylo zjistit jak, počet systémů, které se zároveň testují, ovlivňuje dobu vyhledávání indikátorů kompromitace. Výsledky z tohoto měření jsou dostupné na obrázku 7.10. Je patrné, že s rostoucím počtem systémů se doba vyhledávání prodlužuje. Díky zavedení paralelizace však doba neroste lineárně, což vede ke značnému zlepšení používání aplikace. V tomhle měření narážíme i na limity hardwarové kapacity testovacího stroje, které mohou výsledky zkreslit. Proto zde není vhodné testovat větší množství systémů. Takové testování lze provést v produkční topologii a toto testování je popsáno v sekci 7.2.



Obrázek 7.10: Graf ukazující vliv počtu testovaných systémů na délku trvání aplikace.

7.2 Testování v infrastruktuře kybernetické cvičení

Původním plánem této práce bylo otestovat výslednou aplikaci také v produkčním prostředí s reálnými daty. Od tohoto plánu jsem však byl nucen ustoupit a to z následujících důvodů: Aby byly výsledky relevantní, bylo by za potřebí použít indikátory kompromitace získané z reálného bezpečnostního incidentu, ty však většinou podléhají utajení a nebylo by je možné v rámci této práce zveřejnit; pokud by aplikace některý z indikátorů kompromitace v síti našla, tato skutečnost by nemohla být zveřejněna; subjekty nejevily ochotu s jejich uveřejněním ve výsledné práci. Dobrou alternativou, která řeší všechny uvedené problémy je ale použití sítě modrého týmu z kybernetického cvičení. Vybrané kybernetické cvičení využívá formát tzv. red-blue team, kdy červený (red) team útočí na modrý (blue) team. Modrý tým má vybudovanou vlastní segmentovanou infrastrukturu s různými operačními servery a aplikacemi, přičemž v této infrastruktuře jsou od počátku cvičení nainstalované různé zranitelnosti, miskonfigurace a malware. Modrý tým má za úkol nejen zabezpečit celou infrastrukturu, ale také nalézt a opravit co nejvíce těchto anomálií. Topologie jednoho modrého týmu z toho cvičení lze vidět na obrázku 7.11. Tato infrastruktura odpovídá infrastruktuře menšího podniku, je segmentovaná a jsou v ní zastoupeny různé operační systémy (Windows 7, Windows 10, Windows 2012, Windows 2016, Ubuntu 16.04, CentOS 7, Kali Linux). Na strojích jsou nainstalované a nakonfigurované aplikace ve stejném duchu, jako by byly v produkčním prostředí. Největší výhodou využití tohoto prostředí je to, že na systémech jsou nainstalované zranitelnosti a miskonfigurace, o kterých předem víme, lze tudíž sestrojit seznam indikátorů kompromitace podle těch zranitelností, a tím ověřit, zda-li je nástroj dokáže všechny odhalit v relevantním čase.



Obrázek 7.11: Topologie používaná pro testování.

Pro testování bylo třeba vytvořit inventář (k vidění na obrázku 7.12), seznam indikátorů kompromitace (na obrázku 7.13) a konfiguraci (na obrázku 7.14). Stejně jako u lokálního

testování (popsaného v sekci 7.1), i zde byly cíle testování dva. Prvním bylo ověřit, že nástroj dokáže indikátory kompromitace správně na systému detekovat a druhým ověřit, že doba vyhledávání je únosná i pro různě velké množiny indikátorů kompromitace. Toto testování navíc na rozdíl od lokálního bylo prováděno na vzdálených strojích, které více odpovídají realitě. Linuxové stroje dostupné ve cvičení měly následující parametry: 4 virtuální procesory, 4 GB RAM a 50 GB HDD.

```
10.7.101.12 dmz-mail
10.7.101.13 dmz-dns
10.7.101.25 server-canteen
10.7.101.26 server-database
10.7.101.27 server-application
10.7.101.28 server-UPS
10.7.101.29 server-monitoring
10.7.101.34 operations-insider
10.7.101.44 user-desk3
10.7.101.45 user-desk4
10.7.101.48 user-adm3
10.7.101.49 user-adm4
```

Obrázek 7.12: Inventář používaný pro testování v infrastruktuře cvičení.

```
typ,hodnota,poznamka
path,/usr/local/apache2/htdocs/install.php,webshell
path,/var/www/html/img/kitteh.jpg,kitten webshell
path,/var/lib/libsnoopy.so,malicious libsnoopy library
path,/tmp/this-path-does-not-exist,nonexistent just for testing purposes
url,cdn.darkonia.ex,detected as command and control server
url,freeapps.darkonia.ex,serves malicious executables to users
url,this-domain-does-not-exist.com,nonexistent
ip,4.122.66.7,detected as command and control server
ip,4.201.35.140,known attackers ip address
ip,4.18.163.86,known attackers ip address
ip,4.12.6.15,known malicious server
ip,100.100.100.100,nonexistent
hash,da6d4e2395e67614c4b45bc52a1d5ac2,malware P0is0n st1ng from APT group
hash,47032d926bf21b5a56d9ffe8105ac756,known vulnerable sshd config distributed by APT group
hash,93bea26443ba0953b4a7973755458160,undocumented malware from APT group
hash,1729cc065b9cb0dd7f768b3d1ca85d96,malicious python binary distributed by APT group
hash,6857977a402addb77c9c24a3eb7bc3c4,md5 hash of malicious bash script downloaded on victims
hash,aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa,nonexistent
```

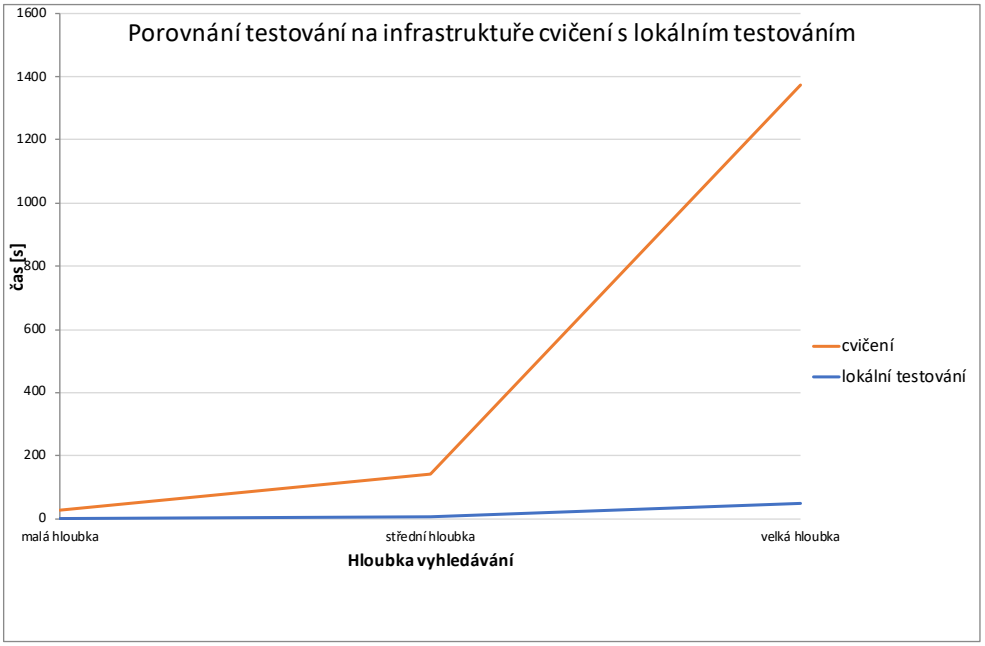
Obrázek 7.13: Indikátory kompromitace používané pro testování v infrastruktuře cvičení.

```
ssh_key=/Users/proj/DIP/src/id_rsa_dip
ssh_user=root
ssh_port=22
inventory_file=./inventory_cviceni
iocs_file=ioc.cviceni.in
selected_categories=all
verbosity=1
report_dir=report_cviceni
```

Obrázek 7.14: Konfigurační soubor používaný v infrastruktuře cvičení.

První test použil indikátory kompromitace z lokálního testování, konkrétně množinu, kde je od každého typu indikátoru deset unikátních položek, dohromady tedy celkem čtyřicet indikátorů. Testování bylo provedeno pro všechny možné hloubky. Výsledky měření

jsou k vidění na obrázku 7.15. Z grafu vyplývá, že doba testování se zvýšila téměř o 170 % pro malou hloubku, více než o 240 % pro střední hloubku a téměř o 280 % pro velkou hloubku testování. Tyto nárůsty jsou způsobeny následujícími faktory: pomalejší odezva a přenosové rychlosti SSH spojení; větší kapacita disků; použití plotnových disků. Naopak při tomto testování stroje měly větší proces procesorů a větší paměť.

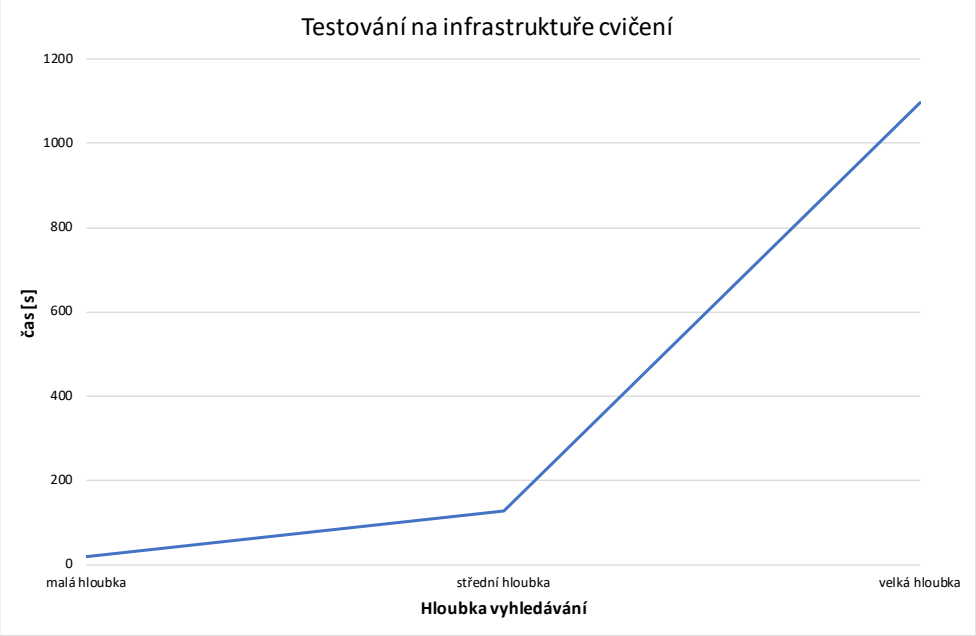


Obrázek 7.15: Porovnání délky vyhledávání v lokálním testovacím prostředí a v infrastruktuře cvičení.

Druhé testování se zaměřovalo pak na ověření funkcionality, tedy že aplikace najde všechny indikátory kompromitace popsané v souboru. Tento test byl pozitivní, výslednou závěrečnou zprávu vygenerovanou aplikací po spuštění aplikace s volbou prohledávání největší hloubky lze vidět na obrázku 7.16. Doba trvání testování je poté zobrazena na grafu, který je součástí obrázku 7.17.

```
date   [host]  (IoC type, IoC value)   message
2019-04-12 19:31:53 [dmz-mail] (path, /usr/local/apache2/htdocs/install.php) exists.
2019-04-12 19:31:53 [server-canteen] (path, /var/www/html/img/kitteh.jpg) exists.
2019-04-12 19:31:54 [user-adm3] (path, /var/lib/libsnoopy.so) exists.
2019-04-12 19:31:54 [user-adm4] (path, /var/lib/libsnoopy.so) exists.
2019-04-12 19:42:12 [server-UPS] (url, https://freeapps.darkonia.ex) found at /tmp/install.sh:# Downloaded from freeapps.darkonia.ex - Just run this script with sudo bash install.sh
2019-04-12 19:43:26 [server-application] (url, https://cdn.darkonia.ex) found at /var/www/html/app/js/app.js:<script src="http://cdn.darkonia.ex/jquery/v1.8.3/js/jquery-1.8.3.min.js"></script>
2019-04-12 19:44:41 [user-adm3] (ip, 4.201.35.140) found at /var/log/secure:May 10 14:41:19 localhost sshd[27953]: Accepted password for root from 4.201.35.140 port 44812 ssh2
2019-04-12 19:44:41 [user-adm3] (ip, 4.201.35.140) found at /var/log/secure:May 10 15:44:55 localhost sshd[28620]: Received disconnect from 4.201.35.140: 11: disconnected by user.
2019-04-12 19:45:08 [user-adm4] (ip, 4.201.35.140) found at /var/log/secure:May 10 10:02:21 localhost sshd[25430]: Accepted password for root from 10.7.99.24 port 55036 ssh2
2019-04-12 19:45:08 [user-adm4] (ip, 4.201.35.140) found at /var/log/secure:May 10 10:13:36 localhost sshd[25456]: Received disconnect from 4.201.35.140: 11: disconnected by user.
2019-04-12 19:45:38 [dmz-dns] (ip, 4.122.66.7) found at /root/.bash_history:ping 4.122.66.7
2019-04-12 19:45:38 [dmz-dns] (ip, 4.122.66.7) found in netstat output
2019-04-12 19:45:38 [dmz-dns] (ip, 4.122.66.7) found in ss output
2019-04-12 19:45:38 [user-desk3] (ip, 4.18.163.86) found at /var/log/secure:May 03 10:11:34 localhost sshd[1381]: Invalid user admin from 4.18.163.86 port 48855
2019-04-12 19:45:50 [server-monitoring] (ip, 4.12.6.15) found at /etc/rsyslog.conf:*.* @@4.12.6.15:514
2019-04-12 19:46:28 [server-application] (hash, 93bea26443ba0953b4a7973755458160) found at /var/www/html/.dont_touch_me
2019-04-12 19:46:32 [dmz-dns] (hash, 47032d926bf21b5a56d9ffe8105ac756) found at /etc/ssh/sshd.conf
2019-04-12 19:46:58 [server-canteen] (hash, 1729cc065b9cb0dd7f768b3d1ca85d96) found at /usr/bin/find
2019-04-12 19:47:15 [user-desk3] (hash, da6d4e2395e67614c4b45bc52a1d5ac2) found at /opt/firefox/libssl3.so
2019-04-12 19:47:23 [user-desk4] (hash, 6857977a402addb77c9c24a3eb7bc3c4) found at /tmp/.x
```

Obrázek 7.16: Ukázka závěrečné zprávy s nalezenými indikátory kompromitace na systémech.



Obrázek 7.17: Graf popisující délku vyhledávání vzhledem k hloubce vyhledávání.

Kapitola 8

Závěr

Cílem této práce bylo seznámit se s pojmem indikátor kompromitace, s různými kategoriemi těchto indikátorů a také s možnostmi, kde se na systému mohou vyskytovat. Dále seznámit se s již existujícími nástroji, které se touto problematikou zabývají a popsat jejich využití. Na základě těchto poznatků byl vybrán formát, ve kterém budou indikátory kompromitace předávány a byl také vytvořen návrh aplikace, která bude provádět skenování stanic na jejich přítomnost. Na základě těchto návrhů proběhl vývoj samotné aplikace. Návrh zápisu indikátorů kompromitace a konfiguračních souborů nenaznal po návrhu v průběhu implementace žádných změn. Pro formát zápisu inventáře byla přidána možnost specifikovat i SSH port. Na základě konzultací byla vyřazena kategorie YARA pravidel. Důvodem byl složitý zápis a malé možnosti využití oproti již vybraným kategoriím. Vycházelo se při tom z dosavadních zkušeností s indikátory kompromitace z reálných bezpečnostních incidentů. Pro vývoj samotné aplikace byl použit jazyk Python 3, jediná nestandardní použitá knihovna byla `paramiko`, která slouží k obsluze SSH spojení. Po ukončení vývoje proběhlo testování aplikace s testovacími daty v izolovaném virtuálním testovacím prostředí. Toto testování se zaměřovalo jak na ověření funkčnosti, tak na nalezení slabých míst aplikace a pomohlo nalézt několik optimalizací, čímž se výrazně zrychlil chod aplikace a vyhledávání indikátorů kompromitace na systémech. Poté proběhlo i testování v síti používané v rámci kybernetického cvičení, kde se opět zkoumala funkčnost aplikace. Ačkoliv byly výsledky řádově horší, než při lokálním testování, stále se s běžnými velikostmi sad indikátorů kompromitace dá síť oskenovat do maximálně pár hodin, což je pro tuto analýzu dostatečná doba.

Na základě testování vyplývá, že aplikace je vhodná do malých až středních sítí u větších sítí by už délka vyhledávání mohla přesáhnout únosnou mez. Velký vliv na rychlost testování má také výpočetní výkon systémů, velikost diskového úložiště a použitá technologie diskového úložiště. Pro použití, ke kterému však aplikace byla myšlena by však měla fungovat dostatečně dobře. Aplikace během testování neměla problémy s nalezením žádného indikátorů kompromitace, který se na cílovém systému měl nacházet a ani nevykazovala žádné známky hlášení false positive výsledků.

8.1 Možná rozšíření

Aplikaci by šlo rozšířit o další kategorie indikátorů kompromitace. Například prohledávání procesů nebo prohledávání paměti. Také by se dala rozšířit o více hashovacích algoritmů, jako jsou například algoritmy z rodiny SHA nebo i jiné algoritmy z rodiny MD. V případě

formátu inventáře, možnosti rozšíření jsou například přidání schopnosti definování SSH uživatele, hesla a klíče pro každý stroj. Co se týče konfigurace, ta by šla rozšířit o možnost specifikovat konkrétní cesty vyhledávání nebo například omezení maximálního počtu konkurentních SSH spojení.

Literatura

[1] *CAPEC Homepage*. [Online; navštíveno 12.12.2018].
URL https://capec.mitre.org/

[2] *Dokumentace nástroje GRR*. [Online; navštíveno 19.11.2018].
URL http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview

[3] *Introduction to TAXII*. [Online; navštíveno 12.12.2018].
URL https://oasis-open.github.io/cti-documentation/taxii/intro

[4] *MAEC 5.0*. [Online; navštíveno 12.12.2018].
URL https://maecproject.github.io/releases/5.0/

[5] *OpenIOC 1.1*. [Online; navštíveno 13.12.2018].
URL https://github.com/mandiant/OpenIOC_1.1

[6] *Oval Homepage*. [Online; navštíveno 13.12.2018].
URL https://oval.mitre.org

[7] *Paramiko Project Information*.
URL https://www.paramiko.org/

[8] *Security Content Automation Protocol*. [Online; navštíveno 13.12.2018].
URL
https://csrc.nist.gov/projects/security-content-automation-protocol/

[9] *SleuthKit's About Page*. [Online; navštíveno 20.11.2018].
URL https://www.sleuthkit.org/about.php

[10] *Understanding and Detecting Malware Threats Based on File Sizes*. [Online; navštíveno 1.3.2019].
URL https://d3gpjj9d20n0p3.cloudfront.net/fortiguard/research/DetectingMalwareThreats.pdf

[11] *What is STIX?* [Online; navštíveno 12.12.2018].
URL https://oasis-open.github.io/cti-documentation/stix/intro

[12] *YARA Homepage*. [Online; navštíveno 10.11.2018].
URL https://virustotal.github.io/yara/

[13] *YARA's documentation*. [Online; navštíveno 10.11.2018].
URL https://yara.readthedocs.io/en/v3.8.1/

[14] Lord, N.: *What are Indicators of Compromise?* [Online; navštíveno 29.10.2018]. URL https://digitalguardian.com/blog/what-are-indicators-compromise

[15] Marcus, S.: *RSA NetWitness Hunting Guide.* [Online; navštíveno 27.10.2018]. URL https://community.rsa.com/docs/DOC-79618

[16] Menezes, A.; Oorschot, P. V.; Vanstone, S.: *Handbook of Applied Cryptography.* CRC Press, 1997, ISBN 0-8493-8523-7.

[17] Stallings, W.: *Cryptography and Network Security. 4th Edition.* Pearson, 2005, ISBN 0-13-187316-4.

Příloha A

Obsah CD

Součástí práce je přiložené CD, které obsahuje všechny dokumenty týkající se této práce. Celý text práce je uložen v hlavním adresáři pod názvem thesis.pdf, přiloženy jsou i zdrojové soubory textové části práce i aplikace, která je výsledkem práce. Adresářová struktura přiloženého CD je následující:

- **thesis** zdrojové soubory textové části práce
- **source** zdrojové soubory aplikace
- **thesis.pdf**

Příloha B

Ukázky existujících formátů pro ukládání indikátorů kompromitace

```
{
  "type": "bundle",
  "id": "bundle--44af6c39-c09b-49c5-9de2-394224b04982",
  "spec_version": "2.0",
  "objects": [
    {
      "type": "indicator",
      "id": "indicator--d81f86b9-975b-bc0b-775e-810c5ad45a4f",
      "created": "2014-06-29T13:49:37.079Z",
      "modified": "2014-06-29T13:49:37.079Z",
      "labels": [
        "malicious-activity"
      ],
      "name": "Malicious site hosting downloader",
      "pattern": "[url:value = 'http://x4z9arb.cn/4712/']",
      "valid_from": "2014-06-29T13:49:37.079000Z"
    },
    {
      "type": "malware",
      "id": "malware--162d917e-766f-4611-b5d6-652791454fca",
      "created": "2014-06-30T09:15:17.182Z",
      "modified": "2014-06-30T09:15:17.182Z",
      "name": "x4z9arb backdoor",
      "labels": [
        "backdoor",
        "remote-access-trojan"
      ],
      "description": "This malware attempts to download remote files after
          establishing a foothold as a backdoor.",
      "kill_chain_phases": [
        {
          "kill_chain_name": "mandiant-attack-lifecycle-model",
```

```
        "phase_name": "establish-foothold"
      }
    ]
  },
  {
    "type": "relationship",
    "id": "relationship--6ce78886-1027-4800-9301-40c274fd472f",
    "created": "2014-06-30T09:15:17.182Z",
    "modified": "2014-06-30T09:15:17.182Z",
    "relationship_type": "indicates",
    "source_ref": "indicator--d81f86b9-975b-bc0b-775e-810c5ad45a4f",
    "target_ref": "malware--162d917e-766f-4611-b5d6-652791454fca"
  }
]
}
```

Výpis B.1: Ukázka zápisu indikátoru kompromitace ve formátu STIX v jazyce Python, převzato z https://oasis-open.github.io/cti-documentation/examples/indicator-for-malicious-url

```
import stix2

indicator = stix2.Indicator(
    id="indicator--d81f86b9-975b-bc0b-775e-810c5ad45a4f",
    created="2014-06-29T13:49:37.079Z",
    modified="2014-06-29T13:49:37.079Z",
    name="Malicious site hosting downloader",
    description="This organized threat actor group operates to create
        profit from all types of crime.",
    labels=["malicious-activity"],
    pattern="[url:value = 'http://x4z9arb.cn/4712/']",
    valid_from="2014-06-29T13:49:37.079000Z"
)

foothold = stix2.KillChainPhase(
    kill_chain_name="mandiant-attack-lifecycle-model",
    phase_name="establish-foothold"
)

malware = stix2.Malware(
    id="malware--162d917e-766f-4611-b5d6-652791454fca",
    created="2014-06-30T09:15:17.182Z",
    modified="2014-06-30T09:15:17.182Z",
    name="x4z9arb backdoor",
    labels=["backdoor", "remote-access-trojan"],
    description="This malware attempts to download remote files after
        establishing a foothold as a backdoor.",
    kill_chain_phases=[foothold]
)
```

```
28
29 relationship = stix2.Relationship(indicator, 'indicates', malware)
30
31 bundle = stix2.Bundle(objects=[indicator, malware, relationship])
```

Výpis B.2: Ukázka zápisu indikátoru kompromitace ve formátu STIX v jazyce Python, převzato z https://oasis-open.github.io/cti-documentation/examples/indicator-for-malicious-url

```
1  <?xml version="1.0" encoding="us-ascii"?>
2  <ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http
       ://www.w3.org/2001/XMLSchema" id="fc2d3e44-80a6-4add-ad94-de9f289e62ff"
        last-modified="2011-10-28T21:00:13" xmlns="http://schemas.mandiant.com
       /2010/ioc">
3    <short_description>CCAPP.EXE</short_description>
4    <description>Custom Reverse shell.</description>
5    <keywords />
6    <authored_by>Mandiant</authored_by>
7    <authored_date>2010-12-13T12:49:53</authored_date>
8    <links>
9      <link rel="grade">Alpha</link>
10   </links>
11   <definition>
12     <Indicator id="d610019c-379f-4d3e-b299-f0b0e5c3313a">
13       <IndicatorItem id="86d0e6fc-ff41-4743-8a9a-c323ef8ad8cb" condition="
             is">
14         <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
15         <Content type="md5">9855c23be2b6f38630756a277b52cdd2</Content>
16       </IndicatorItem>
17     </Indicator>
18     <Indicator id="3902a731-260b-49e8-84a5-77d2a420716e">
19       <IndicatorItem id="034d5703-bc58-4a09-9333-e24cf4c41fe9" condition="
             contains">
20         <Context document="FileItem" search="FileItem/PEInfo/Sections/
               Section/Name" type="mir" />
21         <Content type="string">.vmp0</Content>
22       </IndicatorItem>
23     </Indicator>
24   </definition>
25 </ioc>
```

Výpis B.3: Ukázka zápisu indikátoru kompromitace ve formátu OpenIOC v jazyce XML (převzato z https://github.com/STIXProject/openioc-to-stix/blob/master/examples/ccapp.ioc.xml, upraveno)