Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nástroj na detekciu indikátorov kompromitácie v systémoch Windows

Bakalárska práca

2016
Ľuboš Miklošovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nástroj na detekciu indikátorov kompromitácie v systémoch Windows

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Jaroslav Janáček PhD.

Bratislava, 2016
Ľuboš Miklošovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Ľuboš Miklošovič
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Nástroj na detekciu indikátorov kompromitácie v systémoch Windows
Tool for Detection of Indicators of Compromise in Windows Systems

Cieľ: Cieľom práce je implementovať nástroj na detekciu indikátorov kompromitácie pre OS Windows. Vytvorená aplikácia musí byť schopná zistiť prítomnosť indikátorov kompromitácie ako výskyt súboru s určeným menom a hashovacou hodnotou obsahu, prítomnosť určeného procesu, mutexu, komunikačného objektu, certifikátu, DNS záznamu v cache, kľúča v registroch alebo ich logickej kombinácie. Aplikácia musí byť spustiteľná na operačných systémoch Windows XP až Windows 10 a Windows Server 2003 až 2012 R2. Aplikácia nesmie vyžadovať inštaláciu žiadnych dodatočných komponentov do systému, aby bola spustiteľná na veľkom množstve počítačov v heterogénnom prostredí. Aplikácia musí byť schopná definíciu indikátorov načítavať voliteľne zo súboru alebo zo servera a výsledky kontroly zapísať tiež voliteľne do súboru a/alebo odoslať na server.

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 28.10.2015

Dátum schválenia: 28.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím uvedených zdrojov.

V Bratislave 2016 .....

Poďakovanie: Chcel by som sa poďakovať môjmu vedúcemu RNDr. Jaroslav Janáček PhD. za trpezlivosť a ústretovosť, pomoc pri riešení problémov a za dohľad nad mojou činnosťou.

Abstrakt

Cieľom práce je navrhnúť a implementovať nástroj na detekciu indikátorov kompromitácie na OS Windows. Vytvorená aplikácia bude schopná detegovať prítomnosť indikátorov ako výskyt súboru s určeným menom alebo hashovacou hodnotou obsahu, prítomnosť procesu, mutexu, komunikačného objektu, certifikátu, DNS záznamu, kľúča v registroch alebo ich logickej kombinácie. Aplikácia bude spustiteľná na systémoch Windows XP SP2 až Windows 10 a Windows server 2003 SP1 až 2012 R2. Aplikácia nebude vyžadovať inštaláciu dodatočných komponentov. Aplikácia bude schopná načítať definície indikátorov voliteľne zo súboru alebo zo servera a výsledky kontroly zapísať tiež voliteľne do súboru alebo odoslať na server.

Kľúčové slová: indikátor kompromitácie, bezpečnosť, OS Windows, detekcia, škodlivý softvér

Abstract

The purpose of thesis is to design and implement tool for detection of indicators of compromise on OS Windows. Application will be able to detect presence of indicators by presence of files with specified name or hash value, presence of process, mutex, communication object, certificate, DNS entry, registry key or their logical combination. Application will be executable on operating system Windows XP SP2 to Windows 10 and Windows server 2003 SP1 to 2012 R2. Application will not require installation of any additional components. Application will be able to read definition of indicators optionally from file or server and write results also optionally into file or send to server.

Keywords: indicator of compromise, security, OS Windows detection, harmful software

Obsah

Kapitola 1

Úvod

V dnešnej dobe je počítačová bezpečnosť jedna z najviac diskutovaných tém v informatike. Firmy sa snažia, aby boli ich dôležité informácie chránené všetkými dostupnými prostriedkami, napríklad antivírusmi alebo firewallmi. Tieto programy buď pomocou preddefinovaných znakov, alebo pomocou analýzy správania odhaľujú škodlivý softvér. Dokážu odhaliť ale len softvér, ktorý je už známy a sú zraniteľné takzvanými zero-day útokmi.

Vo firemnom alebo štátnom prostredí je pravdepodobnosť cieleného útoku vysoká a spoliehať sa len na bežné ochranné prvky nestačí. Ak sa škodlivému softvéru podarí preniknúť do systému, môže tieto prvky vypnúť, bez toho aby sme to zistili. Niektoré organizácie sa preto špecializujú na analýzu škodlivého softvéru, zbieranie dát a zabezpečovanie ochrany. Táto ochrana je buď vo forme analýzy bezpečnostných rizík spoločnosti alebo skenovaním počítačov. Príkladom je organizácia Computer Security Response Team Slovakia, ktorú si zriaďuje štát a pre ktorú je táto aplikácia určená.

1.1 Motivácia

Existuje množstvo nástrojov, ktoré dokážu pomocou indikátorov kompromitácie odhaliť škodlivý softvér na počítači. Problémom je, že spoločnosti, ktoré ich vytvorili, ich neponúkajú zadarmo. Aplikácie si musíme buď kúpiť, alebo si zaplatiť prístup k indikátorom. Tieto poplatky sú obvykle pre malé firmy alebo štátny sektor príliš vysoké na pravidelnú kontrolu. Naším cieľom je vytvoriť open source aplikáciu so základnou funkcionalitou, ktorá sa bude dať ľahko rozšíriť, ponúknuť tak bezplatnú bezpečnostnú ochranu pre tieto oblasti a zlepšiť tak počítačovú bezpečnosť v týchto oblastiach.

1.2 Členenie práce

V druhej kapitole si uvedieme základné pojmy, definujeme si pojem indikátora kompromitácie, povieme si, ako sa vytvárajú, spomenieme si niektoré používané formáty indikátorov, pozrieme sa, kde môžeme získať prístup k indikátorom a povieme si o Windows API. V tretej kapitole si špecifikujeme funkcionalitu aplikácie. V štvrtej kapitole sa pozrieme na formát indikátorov, ktorý sme sa rozhodli použiť, pozrieme sa na to, ako je aplikácia implementovaná, na aké problémy sme počas implementácie narazili a ako sme sa s nimi vysporiadali a oboznámime sa aj s externými knižnicami, ktoré sme použili.

Kapitola 2

Úvod do problematiky

V tejto kapitole si uvedieme základné informácie o indikátoroch kompromitácie, oboznámime sa s postupom pri hľadaní nového škodlivého softvéru a ako k nemu môžeme vytvoriť indikátory. Ďalej sa pozrieme na niektoré známe schémy, ktoré sa na popis indikátorov používajú, uvedieme si zdroje, z ktorých sa dajú indikátory získať a povieme si o Windows API, ktoré je dôležité pre prácu so systémovými objektami.

2.1 Indikátory kompromitácie (IOC)

Indikátory kompromitácie sú forenzné artefakty alebo ich zoskupenia, ktoré sa viažu k určitému škodlivému softvéru. Pri zostavovaní indikátora pre konkrétny malvér je naším cieľom čo najpresnejšie popísať jeho správanie a vlastnosti, aby sme pri ďalšom zaznamenaní takého správania vedeli s určitosťou povedať, či sa jedná o daný malvér. Indikátory kompromitácie sa používajú na popísanie pozorovateľných objektov.

Takýmto objektom môže byť udalosť, ktorými sa dajú popísať taktiky, techniky a procedúry (TTP) škodlivého softvéru alebo to môžu byť vlastnosti objektov.

- udalosti - vytvorenie súboru na disku, pridanie kľúča do registrov, spustenie procesu
- vlastnosti objektov - hash hodnota súboru, meno procesu, kľúč v registroch a jeho hodnota

Indikátory môžeme následne spájať do zložitejších štruktúr napríklad stromov, v ktorých listy sú indikátory a vo vnútorných vrcholoch sú logické spojky AND a OR, ktorými sa dá popísať správanie škodlivého softvéru.

2.1.1 Vytváranie indikátorov kompromitácie

V tejto práci sa zameriavame na odhaľovanie indikátorov a nie na ich vytváranie, preto spomenieme len základné princípy ich vytvárania. Viac informácií sa môže čitateľ nájsť v odborných článkoch [9], [10].

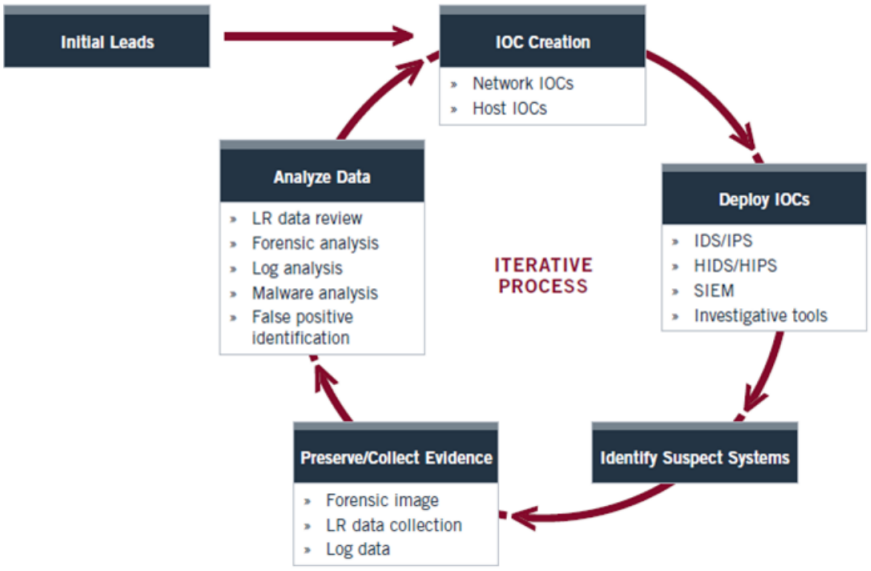
Každý deň sa objaví nový škodlivý softvér. V prípade, že nesie znaky už zdokumentovaných programov, ktoré boli označené ako škodlivé, je zachytený. Ak však ide o nový druh, je potrebné k nemu vytvoriť indikátory, aby sme ho v budúcnosti spoznali.

Z praktických aj bezpečnostných ohľadov musí analýza malvéru prebiehať v izolovanom prostredí bez prístupu k akýmkoľvek externým prostriedkom. Externé zdroje, ako napríklad prístup na internet, sa simulujú pomocou hardvéru alebo softvéru. Izolácia zabezpečí bezpečnosť testu, nehrozí, že by sa malvér neúmyselne rozšíril a takisto sa ľahko deteguje každá zmena, ktorú malvér spôsobí. Analýzu delíme na statickú a dynamickú.

- Statická analýza - počas tohto procesu sa zo súboru extrahujú všetky dostupné informácie, bez toho aby sme ho spustili. Tieto informácie zahŕňajú hashovacie hodnoty súboru. Zistí sa, ktoré knižnice používa a vyrobia sa antivírusové signatúry. Zistí sa o aký druh súboru ide(spustiteľný, DLL, dokument, a pod.) [9].

- Dynamická analýza - v tejto časti sa program spustí a sleduje sa jeho interakcia so systémom a sieťová komunikácia. Hlavné oblasti skúmania sú interakcia s procesmi, súbormi, registrami, API volania a sieťová aktivita. Častým cieľom malvéru sú registere, lebo sa z nich dajú získať informácie o nainštalovaných programoch. Porovnaním registrov pred a po spustení malvéru môžeme získať informácie o jeho aktivite. Sieťová komunikácia je takisto monitorovaná, pre prípad, že sa softvér pokúsi odoslať dáta útočníkovi. Takto sa dajú získať adresy, na ktoré sa pripája [9].

Následne sa z informácii získaných z analýzy zostaví indikátor, ktorý sa rozdistribuuje medzi klientov, od ktorých sa získa spätná väzba, pomocou ktorej sa indikátor vylepší a celý proces sa opakuje (viď obr. 2.1).



Obr. 2.1: Životný cyklus indikátorov kompromitácie [1]

2.2 Aktuálne riešenia

2.2.1 Formáty

Keď máme zostavené dáta popisujúce správanie malvéru, potrebujeme ich previesť do formátu, ktorý je vhodný pre počítač. V dnešnej dobe existuje množstvo formátov, ktoré sa používajú, no nedá sa povedať, že by existoval ideálny alebo všeobecne zaužívaný formát. My sa zameriame len na niektoré a to konkrétne na OpenIOC [1], [5], CybOX [4] a IOCDEF [6] a ich výhody a nevýhody [7].

Formát CybOX je produktom spoločnosti MITRE. Obsahuje indikátory na popis udalostí aj vlastností objektov. Výhodou je kompatibilita s ďalšími schémami, čo poskytuje možnosť vytvárania komplexnej ochrany. Nevýhodou je, že sa schéma komerčne príliš neuplatnila a že potrebuje ďalšie schémy na popísanie taktík, techník a procedúr(TTP).

Produktom spoločnosti Mandiant je formát OpenIOC. Výhodami sú ľahká rozšíriteľnosť, zadarmo dostupné nástroje na vytváranie indikátorov, plná podpora v produktoch firmy a publikovanie pod Apache 2 licenciou. Nevýhodou je, že schéma je značne orientovaná na produkty firmy a nedajú sa v nej popísať TTP.

Incident Object Description and Exchange Format (IODEF) je otvorený štandard vytvorený organizáciou IETF. Jeho výhodami sú nezávislosť od predajcov, používa sa v praxi a je dobre zdokumentovaný. Nevýhodou je, že bol primárne vyvinutý na zdieľanie dát z incidentov medzi spoločnosťami a nie ako formát na popisovanie indikátorov kompromitácie.

Ďalšími formátmi sú CAPEC, MAEC, YARA, OVAL, IDMEF, Comma Separated Values (CSV), a pod.

2.2.2 Softvér

Mandiant ponúka zadarmo, ale nie open source, IOC Editor, spustiteľný len na systémoch Windows, na vytváranie indikátorov kompromitácie a ponúka tiež Redline, takisto zadarmo, ale nie open source, ktorý slúži na hľadanie škodlivého softvéru.
IOC-EDT je open source webová aplikácia na vytváranie OpenIOC.

2.3 Zdroje indikátorov kompromitácie

Indikátory sa dajú získať z rôznych zdrojov. Bezpečnostné spoločnosti ako RSA, Mandiant, a pod. poskytujú prístup k svojim databázam, ale iba klientom, ktorí za to platia. Alternatívnym zdrojom môže byť štátna organizácia CSIRT, ale tieto organizácie neposkytujú dáta každému a je možné, že vás odmietnu. Treťou možnosťou je skúsiť získať voľne dostupné indikátory. Mnohé organizácie vydávajú správy, v ktorých sa indikátory dajú priamo nájsť alebo obsahujú dosť dát na vytvorenie. Inou možnosťou je skúsiť zadať do internetového vyhľadávača čo hľadáme a skúsiť šťastie. Poslednou alternatívou je IOC Bucket. Ide o internetovú platformu pre bezpečnostnú komunitu na zdieľanie indikátorov kompromitácie.

2.4 Windows API

Na získanie dát zo systému, ako sú zoznam mutexov, ktoré proces drží alebo na hľadanie certifikátov, potrebujeme pristúpiť k dátam, ktoré sú držané operačným systémom a z dôvodu ich bezpečnosti a integrity sa k nim nevieme dostať inak ako volaním špeciálnych funkcií systému Windows, ktoré sa skupinovo volajú Windows API (WinAPI). Tieto funkcie sa delia na dva druhy dokumentované a nedokumentované. Zatiaľ čo dokumentované funkcie garantujú, že sa ich správanie medzi vydaniami nemení, napríklad RegOpenKeyEx, ktorá slúži na prístup ku kľúčom v registroch, nedokumentované funkcie sa môžu medzi vydaniami meniť a v niektorých prípadoch vyžadujú namáhavé hľadanie, pretože sa nenachádzajú v oficiálnej dokumentácii. Môžeme ich nájsť v článkoch a knihách, ktoré boli vytvorené komunitou.
Informácie o funkciách sa dajú získať z oficiálnych stránok Microsoftu www.msdn.com, ale varujeme čitateľa, že nie všetky informácie uvedené na stránkach sú korektné, treba ich brať s rezervou a radšej otestovať na malom príklade než sa pustíte do implementácie.

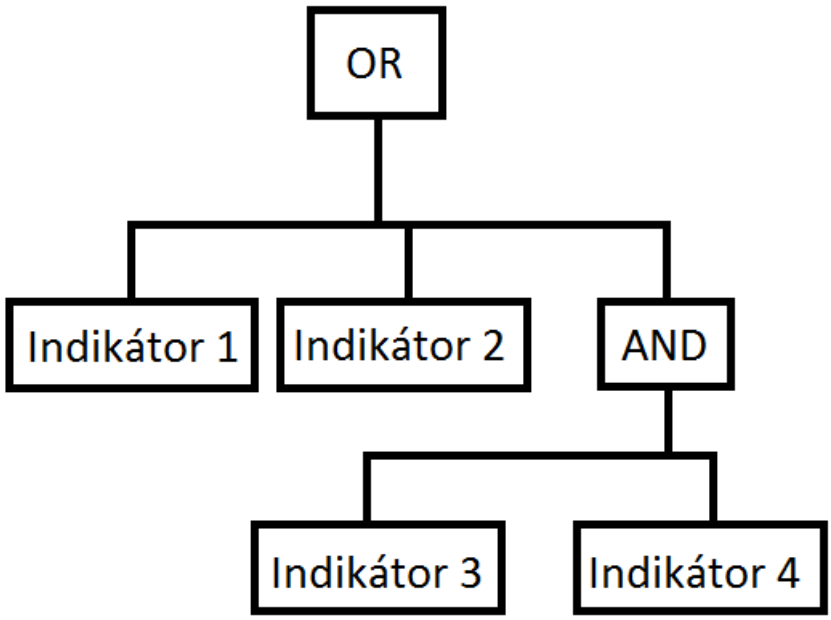
Kapitola 3

Funkcionalita aplikácie

V tejto kapitole špecifikujeme základnú funkcionalitu, ktorá bude v aplikácii implementovaná.

3.1 Reprezentácia dát

Dáta budú reprezentované, ako stromová štruktúra, kde vrchol stromu je buď AND alebo OR, ak sa jedná o vnútorný vrchol alebo identifikátor, ak je vrchol list (viď obr. 3.1).



Obr. 3.1: Abstraktná predstava testovacej sady

3.2 Detekcia malvéru

Existuje množstvo typov indikátorov kompromitácie, ktorými sa dajú popísať udalosti a vlastnosti. Môže to byť napríklad názov súboru, hodnota kľúča v registroch, ale tiež aj čas posledného prihlásenia. V tejto práci sa snažíme obmedziť na tie, ktoré sú najčastejšie a zabezpečiť, aby bol kód programu prehľadný a ľahko rozšíriteľný o ďalšiu funkcionalitu.
Najbežnejším pozostatkom po napadnutí sú **súbory**. Ak sa jedná o menej sofistikovaný softvér, ktorého cieľom nie je utajenie, je bežné, že na disku môžeme nájsť, buď priamo spustiteľné súbory daného softvéru alebo súbory, ktoré používa pri svojom behu. Aplikácia je schopná detegovať prítomnosť súboru na základe jeho názvu, regulárneho výrazu a hashovej hodnoty (MD5, SHA2-256, SHA1).

Ďalším častým objektom, ktorý môžeme objaviť sú bežiace **procesy**. Aplikácia dokáže získať zoznam bežiacich procesov a vyhľadávať v nich podľa mena a regulárneho výrazu. Takisto je schopná získať hash hodnotu(MD5, SHA2-256, SHA1) spustiteľného súboru procesu a porovnať ju z hľadanou hodnotou.

Znakom napadnutia môže byť aj **prítomnosť objektu vzájomného vylúčenia(mutant)**. Tieto môže škodlivý softvér používať napríklad na synchronizáciu alebo ako indikátor, že daný systém je už napadnutý. Aplikácia deteguje takéto objekty podľa ich mena.

Podobne ako pri mutantoch môžu aplikácie použiť na zdieľanie údajov **Windows Registry**. Aplikácia dokáže odhaliť prítomnosť kľúča v registroch podľa troch parametrov cesta, meno hodnoty, hodnota. S tým, že ľubovoľný môže byť vynechaný.

Škodlivý softvér, ktorého cieľom je získať informácie od obete, sa dá odhaliť kontrolovaním **sieťovej komunikácie**. Aplikácia je schopná detegovať neželanú sieťovú komunikáciu na základe IP adresy a doménového mena alebo jeho regulárneho výrazu. Takisto dokáže detegovať **záznamy v DNS cache** na základe názvu.

Posledným indikátorom, na ktorý sa zameriame, je prítomnosť **certifikátu**. Tieto detegujeme podľa mena certifikačnej autority alebo doménového mena asociovaného s certifikátom.

3.3 Komunikácia a zaznamenávanie výsledkov

Aplikácia funguje v dvoch módoch, samostatnom a sieťovom. V samostatnom móde sa definície IOC načítajú z lokálneho súboru a výsledky sa uložia na lokálny súbor. V sieťovom móde sa pripojí na server pomocou protokolu TLS a definície (výsledky) sa stiahnu (nahrajú) na server.

3.4 Kompatibilita

Aplikácia bude prenosná, nebude nevyžadovať inštaláciu a bude podporovať verzie MS Windows XP SP2 až po Windows 10 a Windows Server 2003 SP1 až po 2012 R2.

Kapitola 4

Implementácia

V tejto kapitole sa budeme venovať implementácii aplikácie. Pozrieme sa na formát vstupu a výstupu. Podrobne popíšeme celky, z ktorých sa aplikácia skladá a na koniec sa oboznámime s externými knižnicami, ktoré používa.

4.1 Formát vstupných dát a výstupných dát

Vstupné dáta dostáva aplikácia vo forme Json-u. Json pozostáva z hodnoty *success*, ktorá indikuje, či sú dáta indikátor alebo nie. A poľa hodnôt *data*. Ak, v prípade získavania dát zo servera, testovacia sada neexistuje alebo je problém z jej získaním, odošle server json, v ktorom je hodnota *success* nastavená na false a v hodnote *data* sa nachádza chybová správa. V prípade, že všetko prebehne v poriadku, je hodnota *success* nastavená na true a *data* obsahuje zoznam indikátorov testovacej sady. Indikátory sú reprezentované hodnotami *Id, name, type* a poľom hodnôt *value*. Zatiaľ čo *name* je hodnota len informačná a slúži skôr pre informovanie čitateľa logu, hodnota *Id* presne definuje indikátor v sade a hodnotu *type* používa aplikácia na zistenie typu indikátora. V poli hodnôt *value* je presne popísané, akú hodnotu má aplikácia hľadať. Špeciálnou požiadavkou pri vývoji aplikácie bolo, aby bolo možné spájať indikátory pomocou logických spojok *and* a *or*. Toto je riešené pomocou špeciálnych indikátorov, ktoré majú namiesto hodnoty *data* hodnotu *children*, ktorá obsahuje zoznam indikátorov.

Indikátory 4.1: OR a AND

```
{
    "type": "or",
    "children": [
        {indikator 1},
        {indikator 2}
    ]
}

{
    "type": "and",
    "children": [
        {indikator 1},
        {indikator 2}
    ]
}
```

Výstupným formátom je takisto Json. Obsahuje hodnoty *dev*, *timestamp* a *org*, ktorými môžeme určiť, z ktorého počítača v danej organizácii dáta pochádzajú a kedy boli získané. Ďalej obsahuje pole *results*, v ktorom je zoznam indikátorov. Ku každému indikátoru je priradená hodnota *result* nastavená na *true* alebo *false* podľa toho, či bol nájdený a ak bol, informácie, s pomocou ktorých ho môžeme znova vypátrať. Poslednou zložkou je pole *FailedToOpen*, ktoré obsahuje zoznam certifikačných úložísk a kľúčov v registroch, ktoré aplikácia počas behu nedokázala otvoriť.

4.2 Aplikácia

Aplikácia je rozdelená na hlavný program a pomocné moduly. Úlohou hlavného programu je načítať konfiguračný súbor, v ktorom je špecifikované meno inštitúcie, to či sa má aplikácia spustiť v sieťovom alebo lokálnom móde, ip adresa servera, meno testovacieho súboru a či sa majú prehľadávať otvorené IPv6 spojenia. Dôvodom pre vypnutie IPv6 spojení je, že v prípade, keď nie je nainštalovaný IPv6 protokol môže dôjsť k nedefinovanému správaniu, čomu sa chceme vyhnúť. Ak je aplikácia spustená v sieťovom móde, tak sa po načítaní konfiguračného súboru pripojí na server a stiahne definície indikátorov do lokálneho adresára. Následne sa definície spracujú a zavolajú sa príslušné moduly na prehľadanie systému. Každý typ indikátora ma vlastnú prioritu, ktorá určuje poradie, v ktorom sa vyhodnocujú. Dôvodom pre priority je, že ak by sme chceli v budúcnosti implementovať len zisťovanie, či je systém napadnutý, stačí nám prehľadávať len dovtedy, kým to nevieme s určitosťou potvrdiť alebo vyvrátiť. V

tomto prípade je pre nás výhodnejšie uprednostniť časovo menej náročné úlohy, ako je napríklad zistenie prítomnosti bežiaceho procesu, pred úlohami ako napríklad prehľadávanie registrov alebo súborového systému. Po dokončení hľadania hlavný program vytvorí z informácii, ktoré dostane z modulov výstupný súbor. Ak je aplikácia spustená v sieťovom režime, tak sa súbor odošle na server.

Teraz sa pozrieme na funkcie pomocných modulov a to ako sú implementované.

4.2.1 Kontrola certifikátov

Prvým programom, ktorý si priblížime je modul na kontrolu prítomnosti nežiadaných certifikátov. Existuje množstvo vlastností, na ktoré by sme mohli certifikáty testovať. My sa zameriame na certifikačnú autoritu, ktorá certifikát podpísala a na domény, ktoré sú v certifikáte zapísané.

Podľa požiadaviek v špecifikácii sme vytvorili dva druhy indikátorov. Prvým typom, *"type": "cert-ca"*, sa kontroluje meno certifikačnej autority a druhým, *"type": "cert-dom"*, sa kontrolujú domény, ku ktorým je certifikát vystavený. Obidva indikátory obsahujú jednu hodnotu, ktorá určuje hľadaný reťazec.

Indikátory 4.2: certifikáty

```
{
    "id": 1,
    "name": "uzivatelov nazov",
    "type": "cert-dom",
    "value": [
        "nazov domeny"
    ]
},
{
    "id": 2,
    "name": "sample name",
    "type": "cert-ca",
    "value": [
        "nazov certifikacnej autority"
    ]
}
```

Pozrime sa najprv na to, ako je celá štruktúra certifikátov uložená. My sa zameriame len na základné pojmy, viac informácií môžeme nájsť na stránkach Microsoftu [11]. V systéme je niekoľko lokácií certifikačných úložísk. Tieto obvykle obsahujú *Collection Store*. Ide vlastne o spájaný zoznam certifikačných úložísk, ktoré sa v danej lokácii

nachádzajú. Následne každé certifikačné úložisko obsahuje spájaný zoznam certifikátov.

Prvým krokom je získať zoznam lokácií certifikačných úložísk. Tento zoznam získame pomocou funkcie *CertEnumSystemStoreLocation*. Táto berie ako parametere programátorom definovanú štruktúru *ENUM_ARG*, v ktorej si budeme podávať hľadané certifikáty a callback funkciu, ktorá je definovaná podľa prototypu *PFN_CERT_ENUM_SYSTEM_STORE_LOCATION*, ktorú zavolá pre každú lokáciu. Naša callback funkcia je vcelku jednoduchá. Zo vstupného parametra dwFlags, v ktorom sú uložené informácie o úložisku získame jeho lokáciu a zavoláme funkciu *CertEnumSystemStore*. Rovnako ako funkcia pre zoznam lokácií, berie ako vstup callback funkciu definovanú podľa prototypu *PFN_CERT_ENUM_SYSTEM_STORE* a *ENUM_ARG*. V tejto chvíli máme k dispozícii všetky potrebné údaje na získanie certifikátov. Z dátovej štruktúry *ENUM_ARG*, ktorú sme si podávali v callback funkciách získame zoznam hľadaných certifikátov a pomocou funkcií *CertOpenStore* a *CertEnumCertificatesInStore* otvoríme úložisko a postupne získame všetky certifikáty. Pre každý certifikát získame pomocou funkcie *CertNameToStrW* meno vo formáte X.500. Pokiaľ certifikát obsahuje alternatívne meno, je pomocou funkcie *CryptDecodeObjectEx* otvorený. Výstupná štruktúra *CERT_ALT_NAME_INFO* obsahuje zoznam alternatívnych mien certifikátu. Každé alternatívne meno, ktoré má nastavenú hodnotu *dwAltNameChoice* na *CERT_ALT_NAME_DNS_NAME*, je porovnané so zoznamom hľadaných domén. V prípade, že sa nepodarí otvoriť certifikačné úložisko, napríklad z dôvodu, že nemáme dostatočné prístupové práva, uloží sa o tom záznam, ktorý sa neskôr zapíše do výstupného súboru.

4.2.2 Kontrola DNS cache

Ďalším modulom bude program na kontrolovanie DNS cache.

Na kontrolu DNS cache sme vytvorili jeden typ indikátora *"type": "dns"*, ktorý obsahuje jednu hodnotu určujúcu hľadaný reťazec.

Indikátory 4.3: dns záznamy

```
{
        "id": 1,
         "name": "dnsName",
         "type": "dns",
         "value": [
                  "nazov dns zaznamu"
         ]
}
```

Prvým pokusom o kontrolovanie DNS záznamov bolo použitie funkcie *DnsQuery*.

Táto funkcia bola zavrhnutá, pretože sa ňou nedajú kontrolovať regulárne výrazy. Momentálne na získanie záznamov používame *DnsGetCacheDataTable*, na ktorú som narazil na stackoverflow-e [16]. Ide o nedokumentovanú funkciu Windows API, ktorá získa všetky DNS záznamy vo formáte *wchar_t**. Neskôr sa síce od hľadania podľa regulárnych výrazov upustilo, ale nebol by problém ich pridať. Na stackoverflow-e bola priamo aj implementácia, tú bolo však nutné upraviť lebo nesprávne uvoľňovala pamäť.

4.2.3 Kontrola procesov

Úlohou tretieho modulu, ktorý aplikácia obsahuje, je kontrola procesov. Hľadať v zozname procesov by sa dalo podľa rôznych parametrov, ale najlogickejšie je hľadať podľa názvu procesu a v špecifikácii sme si definovali aj hľadanie podľa hash hodnoty ich spustiteľného súboru.

Indikátory na kontrolu procesov sú definované, ako *"type": "process-name"*, *"type": "process-regex"* a *"type": "process-hash"*. Prvé dva obsahujú jednu hodnotu určujúcu konkrétny reťazec a regulárny výraz, ktorý porovnávame s názvom procesu. Tretí obsahuje dve hodnoty prvou je typ hashu, ktorý môže byť *MD5*, *SHA1* alebo *SHA256* a druhou hodnotou je hash samotný.

Indikátory 4.4: procesy

```
{
    "id": 1,
    "name": "uzivatelov nazov",
    "type": "process-hash",
    "value": [
        "typ hashu",
        "hash"
    ]
},
{
    "id": 2,
    "name": "sample name",
    "type": "process-name",
    "value": [
        "explorer.exe"
    ]
},
```

```
{
    "id": 3,
    "name": "meno",
    "type": "process-regex",
    "value": [
        "explor.*\.exe"
    ]
},
```

Existuje viacero spôsobov na získanie bežiacich procesov z pamäte. Prvým spôsobom, ktorý sme vyskúšali bola funkcia *CreateToolhelp32Snapshot*. Táto funkcia je dokumentovaná a dokáže získať zoznam procesov a pomocných objektov, ktoré používajú a tak bola našou prvou voľbou. Neskôr sa bohužiaľ zistilo, že ak je proces, ktorý ju volá 32-bitový, ale systém je 64-bitový, funkcia skončí chybou. Problém by sa dal odstrániť tým, že by sa vytvorili dve distribúcie aplikácie, čo by ale mohlo pôsobiť zlým dojmom. Následne bol tento nápad úplne zavrhnutý objavením, že ak je aplikácia 64-bitová a funkcia sa zavolá na 32-bitový proces, funkcia vráti nesprávne alebo nekompletné informácie o procese. Alternatívou je nedokumentovaná funkcia *NtQuerySystemInformation*. Na oficiálnych stránkach Microsoftu síce funkciu nájdeme, ale nie je tam veľa informácií, takže väčšinu funkcionality sme objavili skúšaním rôznych vstupných parametrov, ktoré sme našli na internete [3]. Keď ju zavoláme s parametrom *SystemProcessInformation (5)*, dostaneme kompletný zoznam procesov.

Procesy prehľadávame na základe dvoch parametrov. Prvým je meno a druhým je hash hodnota spustiteľného súboru procesu. Na porovnanie mena nám stačia doteraz spomínané funkcie a administrátorské práva. Po zavolaní *NtQuerySystemInformation* dostaneme v štruktúre *SYSTEM_PROCESS_INFORMATION* zoznam všetkých procesov obsahujúci základné informácie o každom procese. Súčasťou týchto informácií je aj meno procesu zakódované v štruktúre *UNICODE_STRING*.

Na získanie hash hodnoty potrebujeme najprv zistiť, kde sa spustiteľný súbor nachádza. Toto sa dá zistiť pomocou *GetProcessImageFileName*. Tu sme narazili na problém, že ak nemáme prístupové práva k procesu, aby sme ho otvorili cez *OpenProcess*, nemôžeme zistiť jeho cestu. Ak je, ale aplikácia spustená s administrátorskými právami, dokáže si priradiť *SeDebugPrivilege*, ktorá dovoľuje pristupovať k ľubovoľnému procesu. Ďalším problémom je, že *GetProcessImageFileName* vracia cestu vo forme zariadenia. Takže napríklad *C:/Windows/explorer.exe* by mal tvar */Device-/Harddisk0/Partition1/Windows/explorer.exe*. Najprv teda musíme získať zoznam diskov pomocou funkcie *GetLogicalDriveStrings*, ktorý následne prekonvertujeme pomocou *QueryDosDevice* na správny formát. Potom nám už len stačí nájsť zhodu, získať hash súboru a porovnať ich. Informácie o hashovacích funkciách, ktoré používame náj-

dete v sekcii Hashovacie funkcie.

4.2.4 Kontrola Mutantov

Štvrtým podprogramom našej aplikácie je modul na kontrolu prítomnosti objektov vzájomného vylúčenia (muntantov).

Na kontrolu mutantov sme vytvorili jeden indikátor *"type": "mutex-name"*, ktorý obsahuje jednu hodnotu určujúcu meno mutanta.

Indikátory 4.5: mutanty

```
{
    "id": 1,
    "name": "sample name",
    "type": "mutex-name",
    "value": [
        "meno mutanta"
    ]
}
```

Predtým ako sa pozrieme na implementáciu si najprv povieme ako sa k mutantom dostaneme [15], [2]. Každý proces v systéme si pamätá zoznam *handle*, ktoré má aktuálne otvorené. Pod pojmom *handle* si môžeme predstaviť referenciu k prostriedku ako napríklad súbor, vlákno alebo kľúč v registroch, avšak hodnota handle nenesie žiadne informácie o objekte, na ktorý ukazuje, čo dovoľuje definovať systémové funkcie, ktoré pracujú s handle bez ohľadu na typ objektu. Hodnota handle je unikátna len vrámci procesu, čo znamená, že na zistenie typu a mena handle budeme potrebovať aj identifikačné číslo procesu.

Pozrime sa teraz na to, ako je celý proces implementovaný. Najprv musíme získať zo systému handle tabuľku. Pomocou funkcie *NtQuerySystemInformation* s parametrom *SystemHandleInformation (16)* získa aplikácia štruktúru *SYSTEM_HANDLE_TABLE*, ktorá obsahuje zoznam všetkých handle hodnôt a identifikátor procesu pre každú handle. Pri tomto volaní si treba dať pozor, pretože v dokumentácii je napísané, že ak funkcia skončí s chybou *STATUS_INFO_LENGTH_MISMATCH*, vráti nám v jednom s parametrov požadovanú veľkosť premennej. V tomto prípade to tak ale nie je a tak zdvojnásobujeme veľkosť premennej dovtedy, kým funkcia neskončí s hodnotou *STATUS_SUCCESS*. Teraz potrebujeme zistiť o každej handle na aký typ objektu ukazuje a ak je to mutant tak aké je jeho meno. Na získanie týchto informácií, potrebujeme najprv získať prístup k objektu, na ktorý handle ukazuje. Ten získa aplikácia tak, že pomocou funkcie *OpenProcess* s nastavením *PROCESS_DUP_HANDLE* otvorí proces a potom zavolá *NtDuplicateObject* a spraví si kópiu handle. Musíme si dať pozor,

pretože ak otvárame procesy, ku ktorým nemáme prístup, systém nás zamietne s chybovou hláškou *ERROR_ACCESS_DENIED*. Toto napravíme tak, že aplikáciu spustíme ako administrátor a počas behu programu si priradíme *SeDebugPrivilege*, ktoré nám dovoľujú otvoriť ľubovoľný proces. Následne zavolá aplikácia *NtQueryObject* s parametrom *objectTypeInformation (2)*. Teraz, ak objektom nie je mutant, pokračuje na ďalší handle, ak je objektom mutant, zavolá znova *NtQueryObject*, tentoraz ale s parametrom *objectNameInformation (1)*. Dôvodom, prečo rovno nezisťujeme meno je, že po prvé, by sme nevedeli určiť či je daný objekt naozaj mutant a po druhé, ak sa zavolá *NtQueryObject* s *objectNameInformation (1)* na objekt typu pomenovaná rúra *(Named Pipe)*, môže sa vykonávanie programu zaseknúť. Posledným krokom je už len vybratie mena zo štruktúry *UNICODE_STRING* a porovnanie.

4.2.5 Kontrola registrov

Ďalším modulom aplikácie je, modul na kontrolu registrov. Dokáže rekurzívne prehľadávať Windows registre a pri tom kontrolovať zhodu na kľúč, meno hodnoty a hodnotu.

Pre registre sú definované dva indikátory, prvý na vyhľadávanie podľa reťazca *"type": "registry"* a druhý podľa regulárneho výrazu *"type": "registry-regex"*. Oba majú v poli *value* 3 hodnoty. Prvá určuje cestu kľúča, druhá určuje meno hodnoty kľúča a posledná je samotná hodnota. Nastavením prázdneho reťazca sa daná hodnota ignoruje, takže ak napríklad vynecháme prvé dve hodnoty, budeme hľadať ľubovoľný kľúč s ľubovoľným menom hodnoty, ktorý má danú hodnotu.

Indikátory 4.6: registre

```
{
    "id": 1,
    "name": "uzivatelove meno",
    "type": "registry",
    "value": [
        "HKEY_LOCAL_MACHINE\\SOFTWARE",
        "Windows version"
        "12345"
    ]
}
{
    "id": 1,
    "name": "uzivatelove meno",
    "type": "registry-regex",
    "value": [
        "HKEY_LOCAL_MACHINE\\\\.*",
```

```
                "Win.*sion"
                "12.*34.*56"
            ]
        }
```

Windows registry obsahuje niekoľko základných kľúčov *HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, atď.*, v ktorých si môžu aplikácie ukladať dáta. Naša aplikácia postupne každý z týchto kľúčov otvorí a rekurzívne ho prehľadá. Otvorenie kľúča sa realizuje volaním funkcie *RegOpenKeyEx* s parametrom *KEY_READ* a *KEY_WOW64_64KEY*. Na 64-bitových systémoch Windows dochádza k virtualizácii registrov [2], čo znamená, že ak sa pokúsi 32-bitová aplikácia pristúpiť do registrov, je automaticky presmerovaná do špeciálneho podstromu. Napríklad ak sa pokúsi 32-bitová aplikácia pristúpiť do *HKCU\SOFTWARE* je presmerovaná do *HKCU\SOFTWARE\Wow6432Node*. Použitím parametra *KEY_WOW64_64KEY* sa táto virtualizácia vypne a my môžeme z 32-bitovej aplikácie prehľadávať celý register. Ak sa nám kľúč nepodarí otvoriť, napríklad preto, lebo nemáme oprávnenie, tak si to zaznamenáme a neskôr zapíšeme do výstupného súboru. Po otvorení kľúča, skontrolujeme jeho cestu a pokiaľ je zadaná hodnota alebo jej názov spustíme kontrolu hodnoty. Hodnotu porovnávame nasledovne. Najprv pomocou funkcie *RegQueryInfoKey* aplikácia zistí počet hodnôt kľúča. Potom postupne pre každú hodnotu zistí jej meno, typ a dáta pomocou *RegEnumValue*. Hodnota v registroch môže mať niekoľko typov, o každom si teraz povieme ako je reprezentovaný a ako ho porovnávame.

Prvým typom môžu byť binárne dáta *REG_BINARY*. Tie sú uložené ako sekvencia byte-ov. Naša aplikácia v tomto prípade očakáva, že hodnota dát, ktoré má v indikátore je v hexadecimálnom formáte a sama si dáta skonvertuje na binárne dáta a porovná. Ak vstupné dáta nie sú v hexadecimálnom formáte, aplikácia ich netestuje.

Ďalším druhom môže byť číselná hodnota. Windows podporuje hneď niekoľko formátov: *REG_DWORD, REG_QWORD, REG_DWORD_LITTLE_ENDIAN, REG_DWORD_BIG_ENDIAN, REG_QWORD_LITTLE_ENDIAN*. *DWORD* označujú 32-bitové a *QWORD* 64-bitové hodnoty. Aplikácia testuje dáta v indikátore, či sú číslo a či neprekračujú povolený rozsah pre danú premennú. V prípade *little endian* formátu otočia postupnosť bitov, aby zodpovedala danému formátu.

Tretím druhom je reťazec znakov. Takisto ako pri číslach, existuje viacero druhou: *REG_EXPAND_SZ REG_LINK REG_SZ*. Ide o ľubovolnú postupnosť znakov ukončenú null znakom *'\0'*. Aplikácia v tomto prípade len skonvertuje dáta na *wchar_t** postupnosť a porovná s dátami v indikátore.

Posledným typom je *REG_MULTI_SZ*. Ide sekvenciu sekvencií ukončenú null znakom *'\0'* napríklad *prva sekvencia\0druha sekvencia\0tretia sekvencia\0\0*. Bohužiaľ v json formáte nemôžeme zadávať null znaky a tak je v dátach v indikátore namiesto

toho vložená medzera. Naša aplikácia potom vezme dáta s registra a nahradí null znaky medzerami a porovná ich so vstupnými dátami.

Dva najväčšie problémy pri implementácii, boli kontrola správnosti hodnôt a problém s privilégiami na otvorenie kľúča.

Pri zavolaní *RegEnumValue* nám funkcia totiž vráti dáta v poli *BYTE**, ktoré treba skonvertovať na vhodné dátové štruktúry, aby sme ich mohli porovnať.

Druhým problémom je, že Windows sa pri prístupe ku kľúču skontroluje zoznam pre riadenie prístupu (access control list) (ACL) a pokiaľ užívateľ, v našom prípade administrátor, nie je v zozname povolený zamietne otvorenie kľúča. Tento problém bol konzultovaný, alternatívou by bolo otvoriť kľúč s parametrom *WRITE_OWNER*, prepísať vlastníka na administrátora a priradiť si ACL. Problémom je však, že aplikácia by takto príliš narušila registre, čomu sa chceme vyhnúť, takže sme sa rozhodli neotvárať ich, ale urobiť si záznam o tom, že tieto kľúče sme neotvorili.

4.2.6 Kontrola súborov

Predposledný podprogram našej aplikácie, je modul na kontrolu súborov.

Pre súbory máme definované takisto ako pre registre dva indikátory. Jeden pre hľadanie presných reťazcov *"type": "file"* a druhý na hľadanie podľa regulárneho výrazu *"type": "file-regex"*. Takisto ako pri hľadaní v registroch, ak nie je hodnota nastavená hľadáme ľubovoľnú, ktorá pasuje na zvyšné dve hodnoty. Pri hľadaní podľa presného reťazca obsahuje pole *value* štyri hodnoty. Prvou je cesta bez názvu súboru, druhou je názov súboru, tretia určuje, aký typ hashu hľadáme a štvrtá je samotný hash. Pri hľadaní podľa regulárneho výrazu, sa prvé dve hodnoty v *value* spájajú dokopy.

Hľadá súbor C:\windows\system32\aplikacia.exe s MD5 hash hodnotou 951c08a86e6416eb38cd9307bffb33d1

Indikátory 4.7: súbory

```
{
    "id": 1,
    "name": "app subor",
    "type": "file",
    "value": [
        "C:\\windows\\system32",
        "aplikacia.exe"
        "md5"
        "951c08a86e6416eb38cd9307bffb33d1"
    ]
}
```

```
{
    "id": 1,
    "name": "app subor",
    "type": "file-regex",
    "value": [
        "C:\\\\win.*\\\\app.*cia.*",
        "md5"
        "951c08a86e6416eb38cd9307bffb33d1"
    ]
}
```

Pri kontrolovaní súborov najprv aplikácia získa zoznam diskov pomocou *GetLogicalDriveStrings*. Následne rekurzívne prehľadáva každý z diskov. Funkcia *FindFirstFile* vráti prvý súbor v adresári a potom aplikácia pomocou *FindNextFile* prehľadá celý adresár. Funkcie pracujú so štruktúrou *WIN32_FIND_DATA*. Táto štruktúra obsahuje hodnotu *cFileName*, ktorú používame na porovnávanie s hľadaným menom a pomocou *dwFileAttributes* & *FILE_ATTRIBUTE_DIRECTORY* vieme určiť, či je súbor adresár. Pre každý súbor sa pozrie, či sa nenachádza v zozname hľadaných súborov. V prípade, že je súbor adresár rekurzívne sa do neho aplikácia zavolá a prehľadá ho.

4.2.7 Kontrola otvorených spojení

Posledným modulom je, program na kontrolovanie otvorených sieťových spojení.

Na definovanie otvorených sieťových spojení sme vytvorili tri indikátory. Prvý hľadá ip adresu *"type": "network-ip"*, druhý hľadá presnú zhodu s doménou *"type": "network-name"* a tretí hľadá doménu podľa regulárneho výrazu *"type": "network-regex"*. Každý má v poli *value* iba jednu hodnotu.

Indikátory 4.8: otovrené spojenia

```
{
    "id": 1,
    "name": "zla ip",
    "type": "network-ip",
    "value": [
        "192.168.1.2",
    ]
}
```

```
{
    "id": 2,
    "name": "dobra domena",
    "type": "network-name",
    "value": [
        "www.fmph.uniba.sk",
    ]
}
{
    "id": 2,
    "name": "dobra domena regex",
    "type": "network-regex",
    "value": [
        "www\..*uniba\.sk",
    ]
}
```

Funkcia na kontrolu otvorených sieťových spojení berie ako jeden s parametrov boolean *checkIpv6*. Dôvodom je, že ak nie je nainštalovaná podpora pre IPv6, funkcie, ktoré dané spojenia kontrolujú vykazujú nedefinované správanie.

Najväčšou prekážkou pri implementácii bolo, že funkcie na získavanie otvorených spojení sa zmenili medzi verziami service packov Windows XP. Tieto zmeny sa týkali funkcií *GetNameInfo* a *GetExtendedTcpTable, GetExtendedUdpTable*. Túto zmenu sme vyriešili tak, že aplikácia zistí na akej verzii Windows je spustená a podľa toho používa buď nové alebo staré funkcie.

Teraz k implementácii. Ako je uvedené vyššie aplikácia najprv skontroluje verziu systému pomocou funkcií *IsWindowsVistaOrGreater* v prípade, že ide o desktop verziu a *IsWindowsXPSP3OrGreater* v prípade server verzie a podľa toho používa funkcie. Windows neimplementuje funkcie na zisťovanie verzie serveru, namiesto toho, má funkciu *IsWindowsServer*, ktorá povie či sa jedná o server. Funkcia *IsWindowsXPSP3OrGreater* vráti *true* v prípade, že systém je Windows server 2008 alebo novší.

Najprv sa skontrolujú TCP spojenia. Funkcie *GetExtendedTcpTable* v prípade novšieho alebo *AllocateAndGetTcpExTableFromStack, GetTcpTable* v prípade staršieho systému sa zavolajú s parametrom *AF_INET* pre IPv4 a potom s *AF_INET6* pre IPv6, načítajú štruktúru *MIB_TCPTABLE* a *PMIB_TCP6TABLE_OWNER_PID*. Táto štruktúra obsahuje zoznam všetkých TCP spojení. Pomocou *GetNameInfo* potom môžeme získať zo záznamov v tejto tabuľke ich ip adresu a meno domény. Stojí za pripomenutie, že *GetNameInfo* musíme volať vždy s parametrom *NI_NUMERICSERV*,

pretože ak sa číslo portu neviaže na žiadnu známu službu, na systémoch Windows server 2003 funkcia skončí s chybou. Po skontrolovaní TCP spojení sa celý proces zopakuje pre UDP spojenia volaním *GetUdpTable*, *AllocateAndGetUdpExTableFromStack*, *GetExtendedUdpTable*.

4.2.8 Hashovacie funkcie

Pri voľbe hashovacej knižnice sme sa snažili zvoliť takú, ktorá by sa ľahko používala a podporovala všetky požadované algoritmy. Týmito sú MD5, SHA1 a SHA2-256. Prvým výberom bola Windows knižnica CryptoAPI. Táto knižnica poskytuje základnú funkcionalitu potrebnú pre vytváranie a spracovávanie zašifrovaných informácií. Podporuje aj vytváranie požadovaných hashov. Prekážkou je jej problematické používanie na systémoch Windows XP, kde je implementovaný SHA2-256 algoritmus až od XP SP3. Alternatívami boli Crypto++ a hashlib++, ale nakoniec zvíťazila OpenSSL, hlavne z dôvodu, že sme s ňou pracovali na iných projektoch a mali sme s ňou dobré skúsenosti. Knižnica sa ľahko používa, stačí zavolať *init* funkciu a potom pomocou *update* funkcie postupne, po krátkych úsekoch, vložiť súbor a následne zavolať *final* a knižnica sama vytvorí hash bez toho, aby sme museli niečo nastavovať.

4.2.9 Komunikácia so serverom

Požiadavkou pri výbere sieťovej knižnice bolo, aby podporovala najnovšiu sadu šifrovacích protokolov TLS 1.2. Windows ponúka sadu nástrojov Secure Channel (SCHANNEL), ktorá ale nepodporuje TLS 1.2 na všetkých požadovaných verziách a protokol je dostupný až od verzie Windows 7 a Windows Server 2008 R2 [13]. Keďže sme použili OpenSSL na vytváranie hashov, bolo logickou voľbou použiť ju ako náhradu. Ukázalo sa ale, že táto knižnica je pre naše potreby príliš nízko-úrovňová. Ďalšími alternatívami boli Boost.Asio a mbed TLS. Obidve boli zavrhnuté v prospech open source knižnice libcurl.

Libcurl ponúka vysoko-úrovňové API, ktoré značne uľahčuje implementáciu. Je možné ho linkovať s radom pomocných nástrojov, ktoré rozširujú jeho funkcionalitu bez toho, aby sme sa museli starať o ich funkcie. Použili sme verziu 7.40, ktorú sme skompilovali s OpenSSL funkcionalitou pomocou balíčka funkcií Visual Studio 2015. Menším problémom bolo, že knižnica používa funkciu *GetTickCount64*, ktorá je implementovaná až od Windows Vista. Neostalo nám teda nič iné, ako upraviť zdrojový kód a pridať dynamické načítavanie, aby keď sa aplikácia spustí na Windows XP, bola načítaná sa staršia verzia funkcie.

Použitie knižnice je jednoduché. Na začiatku sa zavolá *curl_easy_init*, ktorá inicializuje všetky potrebné komponenty. Následne sa pomocou *curl_easy_setopt* nastaví spôsob spojenia. Prvým zavolaním funkcie s konštantou *CURLOPT_URL* nastavíme

adresu serveru, na ktorú sa neskôr pripojíme a získame definície indikátorov. Nastavením *CURLOPT_SSL_VERIFYHOST* na 2 a *CURLOPT_SSL_VERIFYPEER* na 1 zabezpečíme, že server, z ktorého dáta sťahujeme je dôveryhodný. Nastavením *CURLOPT_CAINFO*, *CURLOPT_SSLCERT* a *CURLOPT_SSLKEY* určíme certifikát serveru, klienta a súkromný kľúč. Posledným krokom je zapnutie TLS 1.2 protokolu. Nastavením *CURLOPT_SSLVERSION* na *CURL_SSLVERSION_TLSv1_2* a *CURLOPT_USE_SSL* na *CURLUSESSL_ALL* zabezpečíme, že aplikácia používa TLS 1.2 na všetku komunikáciu so serverom. Zavolaním *curl_easy_perform* sa výsledná požiadavka vykoná a prijaté dáta sa uložia do lokálneho súboru, z ktorého neskôr prečítame definície indikátorov.

4.2.10 Získavanie privilégií

Viackrát sa v texte spomína priraďovanie si špeciálnych privilégií, ktoré nám umožňujú získať oprávnenia, ktoré normálna aplikácia nemá, ako napríklad pristupovať k informáciám o procese. Na stránkach Microsoftu sa nachádza návod ako si priradiť tieto privilégiá [12]. My používame informácie z tohto návodu, s miernymi úpravami, ktoré boli potrebné pre naše účely. Aplikácia získa bezpečnostný token pomocou *OpenProcessToken* a následne použije funkciu *AdjustTokenPriviliges* na nastavenie privilégií, ktoré požadujeme. Po tom ako sa vykoná kód, ktorý potrebuje špeciálne povolenia, zase privilégiá odoberie.

4.2.11 Práca s Json formátom

Vstupné aj výstupné dáta aplikácie sú vo forme Json-u. Na internete nájdeme množstvo knižníc, ktoré umožňujú manipuláciu s json-mi. Prvou voľbou bola knižnica JsonCpp [8]. Nastal však problém, keď sa ukázalo, že nedokáže pracovať s unicode znakmi. Toto sa neskôr ukázalo, ako veľký problém, pretože naša aplikácia pracuje, až na pár výnimiek, len s dátami v tomto tvare. Vhodnou alternatívou sa ukázala knižnica jsoncons [14]. Táto obsahuje špeciálnu triedu *wjson*, ktorá pracuje s unicode znakmi.

Záver

V tejto práci sme implementovali nástroj na detekciu indikátorov kompromitácie. Projekt bol náročný, vyžadoval si časovo náročné skúmanie jednotlivých funkcií a ladenie na rôznych verziách systému Windows a vyžiadal si tak viac času, ako sme pôvodne očakávali. Počas implementácie sme narazili na množstvo problémov, ktoré sme ale vyriešili. Naučili sme sa pracovať s funkciami Windows API, čo je podľa nás veľmi vzácna skúsenosť, lebo nie každému sa s nimi podarí prísť do kontaktu a zoznámili sme sa s užitočnými knižnicami na sieťovú komunikáciu, manipuláciu s jsonmi a vytváranie hashov.

Veľkým problémom sa ukázala práca s dátami v systéme Windows, pretože používa unicode znaky, s ktorými sme predtým do veľkej miery nepracovali a mali sme na začiatku problémy s ich konvertovaním pri kontrole registrov.

Na aplikácii sme otestovali všetku funkcionalitu, ale je možné, že v nej ešte ostalo zopár chýb, ktoré sa po nasadení prejavia a bude ich treba odstrániť.

Problémom, ktorý ostáva vyriešiť je, rozšírenie kompatibility na všetky distribúcie Windows XP a Windows server 2003, lebo momentálne je najnižšia podporovaná verzia Windows XP SP2 a server 2003 SP1. Poznáme problém, ktorý spôsobuje nekompatibilitu, ale z časových dôvodov ho nemôžme riešiť.

Možné vylepšenie vidíme v zmene reprezentácie dát v aplikácii. Momentálne sa na identifikáciu typu indikátora používa jeho priorita, čo nie je najvhodnejší spôsob, už len z dôvodu, že by sme mohli chcieť viacero indikátorov z rovnakou prioritou. Tento problém nie je akútny, ale s postupom času ho bude nutné vyriešiť.

Myslíme si, že aplikácia je v dobrom stave a po odstránení zostávajúcich nedostatkov, bude použiteľná ako náhrada za platené verzie rovnakých nástrojov a veríme, že sprístupnením bezplatného nástroja sa zvýši bezpečnosť vo firmách, ktoré si doteraz nemohli testovanie finančne dovoliť. Momentálne sme už dostali od CSIRT SK ďalšie požiadavky, ktoré by chceli mať implementované v aplikácii a ktoré sa budeme snažiť v nasledujúcich týždňoch zrealizovať.

Literatúra

[1] MANDIANT Corporation. Sophisticated indicators for the modern threat landscape: An introduction to OpenIOC. Technical report, MANDIANT Corporation, 2013.

[2] Martin Dráb. *Jádro systému Windows*. Computer Press, a. s., first edition, 2011. ISBN 978-80-251-2731-5.

[3] Matthew Graeber. Undocumented ntquerysysteminformation structures, 2013. Dostupné z `http://www.exploit-monday.com/2013/06/undocumented-ntquerysysteminformation.html`.

[4] Will Gragido. Understanding indicators of compromise *IOC* part i, 2012. Dostupné z `http://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-i/`.

[5] Will Gragido. Understanding indicators of compromise (IOC) part ii, 2012. Dostupné z `https://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-ii/`.

[6] Will Gragido. Understanding indicators of compromise (IOC) part iii, 2012. Dostupné z `https://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-iii/`.

[7] Chris Harrington. Sharing indicators of compromise: An overview of standards and formats. In *RSA Conference 2013*, 2013.

[8] Baptiste Lepilleur. Jsoncpp. Dostupné z `https://github.com/open-source-parsers/jsoncpp`.

[9] Hun-Ya Lock. Using IOC (indicators of compromise) in malware forensics. *SANS Institute*, 2013.

[10] David McMillen. Indicators of compromise finding the footprints that attackers leave behind when they breach your defenses. Technical report, IBM Corporation, 2015.

[11] Microsoft. Managing certificates with certificate stores. Dostupné z `https://msdn.microsoft.com/en-us/library/windows/desktop/aa386971(v=vs.85).aspx`.

[12] Microsoft. How to obtain a handle to any process with sedebugprivilege, 2006. Dostupné z `https://support.microsoft.com/en-us/kb/131065`.

[13] Kaushal Kumar Panday. Support for SSL/TLS protocols on windows, 2011. Dostupné z `https://blogs.msdn.microsoft.com/kaushal/2011/10/02/support-for-ssltls-protocols-on-windows/`.

[14] Daniel Parker. jsoncons. Dostupné z `https://github.com/danielaparker/jsoncons`.

[15] Mark Russinovich. *Windows internals part 1.* Microsoft Press, Redmond, Wash, 6th edition, 2012. ISBN 978-0735648739.

[16] Harry Johnston (`http://stackoverflow.com/users/886887/harry-johnston`). Retrieving what's in the dns cache. Dostupné z `http://stackoverflow.com/questions/7998176` a `http://venom630.free.fr/geo/autre_chose/cachedns.html`.