VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

HLOUBKOVÁ ANALÝZA PODOBNOSTI KÓDU V MALWARE KMENECH

IN-DEPTH ANALYSIS OF CODE SIMILARITY IN MALWARE STRAINS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN VOŠČINÁR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LUKÁŠ ZOBAL**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Voščinár Martin**
Program: Informační technologie
Název: **Hloubková analýza podobnosti kódu v malware kmenech**
In-Depth Analysis of Code Similarity in Malware Strains
Kategorie: Bezpečnost

Zadání:

1. Studujte metody statické a dynamické analýzy binárního spustitelného kódu jako jsou reverzní inženýrství, sandboxing či dekompilace. Dále zkoumejte problematiku škodlivého kódu (malware), jeho jednotlivých typů a v současnosti běžných kmenů.
2. Seznamte se s nástroji a metodami pro detekci podobnosti binárního spustitelného kódu malware vzorků.
3. Pomocí těchto technik proveďte analýzu minimálně 100 kmenů či jejich variant, dodaných společností Avast, za účelem odhalení dosud neznámých vzájemných vazeb ve smyslu podobnosti kódu či znovupoužití stejného kódu ve více kmenech.
4. Po domluvě s konzultantem ze společnosti Avast proveďte hloubkovou analýzu minimálně deseti vybraných detekovaných překryvů kódu pomocí metod reverzního inženýrství. Snažte se odhalit účel a původ takového kódu. Své poznatky důkladně zdokumentujte.
5. Dále využijte těchto poznatků pro návrh detekčních vzorů, které umožní tyto hrozby efektivně detekovat a to i u doposud neviděných variant. V případě potřeby rovněž aktivně vylepšujte používané nástroje a metody pro detekci této podobnosti.
6. Svoji práci zhodnoťte a uveďte výhled na další možný vývoj.

Literatura:

- KUBOV, Peter. Scalable Binary Executable File Similarity. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- SIKORSKI, Michael a Andrew HONIG. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012. ISBN 978-1593272906.
- Malpedia repository and The Big Zeus Family Similarity Rundown, dostupné online: https://pnx.tf/slides/zeus_similarity_showdown.html.
- EILAM, Eldad. Reversing: Secrets of Reverse Engineering. Wiley, 2005. ISBN 978-0764574818.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních tří bodů a rozpracování bodů čtyři a pět.

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

Vedoucí práce: **Zobal Lukáš, Ing.**
Konzultant: Křoustek Jakub, Ing., Ph.D., Avast
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cieľom tejto práce je analýza kmeňov malvéru za účelom odhalenia väzieb v zmysle podobnosti kódu či jeho znovupoužitia. Využívajú sa pri tom nástroje určené na detekciu podobnosti binárneho kódu spustiteľných súborov. Vybrané kmene sú ďalej skúmané metódami reverzného inžinierstva s cieľom odhaliť účel a pôvod tohto kódu. Na základe týchto poznatkov sú vytvorené detekčné vzory, ktoré takéto hrozby majú efektívne detegovať a zlepšiť ich klasifikáciu. Výskum zároveň poukazuje na nedostatky používaných nástrojov a prináša návrhy na ich vylepšenie. Na záver sú získané výsledky práce zhrnuté a zhodnotené s výhľadom do budúcnosti.

Abstract

The goal of this thesis is the analysis of malware strains with the aim to discover relationships in terms of code similarity or its reuse. Specialized tools are used for the detection of binary code similarity. Selected strains are then analyzed using reverse engineering techniques to uncover the purpose and origin of such code. Based on these findings, detection patterns are created, efficiently detecting those threats. This research also points out the shortcomings of used tools and proposes options for improvement. In conclusion, the obtained results of this thesis are summarized and evaluated with prospects for the future.

Kľúčové slová

Malvér, analýza malvéru, reverzné inžinierstvo, podobnosť binárneho kódu, YARA.

Keywords

Malware, malware analysis, reverse engineering, binary code similarity, YARA.

Citácia

VOŠČINÁR, Martin. *Hloubková analýza podobnosti kódu v malware kmenech.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Zobal

Hloubková analýza podobnosti kódu v malware kmenech

Prehlásenie

Prehlasujem, že som túto semestrálnu prácu vypracoval samostatne pod vedením Ing. Lukáša Zobala. Ďalšie informácie mi poskytli Ing. Jakub Křoustek, PhD. a Ing. Peter Kubov. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Voščinár
11. mája 2022

Poďakovanie

Predovšetkým by som chcel poďakovať vedúcemu práce Ing. Lukášovi Zobalovi za jeho odbornú pomoc pri tvorbe tejto práce a konzultantovi Ing. Jakubovi Křoustkovi, PhD. za poskytnutý čas, cenné rady a trpezlivosť. Taktiež ďakujem autorovi nástroja YaraZilla, Ing. Petrovi Kubovovi za ochotu a ponúknuté rady. Poďakovanie patrí aj Mgr. Ladislavovi Zezulovi zo spoločnosti Avast za užitočné informácie a pripomienky.

Obsah

1 Úvod **2**

2 Malvér a jeho analýza **3**
2.1 Malvér . . . 3
2.2 Analýza malvéru . . . 6
2.3 Nástroje na detekciu podobnosti kódu . . . 12

3 Analýza podobnosti kódu v kmeňoch malvéru **19**
3.1 Výber nástroja . . . 19
3.2 Štúdia A – Analýza kmeňov typu banker . . . 20
3.3 Štúdia B – Porovnanie vzoriek kmeňov WannaCry a Contopee . . . 23
3.4 Štúdia C – Analýza podobnosti verzií kmeňa Spora . . . 25
3.5 Štúdia D – Hľadanie doposiaľ neznámych vzťahov medzi kmeňmi . . . 29
3.6 Zhodnotenie . . . 32

4 Hĺbková analýza **33**
4.1 Analýza vývoja verzií rodiny Spora . . . 33
4.2 Rodiny Padodor a Kraton . . . 44
4.3 Rodiny LockBit a Zeoticus . . . 45
4.4 Rodiny s podobnosťami v staticky linkovanom kóde . . . 48

5 Vyhodnotenie výsledkov výskumu **53**
5.1 Zhodnotenie analýzy verzií rodiny Spora . . . 53
5.2 Zisťovanie výskytu spoločného kódu rodín LockBit a Zeoticus . . . 57
5.3 Návrh detekčných pravidiel . . . 58
5.4 Zhodnotenie analýzy nástrojom YaraZilla . . . 61

6 Záver **65**

Literatúra **67**

A Obsah priloženého pamäťového média **69**

B Tabuľky výsledkov analýzy kmeňov typu banker **70**

C Detailný popis stavov a priebehu spustenia malvéru Spora **75**

Kapitola 1

Úvod

V dnešnej dobe masívneho rozšírenia informačných technológií sa aspekty digitálneho sveta týkajú aj bežného človeka. Na jednej strane tieto technológie spájajú ľudí po celom svete, umožňujú prístup k dôležitým službám a uľahčujú každodenný život, na strane druhej rovnako poskytujú všetky tieto možnosti ľuďom, ktorí ich chcú zneužiť. Na každého používateľa týchto technológií tak číhajú hrozby mnohých podôb, ktoré sa stále vyvíjajú. Medzi ne patrí aj škodlivý softvér (*malicious software*) – *malvér*, ktorý dokáže spôsobiť obeti nepríjemnosti ako spomalenie zariadenia, poškodenie či stratu dát, odcudzenie citlivých informácií, stratu súkromia – špehovanie a mnoho iných. Malvér sa neustále vyvíja, preto je dôležité, aby sa súčasne taktiež zdokonaľovala obrana proti nim.

Na budovanie ochrany proti malvéru mu musíme najskôr porozumieť, na čo slúži analýza malvéru. Táto práca sa zameriava práve na tento proces. Malvéroví analytici skúmajú vzorky rôznych typov a klasifikujú ich do kmeňov alebo inak povedané rodín, teda skupín malvéru ktorý vznikol z rovnakého kódu. Vzorky bývajú vo forme binárnych spustiteľných súborov, na ich analýzu je tak potrebné využiť techniky *reverzného inžinierstva*. To zahŕňa celý rad techník, ktoré sú v práci popísané a ďalej využívané. Na obranu proti malvéru je nutné ho detegovať, na čo slúžia napríklad *detekčné vzory*, ktorým sa venuje táto práca. Na ich návrh sa v tejto práci používa jazyk YARA, ktorý umožňuje tvorbu tzv. *pravidiel*.

Medzi niektorými vzorkami malvéru teda existujú vzťahy, ktoré nám môžu pomôcť odhaliť špecializované nástroje na detekciu podobnosti spustiteľného kódu. Jeden z nich, vyvinutý v rámci diplomovej práce v spolupráci so spoločnosťou Avast sa nazýva YaraZilla a venuje sa mu značná časť tejto práce. Je použitý na odhalenie vzťahov medzi rodinami malvéru, ktoré sú ďalej podrobnejšie skúmané. Na základe tohto výskumu môžu byť vylepšené detekčné vzory nielen pre súčasné, ale aj budúce hrozby.

Kapitola 2 oboznamuje čitateľa o rôznych typoch malvéru, jeho často využívaných techník, na čo nadväzuje prehľadom možností jeho analýzy a používaných nástrojov.

Kapitola 3 obsahuje analýzu podobnosti kmeňov malvéru pomocou nástroja YaraZilla, ktorý je najskôr otestovaný v praxi a sú preskúmané jeho možnosti. Následne je použitý pri analýze viacerých verzií jedného kmeňa na preskúmanie ich vývoja a hromadnej analýze veľkého množstva kmeňov s cieľom objavenia doposiaľ neznámych vzťahov medzi nimi.

V kapitole 4 sú vybrané vzorky na základe analýzy v predchádzajúcej kapitole hlbšie preskúmané, pričom sa využívajú techniky reverzného inžinierstva.

Kapitola 5 interpretuje celkové výsledky výskumu, obsahuje návrh detekčných pravidiel pre odhalené doposiaľ neznáme vzťahy kmeňov a zhodnotenie práce s nástrojom YaraZilla.

Záverečná kapitola 6 ponúka súhrnné zhodnotenie dosiahnutých výsledkov a výhľad na budúci vývoj.

Kapitola 2

Malvér a jeho analýza

Pre skúmanie škodlivého kódu je kľúčové pochopiť definíciu tohto pojmu, jeho typy a ich používané techniky. Táto kapitola sa zameriava práve na hlavné informácie o malvéri. Taktiež sú tu rozobraté rozličné metódy analýzy malvéru a techniky reverzného inžinierstva. V poslednej časti sú priblížené nástroje určené na detekciu podobnosti binárnych spustiteľných súborov.

2.1 Malvér

Malvér – *malware*[1], škodlivý kód – je softvér vytvorený s cieľom útoku či zneužitia infikovaného zariadenia, jeho dát alebo majiteľa. Klasifikuje sa do *kmeňov* (*rodín*), čo sú skupiny malvéru ktorý má spoločný pôvod kódu. Skúmané škodlivé súbory môžeme označiť pojmom *vzorky*. Malvér môže mať nespočetné množstvo podôb a využívať široký záber rôznorodých techník. Rozdelenie do typov a niektoré dôležité techniky sú popísané v tejto sekcii.

2.1.1 Typy malvéru

Taxonómia malvéru sa zaoberá klasifikáciou malvéru do typov, predovšetkým na základe správania a účelu malvéru. Táto klasifikácia však nie je jednoznačná, pretože viaceré typy môžu zdieľať niektoré svoje vlastnosti. Konkrétne rodiny taktiež môžu kombinovať rôzne techniky špecifické pre určité druhy. Rôzne spoločnosti venujúce sa tejto skúmaniu malvéru preto majú často svoje vlastné varianty klasifikácie. V nasledujúcich odsekoch sú predstavené niektoré známe typy, ktorých definície sú všeobecne konzistentné, uvedené v literatúre popisujúcej analýzu [18], celkovo výskum a obranu proti malvéru [19], a konkrétne typy škodlivého softvéru [10] [3].

- **Vírus** predstavuje škodlivý kód s parazitickým správaním a schopnosťou replikácie. Jeho charakteristickou aktivitou je infikovanie neškodlivých súborov, odkiaľ pochádza aj jeho pomenovanie. Často to robí vložením svojho kódu do spustiteľných súborov, ktoré po spustení aktivujú vírus. Infikovanie súborov slúži tiež ako metóda perzistencie.
- **Červ** je druhom malvéru, ktorý je narozdiel od vírusov schopný pri šírení bez vonkajšej pomoci napádať aj na iné zariadenia. Obvykle pri útoku využíva nejakú skrytú zraniteľnosť operačného systému. Šíri sa zvyčajne v sieťach, no môže na to využívať aj prenosné médiá ako USB flash disky.

[1] malware – skratka z *malicious software* – škodlivý softvér

- **Trojan** alebo trójsky kôň je všeobecný pojem označujúci druh malvéru, ktorý maskovaním za užitočný softvér skrýva svoj zámer. Preto môžeme pod tento druh zaradiť množstvo iných z nižšie spomínaných, ktoré toto splňujú, ako backdoor, botnet, banker... Typicky sa narozdiel od vírusov či červov nevedia šíriť bez vonkajšej pomoci.
- **Backdoor** – zadné vrátka – je druh škodlivého kódu, ktorý infikuje zariadenia s cieľom poskytnutia prístupu útočníkovi. Útočník sa môže potom pripojiť k infikovanému zariadeniu a vykonávať na ňom rôzne druhy útokov.
- **Bot**, podobne ako backdoor umožňuje útočníkovi prístup k infikovaným zariadeniam, ale tieto zariadenia sú ovládané súčasne a centrálne sú im zasielané príkazy. Účelom ich zneužitia býva koncentrovaný útok na zvolený cieľ ako napr. formou DDoS[2], či útok hrubou silou (*brute-force attack*). Pojmom botnet sa môže označovať sieť napadnutých zariadení alebo druh malvéru bot.
- **Banker** alebo banking trojan sú špecifickým druhom trojanov, ktoré sa zameriavajú na odcudzenie financií napadnutých osôb najmä v prostredí internetového bankovníctva.

  Príkladom je rodina Zeus, ktorá sa prvý krát objavila v roku 2007. Zdrojový kód tohto malvéru bol však neskôr zverejnený, a tak vzniklo mnoho ďalších rodín, ktoré sa z neho vyvinuli, ako napríklad VM Zeus, Citadel, Atmos a Gameover.
- **Downloader** je druh malvéru, ktorý slúži, ako názov napovedá, na stiahnutie ďalšieho škodlivého kódu. Zvyčajne ich používajú útočníci pri prvotnom napadnutí zariadenia.
- **Keylogger** je druh škodlivého kódu, ktorý zaznamenáva na infikovanom systéme stlačené klávesy, prípadne pohyb myši či obsah obrazovky. Týmto spôsobom môže útočník získať prístup predovšetkým k citlivým dátam ako sú prístupové mená, heslá, osobné či platobné informácie.
- **Loader** je druh malvéru, ktorý má za úlohu spúšťať iné škodlivé programy, napríklad so zámerom utajenia či spustenia s väčšími prístupovými právami.
- **Ransomware** je pojem pre malvér, ktorý útočníci používajú na vymáhanie výkupného od obete. Po útoku obeti sľubujú za poplatok odstránenie napáchaných škôd. Podľa spôsobu útoku sa delia na rôzne druhy ako:
  - tie, ktoré šifrujú súbory na infikovanom zariadení, alebo k nim znemožňujú prístup,
  - znemožňujú prístup k samotnému zariadeniu,
  - odcudzujú citlivé informácie a vyhrážajú sa ich zverejnením,
  - metódami zastrašovania a sociálneho inžinierstva manipulujú obete, aby zaplatili za vyriešenie neexistujúceho problému. Tento druh sa bližšie nazýva *scareware*.

  Platby sú vykonávané najmä formou kryptomien zo snahou utajiť identitu útočníkov. Aj po ich uhradení často nedochádza k úplnému vyriešeniu problému – odblokovaniu zariadenia či sprístupneniu dát. Inštrukcie k platbe typicky zanechávajú útočníci v odkaze pre obeť – tzv. *ransom note*.

  Neslávne sa preslávil najmä ransomware WannaCry, ktorý v máji 2017 v priebehu jedného dňa infikoval viac ako 230 000 zariadení po celom svete a spôsobil straty v hodnote približne 4 miliardy USD [1].

[2] DDoS (distributed denial-of-service) – útok s cieľom zahltiť server a znemožniť jeho službu

- **Rootkit** je malvér skrývajúci činnosť iného škodlivého kódu, typicky vďaka nadobudnutiu privilegovaných oprávnení.
- **Stealer** Stealer alebo *information-stealing malware* je druh malvéru, ktorý sa pokúša odcudziť citlivé informácie. Príkladom je password stealer, ktorý sa zameriava na získavanie prihlasovacích údajov používaných v aplikáciách na napadnutom zariadení.

2.1.2 Vybrané metódy používané v malvéri

Pri skúmaní škodlivého softvéru sa malvéroví analytici neraz stretávajú s určitými technikami, ktoré autori škodlivého kódu aplikujú na zvýšenie jeho efektivity. V tejto podsekcii sú spomenuté niektoré z nich, ktoré môžu spôsobiť sťaženie analýzy a vyhnutie sa detekcii, alebo ďalšie, ktoré zabraňujú behu viacerých inštancií malvéru na napadnutom zariadení. Tieto techniky sú natoľko rozšírené, že sa objavujú aj pri analýze mnohých vzoriek v nasledujúcich kapitolách.

Metódy obfuskácie

Obfuskácia (zahmlievanie) je proces transformácie kódu s cieľom zamaskovania účelu jeho činnosti. Tvorcovia malvéru sa takto snažia predísť detekcii a zároveň môžu skomplikovať analýzu zachytených vzoriek. Existuje mnoho metód realizujúcich tento proces. V tejto časti budú predstavené najdôležitejšie z nich v kontexte škodlivého kódu.

Najjednoduchším spôsobom zhoršenia čitateľnosti kódu by bolo odstránenie bielych znakov, neprehľadné formátovanie alebo syntax, či mätúce identifikátory. Tieto úpravy sa bežne používajú v škodlivých skriptoch. Tvorca malvéru obvykle nemá v úmysle zdieľať zdrojový kód s verejnosťou, a teda pri preložených binárnych súboroch taká obfuskácia nemá zmysel. Vtedy je možné namiesto toho transformovať kód tak, že si zachová svoju funkcionalitu, ale syntaktický zápis bude oveľa komplikovanejší, prípadne bude obsahovať množstvo zbytočných príkazov. Na tieto transformácie tiež môžu tvorcovia malvéru niekedy vytvárať a využívať rôzne nástroje, ktoré tento proces automatizujú [19].

Packery [18] alebo packing programy sú určené na „zabalenie“ – packing binárnych súborov, ktoré sú využívané pri tvorbe malvéru z viacerých dôvodov. Ich využitím sa docieli zmenšenie spustiteľného súboru ale najmä skrytie jeho skutočnej funkcionality a znemožnenie statickej analýzy. Vtedy je nutné pred statickou analýzou vykonať unpacking spomínaný nižšie. Statickej analýze sa venuje podsekcia 2.2.2.

Základný princíp packerov je vytvorenie nového spustiteľného súboru, ktorého dáta tvorí skomprimovaný kód pôvodného spustiteľného súboru. Takýto súbor pri spustení zariadi dekomprimáciu a spustenie kódu pôvodného programu. Výsledné „zabalené“ – packed programy sú druhom obfuskovaných programov.

Bežné je využitie voľne dostupných packerov, akým je napríklad UPX[3], ale možné je aj použitie vlastného riešenia, čo komplikuje proces „rozbalenia“ – unpacking. Metódy získavania „rozbaleného“ – unpacked binárneho súboru sú rôzne. Všeobecne sa ale dajú rozdeliť do troch kategórií [18]:

1. automatizované statické,
2. automatizované dynamické,
3. manuálne.

[3]UPX – https://upx.github.io/

Najjednoduchšou metódou je použitie nástrojov určených na unpacking binárnych súborov. V prípade bežne rozšírených packerov je možnosťou využitie nástrojov statického druhu. Tie pracujú so znalosťou packing algoritmu, ktorú využívajú na získanie pôvodného súboru zo „zabaleného“ bez jeho spúšťania. Statické metódy sú špecifické pre daný packer, a teda pri zmene použitého riešenia prestávajú byť užitočné. Nástroje dynamických metód spúšťajú skúmaný spustiteľný súbor, čím umožnia jeho „rozbalenie“. Potom tento nástroj automaticky rekonštruuje hlavičku spustiteľného súboru a výsledný súbor je zapísaný na disk. Tento súbor nemusí byť identický s pôvodným.

Obe automatizované metódy majú obmedzenú úspešnosť [18], a tak pri ich zlyhaní je nutné použiť manuálne metódy. Medzi tie patrí napríklad zistenie algoritmu použitého pre packing a následnú implementáciu reverzného algoritmu, ale hlavne spustenie skúmaného programu v debuggeri, a vytvorenie výpisu pamäte. Keďže druhá spomenutá metóda patrí medzi dynamické, bližšie je vysvetlená v podsekcii 2.2.3, ktorá sa týka dynamickej analýzy malvéru.

Metódy predchádzania viacnásobnej infekcii

Beh viacerých inštancií rovnakej verzie malvéru na tom istom zariadení je typicky kontraproduktívny, autori malvéru sa teda často tejto situácii snažia predísť. Pokúšajú sa tak v kóde rôznym technikami implementovať detekciu už bežiacej inštancie tej istej verzie. Bežnými technikami je použitie synchronizačných mechanizmov, ako sú pomenované objekty, ale používajú sa aj špecificky nazvané súbory či kľúče a hodnoty v *registri*[4].

Mutex [12] je pomenovaný objekt používaný na synchronizáciu vlákien pri súčasnom prístupe k zdieľanému zdroju. Tvorcovia malvéru ich ale často využívajú na kontrolu, či systém nie je infikovaný rovnakým malvérom. Obvykle sa malvér pred vykonávaním škodlivej činnosti snaží vytvoriť mutex so špecifickým menom, a ak takýto existuje, svoju činnosť končí.

2.2 Analýza malvéru

Pre analýzu malvéru je dôležitý pojem *reverzné inžinierstvo* [4]. Jedná sa o proces analýzy systému, jeho súčastí a ich vzájomných vzťahov, s cieľom pochopiť ich fungovanie. Tento systém konkrétne pri analýze malvéru predstavuje program v podobe zachytenej vzorky. Proces reverzného inžinierstva teda dovoľuje skúmať štruktúru programu, spôsob a príčiny jeho správania aj bez prístupu k zdrojovému kódu.

V drvivej väčšine prípadov prebieha analýza na preložených spustiteľných súboroch, ktoré normálne nie sú čitateľné bez použitia nástrojov určených na tento účel. Nasledujúca časť predstavuje formát spustiteľných súborov, ktoré sú analyzované v rámci tejto práce. Pri analýze malvéru existujú dva základné prístupy [18], ktoré sú ďalej predstavené.

2.2.1 Formát spustiteľných súborov

Operačný systém UNIX a systémy z neho vychádzajúce, ako aj Linux, používajú ELF (Executable and Linkable Format) formát spustiteľných súborov a zdieľaných knižníc. Pretože ale drvivá väčšina malvéru je určená pre Windows, táto práca sa zameriava na malvér tejto platformy a formát PE (Portable Executable). Oba formáty však vychádzajú z podobných princípov.

[4]Windows Registry – systémová databáza Windows, určená najmä pre konfiguračné dáta

Súborový formát PE sa používa v súčasných verziách Windows pre spustiteľné súbory, ovládače aj knižnice DLL (Dynamic Link Library) [13] . Jedná sa o dátovú štruktúru obaľujúcu spustiteľný kód a obsahujúcu informácie dôležité pre jeho spustenie.

Súbor formátu PE začína hlavičkou, po ktorej nasledujú sekcie. Hlavička obsahuje informácie ako cieľovú architektúru súboru, čas vytvorenia súboru, počet sekcií a počiatočnú adresu programu. Nasleduje zoznam sekcií, ktorý obsahuje hlavne ich názov, adresu, veľkosť na disku a po načítaní v pamäti, či ich určenie – čítanie, zapisovanie, vykonávanie kódu.

Samotné sekcie môžu obsahovať dáta alebo spustiteľný kód. Názov a obsah sekcií môže byť ľubovoľný, no kompilátory bežne používajú zaužívané označenie:

- `.text` obsahuje spustiteľný kód,
- `.bss` obsahuje neinicializované dáta,
- `.data` obsahuje inicializované dáta,
- `.rdata` obsahuje inicializované dáta len na čítanie,
- `.idata` obsahuje tabuľku importovaných funkcií,
- `.edata` obsahuje tabuľku exportovaných funkcií.

Pri analýze najmä malvéru sa však na toto značenie vždy spoľahnúť nedá.

2.2.2 Statická analýza

Statická analýza programov je proces skúmania kódu bez jeho spúšťania. V kontexte analýzy malvéru sa jedná o analýzu spustiteľných súborov pomocou špecializovaných nástrojov, prípadne analýzu zdrojového kódu skriptov. Pod statickú analýzu spadajú techniky v nasledujúcich odsekoch.

Základné informácie o spustiteľnom súbore

Hlavička spustiteľného súboru obsahuje informácie ako typ súboru, použité funkcie v tabuľke importovaných funkcií, čo môže poskytnúť dôležité poznatky o činnosti malvéru. Určitú predstavu o funkcionalite spustiteľného súboru dokáže vytvoriť aj jednoduché skúmanie reťazcov obsiahnutých v danej vzorke. Medzi reťazcami sa môžu nachádzať napríklad cesty k súborom, webové adresy, či názvy mutexov.

Disassembling

Disassembling je proces transformácie strojového kódu binárnych súborov na zápis v jazyku symbolických inštrukcií. Nástroje, ktoré zaisťujú tento proces a následné zobrazenie získaných inštrukcií ponúkajú pohľad na kód skúmanej vzorky, aj keď síce na najnižšej úrovni abstrakcie. Medzi tieto nástroje patria napríklad IDA Pro – spomenutá v podsekcii 2.2.4 – či Ghidra[5]. Analýza kódu v assembly s dostatočnými znalosťami dokáže poskytnúť nemalé množstvo informácií o činnosti skúmanej vzorky. Skúmaním konštrukcií kódu a ich postupnosti môžeme odhaliť techniky malvéru, kedy okolnosti ich použitia.

Dekompilácia

Dekompilátory posúvajú možnosti disassemblerov o krok ďalej, a to tým, že dokážu vytvoriť z kódu v assembly lepšie čitateľný pseudokód, často podobný jazyku C. Ponúkajú teda

[5]Ghidra – https://ghidra-sre.org/

pohľad na kód na vyššej úrovni abstrakcie, čo môže často analýzu zrýchliť. Príkladom takéhoto nástroja je Hex-Rays Decompiler, spomínaný v podsekcii 2.2.4, či RetDec[6] od spoločnosti Avast.

2.2.3 Dynamická analýza

Dynamická analýza malvéru je proces skúmania kódu, počas ktorého sú vzorky spúšťané v kontrolovanom prostredí. Typicky teda neprebieha priamo na fyzickom zariadení aby nedošlo k infekcii alebo šíreniu, prípadne z dôvodu simulovania špecifických podmienok pri behu. V prípade správneho použitia dokáže jednoduchšie odhaliť funkcionalitu skúmaných vzoriek. Metódy dynamickej analýzy sú popísané v nasledujúcich odsekoch.

Monitorovanie spustených vzoriek

Základná analýza je možná aj pri obyčajnom spustení, hlavne ak je činnosť spustenej vzorky monitorovaná vhodným nástrojom. Príkladom sú Process Explorer[7] a Process Monitor[8], ktoré dokážu monitorovať aktivitu spúšťaných procesov, súborového systému a registrov. Týmto spôsobom je teda možné zistiť dôležité informácie o činnosti malvéru.

Spúšťanie vzoriek v debuggeri

Pre odhalenie vnútorného stavu malvéru pri jeho spustení je potrebné použiť debugger, čo je nástroj, ktorý umožňuje vykonávanie inštrukcií po krokoch. Dovoľuje tiež vytváranie *breakpointov*, na ktorých sa pozastaví činnosť programu, manipuláciu registrov, či priamu úpravu inštrukcií za behu. Toto umožňuje pokročilú analýzu behu skúmanej vzorky malvéru ale aj spustenie častí kódu, ktoré sú dosiahnuteľné iba v špecifických podmienkach. Použitie debuggera je vhodné kombinovať so statickou analýzou pomocou disassemblingu či dekompilácie.

Unpacking pomocou debuggera

Dynamická analýza môže tiež slúžiť na unpacking, získanie „rozbalených“ – unpacked binárnych súborov, ktoré sú opísané v podsekcii 2.1.2. Toto je možné spustením vzorky, ktorá sa v pamäti „rozbalí“, zastavením vykonávania na vstupnom bode programu a následným vytvorením výpisu pamäte (memory dump). Potom je ale ešte potrebné vytvoriť hlavičku a tabuľku importovaných funkcií tohto spustiteľného súboru, čo môže byť celkom zdĺhavý proces. Tento proces ale našťastie uľahčujú niektoré nástroje, ako zásuvný modul OllyDump pre OllyDbg, spomínané v časti 2.2.4. Avšak tieto nástroje nedokážu rekonštruovať tabuľku importovaných funkcií v prípadoch, kedy ju vzorka sama nevytvára a využíva inú metódu riešenia importov. Vtedy je nutné ju zostaviť manuálne, napríklad pomocou nástroja Import Reconstructor [18].

Sandboxing

Sandbox je nástroj dynamickej analýzy [18], ktorý pozostáva zo zabezpečeného virtuálneho prostredia, ktoré obvykle simuluje sieťovú komunikáciu a všetky potrebné prostriedky,

[6] RetDec (Retargetable Decompiler) – https://retdec.com/
[7] https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer
[8] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

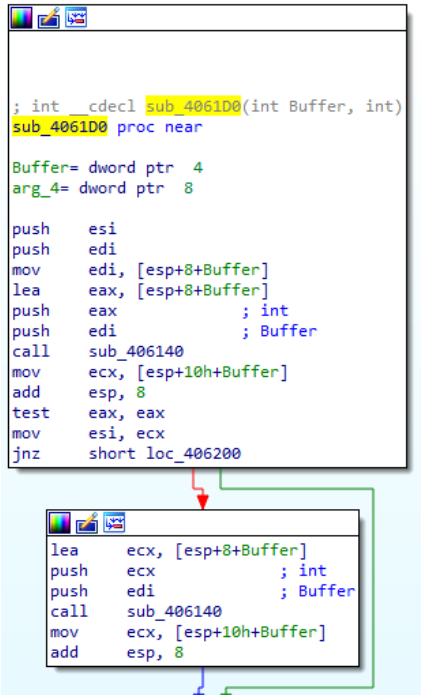
aby malvér bežal vo väčšine prípadov rovnako ako na reálnom infikovanom zariadení. Sandboxy poskytujú automatizovanú dynamickú analýzu vzoriek a nakoniec vygenerujú report z analýzy. Report väčšinou obsahuje záznam udalostí súborového systému, aktivity registrov, vytvorených mutexov a sieťovej komunikácie. Príkladom je nástroj Cuckoo[9].

2.2.4 Nástroje na analýzu a detekciu malvéru

Pri analýze malvéru sa používajú rozličné nástroje [18] využívajúce metódy statickej alebo dynamickej analýzy, prípadne ich kombinácie. Nasledujúce časti obsahujú zoznam a popis nástrojov použitých pri tvorbe tejto práce.

IDA Pro

IDA Pro[10] je pravdepodobne najrozšírenejší disassembler. Je to nástroj, ktorý umožňuje generovanie kódu v assembly zo spustiteľných súborov mnohých formátov, určených pre rôzne operačné systémy a architektúry procesorov. Zároveň pri analyzovaní súboru vykonáva automatickú analýzu kódu, kedy zisťuje výskyt adries funkcí a dát v rôznych častiach kódu (*cross-reference*). Dokáže identifikovať funkcie a ich parametre zo štandardných API[11] operačných systémov. Na detekciu staticky linkovaného kódu využíva technológiu F.L.I.R.T.[12].



Obr. 2.1: Ukážka z IDA Pro v zobrazení grafu

Zobrazenie disassembly je možné v móde grafu, ktorý okrem iného prehľadnejšie vizualizuje skoky v kóde, ako môžeme vidieť na obrázku 2.1. Zobrazenie dát či funkcií sa dá podľa želania upraviť v prípade, že nie sú správne automaticky identifikované. Názvy

[9] Cuckoo – https://cuckoosandbox.org/
[10] IDA Pro (Interactive Disassembler) – https://hex-rays.com/ida-pro/
[11] API (Application Programming Interface) – rozhranie pre programovanie aplikácií
[12] F.L.I.R.T. – https://hex-rays.com/products/ida/tech/flirt/

funkcií, návestí či premenných sa dajú ľubovoľne meniť, k inštrukciám je možné vkladanie komentárov. Aktuálny stav pri analýze vzorky sa ukladá do databázového súboru.

Okrem grafického rozhrania ponúka textové rozhranie pri spustení prostredníctvom príkazového riadku. Taktiež cez príkazový riadok umožňuje spustenie v tzv. *batch móde*, kedy je vstupný súbor spracovaný a jeho kód v disassembly spolu s databázovým súborom sú uložené do výstupných súborov.

Tento disassembler tiež umožňuje zobrazenie reťazcov v binárnom súbore alebo tabuliek importovaných a exportovaných funkcií. Podporuje pripojenie debuggera, či už vzdialeného alebo lokálneho, a jeho použitie v rozhraní nástroja. Zároveň ponúka možnosť zásuvných modulov (plug-in), ktoré ešte viac rozširujú jeho možnosti. Zásuvný modul IDAPython, ktorý je aktuálne predvolenou súčasťou IDA Pro, prináša možnosť použitia jazyka Python v skriptoch, ktoré môžu využívať funkcionalitu nástroja pomocou IDA API.

IDA vo verzii Pro je komerčný nástroj integrujúci dekompilátor s názvom Hex-Rays Decompiler[13]. Jedná sa o zásuvný modul, ktorý rozširuje analýzu o možnosť dekompilácie. Vytvára pseudokód podobný C, čo môže analýzu ešte viac uľahčiť, keďže kód v takejto podobe je často prehľadnejší.

OllyDbg

OllyDbg[14] je jeden z najrozšírenejších debuggerov. Poskytuje debugging 32-bitových spustiteľných súborov (a to vrátane knižníc formátu DLL) prostredníctvom grafického rozhrania. Je vhodný na unpacking vzoriek, hlavne v kombinácii so zásuvným modulom OllyDump, ktorý tento proces zjednodušuje.

YARA

YARA[15] je nástroj, ktorý slúži na detekciu malvéru. Používa sa najmä na identifikáciu a klasifikáciu vzoriek. Bol vyvinutý v roku 2013 Victorom Alvarezom zo spoločnosti VirusTotal, ktorá ho naďalej spravuje. V súčasnosti je jeho použitie rozšírené v mnohých spoločnostiach skúmajúcich malvér, medzi ktoré patria aj Avast, ESET, či Kaspersky Lab.

Jeho základom je deklaratívny jazyk [21], prostredníctvom ktorého sa vytvára popis hľadaných vzorov vo vzorke nazývaný *pravidlo*. Pravidlá sa skladajú z troch základných častí:

- metadáta,
- reťazce,
- podmienka.

Metadáta, začínajúce kľúčovým slovom `meta`, sú nepovinnou časťou pravidla. Slúžia iba na uchovanie rôznych informácií k danému pravidlu. Tvoria ich identifikátory a im pridelené hodnoty, ktoré sú nepodstatné pre fungovanie pravidla. Príkladom môže byť meno autora, typ pravidla či typ malvéru.

Časť reťazce, definovaná kľúčovým slovom `strings`, je taktiež nepovinná. Obsahuje zoznam identifikátorov reťazcov, začínajúcich znakom `$`, ich hodnôt a prípadne modifikátorov. Tieto reťazce sa delia na tri druhy:

- hexadecimálne,

[13] Hex-Rays Decompiler – https://hex-rays.com/decompiler/
[14] OllyDbg – https://www.ollydbg.de/
[15] YARA – http://virustotal.github.io/yara/

- textové,
- regulárne výrazy.

Hexadecimálne reťazce slúžia na definovanie neinterpretovaných binárnych postupností bytov. Okrem znakov hexadecimálnych číslic môžu obsahovať špeciálne konštrukcie, a to tzv. *zástupné znaky*, *skoky* a *alternatívy*. Hodnota hexadecimálneho reťazca musí byť ohraničená zloženými zátvorkami – `{}`.

Zástupné znaky `?` – *wildcards* predstavujú práve jednu ľubovoľnú číslicu `0-F` na danom mieste. Skoky `[`$\texttt{N}_{\texttt{min}}$`-`$\texttt{N}_{\texttt{max}}$`]` – *jumps* zastupujú ľubovoľný počet týchto číslic, kde rozsah tohto počtu je ohraničený hodnotami $\texttt{N}_{\texttt{min}}$ a $\texttt{N}_{\texttt{max}}$. Musí platiť, že $0 \leq \texttt{N}_{\texttt{min}} \leq \texttt{N}_{\texttt{max}}$, pričom ak niektorá hranica chýba, implicitne je nahradená nulou respektíve nekonečnom.

Alternatívy `A|B` – *alternatives* vymedzujú viacero možností pre príslušnú časť reťazca, kde `A` a `B` predstavujú ľubovoľnú postupnosť bytov, ktorá môže obsahovať aj zástupné znaky. Na odlíšenie znakov patriacich alternatíve sa používajú zátvorky: `(A|B)`.

Textové reťazce slúžia predovšetkým na definovanie reťazcov čitateľných znakov, no môžu obsahovať aj znaky zapísané niektorými *escape sekvenciami* známymi napríklad z jazyka C. Medzi podporované patria: `\"`, `\\`, `\r`, `\t`, `\n`, `\xHH`, kde `H` je hexadecimálna číslica. Textové reťazce podporujú tiež modifikátory ako napríklad `ascii` – ASCII reťazec, `wide` – reťazec kódovaný dvoma bytami na znak, `nocase` – reťazec bez ohľadu na veľkosť písmen. Hodnota reťazca je zapísaná v úvodzovkách `"`, za ktorou môžu nasledovať modifikátory.

Regulárne výrazy v jazyku YARA zú založené na ich implementácii v jazyku Perl. Okrem toho majú rovnaké vlastnosti ako textové reťazce, len sú ohraničené lomkami `/`.

Podmienka je jedinou povinnou časťou YARA pravidla. V zásade je to boolovský výraz, ktorý môže obsahovať kľúčové slová, identifikátory reťazcov, konštanty, referencie na iné pravidlá a boolovské, relačné, aritmetické, bitové či reťazcové operátory. Odkazovať môže na konkrétne jednotlivé reťazce alebo viacero naraz pomocou zástupných znakov `*` v rámci identifikátora, či použitím frázy ako `any of them` – všetky reťazce, `2 of them` – aspoň 2 reťazce.

Tzv. *moduly* predstavujú spôsob rozšírenia funkcionality nástroja YARA. Môžu poskytovať rôzne funkcie, konštanty či dátové štruktúry, ktoré dovoľujú vytvárať komplexnejšie podmienky v pravidlách. Niektoré moduly, ako aj tie nasledovné, sú distribuované spolu s nástrojom:

1. **PE** – dokáže spracovať vlastnosti spustiteľných súborov formátu PE ako vstupná adresa programu, exportované aj importované funkcie, či informácie o hlavičke a sekciách.
2. **ELF** – je veľmi podobný modulu PE, len je určený pre formát ELF.
3. **Cuckoo** – umožnuje tvorbu behaviorálnych pravidiel. Odlišuje sa od ostatných modulov tým, že pracuje s reportom, ktorý generuje sandbox Cuckoo, ktorý je spomínaný v podsekcii 2.2.3. V podmienkach pravidla sa potom dá odkazovať na výskyt udalostí v takomto reporte, ako napríklad HTTP požiadavok na špecifikovanú adresu, prístup k danému súboru v súborovom systéme, či vytvorenie mutexu.

Nasledujúce ukážky predstavujú príklady skutočných pravidiel pre vzorky ransomware rodiny `Legion`.

```
rule legion_known_sequences
{
    strings:
```

```
        $s01 = "%s._%02i-%02i-%02i-%02i-%02i-%02i_$%s$%s"
        $s02 = "f_tactics@aol.com"
        $s03 = "http://tuginsaat.com/wp-content/themes/twentythirteen/stats.php"
        $s04 = "centurion_legion@aol.com"
        $s05 = "http://pedi-protexx.com/envisiondreambig/wp-includes/Text/stats.php"
        $h01 = {
                B? 27  // mov dl, 27h
                80 7? FE 20 // xor byte ptr [eax-2], 20h
                80 7? FF 21 // xor byte ptr [eax-1], 21h
                80 3? 22 // xor byte ptr [eax], 22h
                80 7? 01 23 // xor byte ptr [eax+1], 23h
                80 7? 02 24 // xor byte ptr [eax+2], 24h
                80 7? 03 25 // xor byte ptr [eax+3], 25h
                80 7? 04 26 // xor byte ptr [eax+4], 26h
                30 ?? 05 // xor [eax+5], dl
            } // should match this XOR encryption sequence even if the registers change
    condition:
        2 of them
}
```

Výpis 2.1: Príklad statického pravidla

Pravidlo na výpise 2.1 môžeme označiť ako statické, pretože pracuje len s reťazcami zahrnutými v skúmanej vzorke. Použité reťazce sú veľmi špecifické pre skúmané vzorky, na základe čoho boli aj zvolené. Hexadecimálny reťazec `$h01` slúži na zachytenie kľúčového algoritmu malvéru.

```
rule legion_known_behavior_high
{
    condition:
        cuckoo.filesystem.file_write(/\$f_tactics@aol\.com\$//) or
        cuckoo.filesystem.file_write(/\$centurion_legion@aol\.com\$/) or
        cuckoo.filesystem.file_write(/\$\.legion$/) or
        cuckoo.filesystem.file_write(/\$\.cbf$/) or
        cuckoo.filesystem.file_write(/\\Desktop\\read_this_file\.txt$/) or
        cuckoo.network.http_get(/http:\/\/tuginsaat\.com\/wp-content\/themes\/twentythirteen
            \/stats\.php/) or
        cuckoo.network.http_get(/http:\/\/pedi-protexx\.com\/envisiondreambig\/wp-includes\/
            Text\/stats\.php/)
}
```

Výpis 2.2: Príklad behaviorálneho pravidla

Výpis 2.2 obsahuje príklad behaviorálneho pravidla, kde sa využíva modul `cuckoo`. Volania `cuckoo.filesystem.file_write` slúžia na detekciu zápisu zašifrovaných súborov, ktoré majú v tomto prípade špecifickú príponu, a tzv. *ransom note* – odkaz tvorcov ransomware pre obeť. Volania `cuckoo.network.http_get` zas zisťujú, či došlo k odoslaniu HTTP GET požiadavku na špecifikovanú adresu.

2.3 Nástroje na detekciu podobnosti kódu

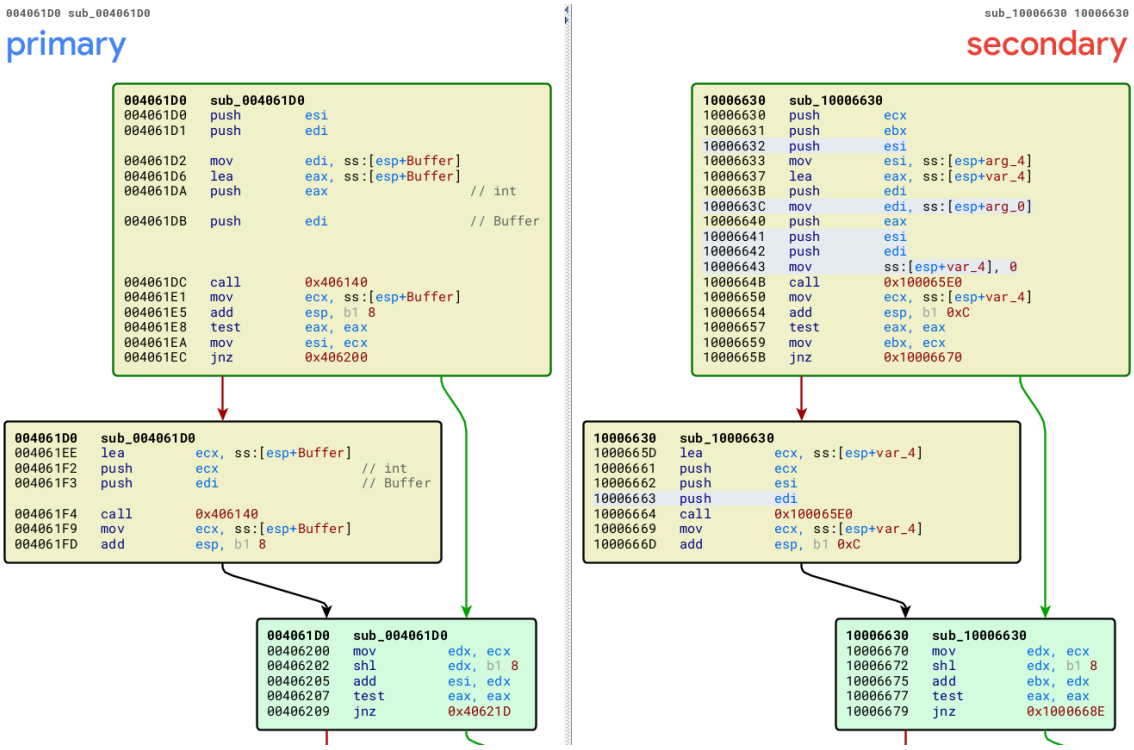
Okrem rôznych nástrojov slúžiacich na analýzu spustiteľných súborov existujú špecializované nástroje, ktoré dokážu zistiť rozdiely medzi binárnym kódom takýchto súborov a vyčísliť ich vzájomnú podobnosť. Vďaka nim je možné skúmať napríklad vývoj jednotlivých verzií rodiny malvéru alebo vzťahy medzi vzorkami jednotlivých rodín.

2.3.1 BinDiff

BinDiff[16] je nástroj určený na zistenie podobnosti dvoch binárnych súborov analyzovaných v disassembleri. Nástroj sa používa v dvoch krokoch – zistenie podobnosti kódu a vizualizácia týchto výsledkov. Zistenie podobnosti prebieha z dvojice databázových súborov disassemblera, z ktorých sa exportuje a porovná ich kód. Výsledné porovnanie sa uloží do súboru vlastného formátu.

Nástroj udáva celkovú percentuálnu podobnosť binárnych súborov a zároveň podobnosť každej funkcie. Funkcie sú kategorizované podľa toho, či v nich bola nájdená zhoda medzi porovnávanými súbormi alebo nie, a teda sa vyskytujú iba v jednom. Zhody funkcií sú určené viacerými parametrami ako napríklad haš funkcie, zhodná postupnosť volania alebo signatúry funkcií importovaných z knižníc. Okrem podobnosti sú funkciám pridelené hodnoty „confidence“, ktoré označujú mieru spoľahlivosti metódy použitej pri detekcii zhody.

BinDiff je možné použiť samostatne cez textové rozhranie prostredníctvom príkazového riadku, kde umožňuje zistenie a exportovanie výsledkov podobnosti. Je tiež dostupný v podobe zásuvného modulu pre disassembler IDA Pro a Ghidra[17]. Poskytuje tiež grafické rozhranie určené iba pre vizualizáciu výsledkov.



Obr. 2.2: Ukážka z grafického rozhrania nástroja BinDiff

Exportovaný BinDiff súbor je možné zobraziť grafickým rozhraním nástroja alebo v rámci IDA Pro. V grafickom rozhraní, ktoré je ukázané na obrázku 2.2, sú farebne odlíšené rozdiely v kóde, kde zhodné bloky funkcie sú zafarbené na zeleno, bloky s rozdielmi žltou alebo červenou podľa miery podobnosti. Takáto vizualizácia môže pomôcť upriamiť pozornosť na skúmané rozdiely.

[16] BinDiff –https://www.zynamics.com/bindiff.html
[17] Ghidra – https://ghidra-sre.org/

2.3.2 Diaphora

Diaphora [18] je nástroj určený na zistenie podobnosti dvoch binárnych súborov analyzovaných v disassembleri. Predstavuje konkurenciu nástroja BinDiff, pretože ponúka veľmi podobnú funkcionalitu. Pracuje taktiež s databázovými súbormi IDA Pro, no pri analýze využíva aj dekompilátor Hex-Rays, ak je dostupný. Nástroj je distribuovaný iba ako skript v jazyku Python, ktorý je potrebné spustiť cez možnosť „Script file“ alebo pomocou interpretu jazyka Python v nástroji IDA Pro. Výsledky exportuje do formátu SQLite.

Skript je potrebné spustiť dvakrát – prvý raz pre exportovanie kódu prvej vzorky, druhý pre exportovanie kódu druhej vzorky a jeho porovnanie s prvou. Vo výstupe sú funkcie zoskupené podľa zhody, no narozdiel od nástroja BinDiff sú vytvorené tri kategórie: „Best matches“, „Partial matches“, „Unreliable matches“. Kategórie značia spoľahlivosť metódy určenia zhodnej funkcie, akými sú haš funkcie, zhodný dekompilovaný pseudokód, zhodná postupnosť volania či špecifické hodnoty použitých konštánt.

```
1int __cdecl sub_4061D0(int Buffer, int a2)
2{
3  int v2; // edi
4  int result; // eax
5  int v4; // ecx
6  int v5; // esi
7  int v6; // esi
8  int v7; // esi
9
10  v2 = Buffer;
11  result = sub_406140(Buffer, (int)&Buffer);
12  v4 = Buffer;
13  v5 = Buffer;
14  if ( !result )
15  {
16    result = sub_406140(v2, (int)&Buffer);
17    v4 = Buffer;
18  }
19  v6 = (v4 << 8) + v5;
20  if ( !result )
21  {
22    result = sub_406140(v2, (int)&Buffer);
23    v4 = Buffer;
24  }
25  v7 = (v4 << 16) + v6;
26  if ( result ||
   (result = sub_406140(v2, (int)&Buffer)) != 0 )
27    *(_DWORD *)a2 = 0;
28  else
29    *(_DWORD *)a2 = v7 + (Buffer << 24);
30  return result;
31}
```

```
1int __cdecl sub_10006630(int a1, int a2, _DWORD *a3)
2{
3  int result; // eax
4  int v4; // ecx
5  int v5; // ebx
6  int v6; // ebx
7  int v7; // ebx
8  int v8; // [esp+Ch] [ebp-4h] BYREF
9
10  v8 = 0;
11  result = sub_100065E0(a1, a2, &v8);
12  v4 = v8;
13  v5 = v8;
14  if ( !result )
15  {
16    result = sub_100065E0(a1, a2, &v8);
17    v4 = v8;
18  }
19  v6 = (v4 << 8) + v5;
20  if ( !result )
21  {
22    result = sub_100065E0(a1, a2, &v8);
23    v4 = v8;
24  }
25  v7 = (v4 << 16) + v6;
26  if ( result ||
   (result = sub_100065E0(a1, a2, &v8)) != 0 )
27    *a3 = 0;
28  else
29    *a3 = v7 + (v8 << 24);
30  return result;
31}
```

Obr. 2.3: Ukážka zo zobrazenia rozdielov pseudokódu v nástroji Diaphora

Vizualizáciu ponúka priamo v IDA Pro, kde sú dostupné viaceré režimy zobrazenia. Zobrazenie grafu, kde sú farebne odlíšené bloky funkcie podľa úrovne zhody je podobné nástroju BinDiff, no nie sú zvýraznené konkrétne odlišnosti inštrukcií. Odlišnosti zobrazuje režim disassembly aj režim pseudokódu, ktorý je ukázaný na obrázku 2.3. Zobrazenie rozdielov pseudokódu je niečo, čo Diaphora ponúka oproti nástroju BinDiff, a môže uľahčiť analýzu rozdielov v skúmanom kóde.

[18] Diaphora – https://github.com/joxeankoret/diaphora

2.3.3 YaraZilla

YaraZilla je nástroj určený na skúmanie podobnosti binárnych súborov. Vznikol v rámci diplomovej práce v roku 2021, v spolupráci so spoločnosťou Avast [9]. Práca s týmto nástrojom je možná v internej sieti Avast prostredníctvom grafického rozhrania vo forme webovej aplikácie alebo API[19], s ktorým sa komunikuje pomocou HTTP požiadavkov. Nasledujúce odseky popisujú vlastnosti služby YaraZilla, podrobnejšie informácie sú v diplomovej práci.

Režimy analýzy

Nástroj funguje v nasledujúcich režimoch:

- **Inšpekcia** zisťuje podobnosť jedného binárneho súboru poskytnutého používateľom s množstvom ďalších. Využíva na to databázu s referenčnými dátami kmeňov malvéru, ktoré vznikli spracovaním ich vzoriek.
- **Extrakcia** umožňuje analyzovať podobnosti ľubovoľného počtu vstupných binárnych súborov. Súbory nie sú porovnávané s obsahom databázy, ale navzájom medzi sebou. Nástroj umožňuje uloženie výsledných dát do referenčnej databázy rozšírením existujúcej rodiny alebo vytvorením novej rodiny.

Úrovne abstrakcie binárnych dát

Ako pri inšpekcii, tak pri extrakcii nástroj využíva tri prístupy k vyhodnoteniu podobnosti:

- **Syntaktická podobnosť** pracuje priamo s binárnym kódom súboru. Pre porovnanie je potrebné súbor rozdeliť do postupností bytov – binárnych sekvencií. Extrahované sú pomocou posuvného okna o veľkosti 16 bytov a posuvom 1 byte. To znamená, že výsledkom sú sekvencie s dĺžkou 16 bytov z každej pozície v binárnom súbore.
- **Sémantická podobnosť** pracuje s kódom binárnych súborov spracovaných pomocou disassemblingu. To vyžaduje podporu inštrukčnej sady cieľovej architektúry súboru. Zo vstupného súboru sú extrahované informácie o základných blokoch (*basic blokoch*) funkcií, teda častiach programu oddelených inštrukciami skoku či návratu z funkcie. Na basic blokoch sa potom pri ich porovnávaní vykonáva normalizácia, ktorá zabezpečí ignorovanie `NOP` inštrukcií a porovnanie nezávisle na konkrétnych operandoch ako sú registre, ofsety a adresy. Po normalizácii sa z basic blokov vypočíta ich haš, ktorý sa porovnáva pri vyhodnotení ich podobnosti.
- **Štrukturálna podobnosť** kombinuje prvky syntaktickej a sémantickej podobnosti. Vstupné súbory musia byť tiež spracované pomocou disassemblingu, no predmetom porovnania nie sú basic bloky, ale funkcie programu. Funkcie sú reprezentované pomocou CFG[20], kde jednotlivé uzly grafu predstavujú hodnoty hašu extrahovaných basic blokov. Pomer zhodných basic blokov potom určuje podobnosť funkcie.

Výsledkom je, že YaraZilla odlišuje tri úrovne abstrakcie binárnych dát: sekvencie, basic bloky a funkcie.

Spracovanie dát

Pred inšpekciou aj extrakciou vstupné vzorky prechádzajú predspracovaním, využívajúcim viacero metód, ktoré je možné individuálne vypnúť. Medzi použité metódy patria:

[19] API (Application Programming Interface) – rozhranie pre programovanie aplikácií
[20] CFG (Control Flow Graph) – grafová reprezentácia ciest v kóde

- filtrácia staticky linkovaného kódu, na ktorého detekciu sa využíva F.L.I.R.T.[21], čo je technológia vyvinutá pre nástroj IDA Pro. Využíva tzv. *signatúry*, ktoré popisujú funkcie pochádzajúce z určitej knižnice v podobe binárnych dát, aby ich bolo možné efektívne detegovať.
- filtrácia kódu označeného ako neškodlivý (*clean*) v databáze YaraZilly. Kód môže takto označiť aj používateľ pri inšpekcii alebo extrakcii.
- filtrácia kódu označeného ako neškodlivý (*clean*) vlastnou službou používanou v spoločnosti Avast, ktorá pracuje s veľkými blokmi bytov vzoriek. Kvôli svojej rýchlosti je použitá vo fázi predspracovania.

Odfiltrovaný kód je potom pri analýze nástrojom YaraZilla ignorovaný.

Po extrakcii je možné extrahované dáta dodatočne prefiltrovať pomocou presnejšej metódy filtrovania, znova vlastnou službou spoločnosti Avast. Služba využíva rozsiahlu databázu obsahujúcu binárne dáta čistých súborov. Je ale časovo náročná, preto je tento krok voliteľný a používa sa až po extrakcii.

Referenčná databáza

Databáza YaraZilly obsahuje rodiny malvéru s príslušnými sekvenciami, basic blokmi a funkciami. Je napĺňaná ukladaním výsledkov extrakcie do existujúcej rodiny či vytvorením nových rodín. Tento proces je možné automatizovať spracovávaním dostupných zbierok vzoriek, kedy sú zaradené do rodín podľa klasifikácie zdrojových dát a pri extrakcii i uložení sú zvolené rovnaké parametre. Ako zdroj vzoriek pre referenčnú databázu slúži najmä Malpedia[22] a zbierka vzoriek zachytených spoločnosťou Avast.

Použitie nástroja

Pri inšpekcii je možné nahrať vzorku v podobe lokálneho súboru alebo zadať jej SHA-256[23] haš. Zadaná vzorka je porovnávaná s kmeňmi v databáze, a je vygenerovaná správa s výsledkami podobnosti. Ukážka takejto správy z webového rozhrania nástroja je na obrázku 2.4.

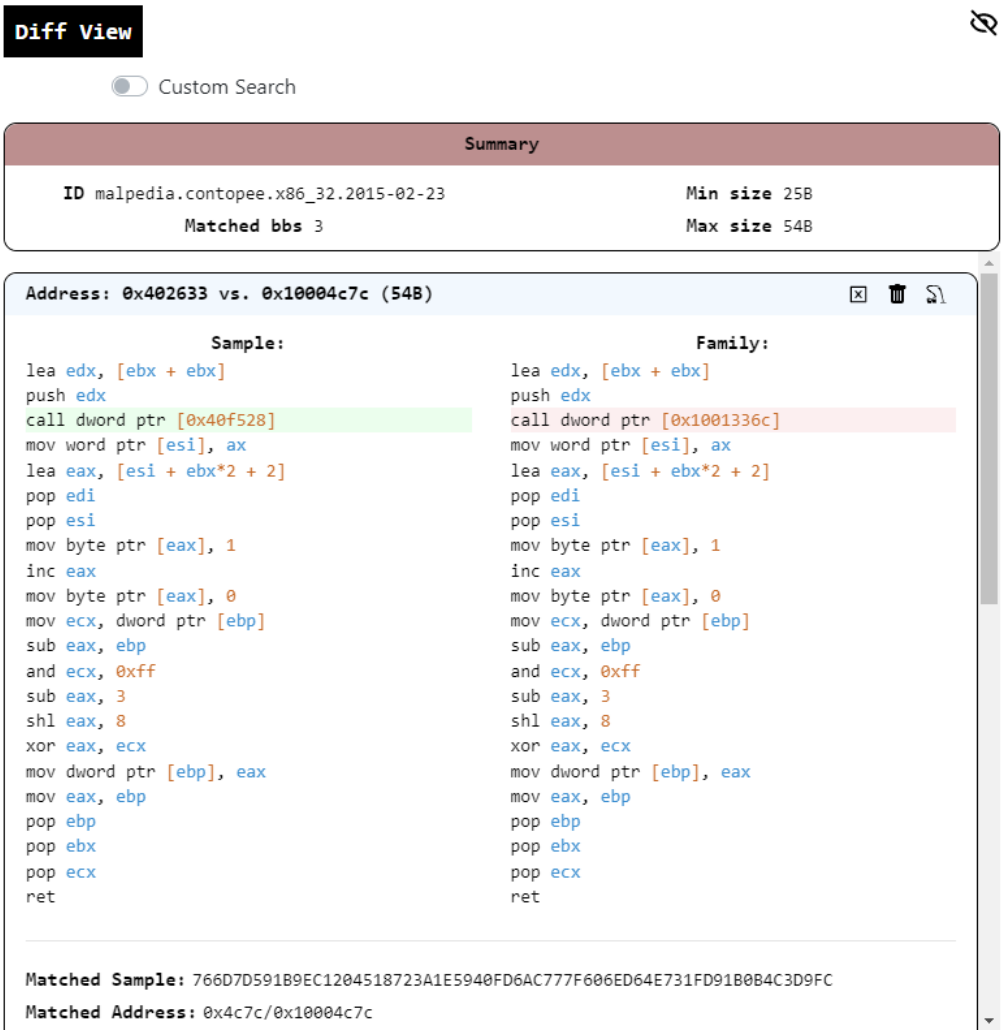
Sequences Similarity

Name	Version	Origin	Architecture	Matched/Total	Similarity	Sequences
wannacryptor	2017-02-09	malpedia	x86_32	3534 / 3534	100%	show
wannacryptor	2017-03-19	malpedia	x86_32	51 / 26372	0%	show
w32times	2007-09-12	malpedia	x86_32	7 / 4177	0%	show
bravonc	2017-04-28	malpedia	x86_32	16 / 18560	0%	show
romeos	2014-07-07-alfa	malpedia	x86_32	4 / 8609	0%	show
romeos	2013-10-11-golf	malpedia	x86_32	4 / 8977	0%	show
Mirage	loader_known_sequences	zoo	x86_32	6 / 17320	0%	show

Obr. 2.4: Ukážka výsledku inšpekcie v nástroji YaraZilla

[21] F.L.I.R.T. – https://hex-rays.com/products/ida/tech/flirt/
[22] Malpedia – https://malpedia.caad.fkie.fraunhofer.de/
[23] SHA-256 – kryptografická hašovacia funkcia s výstupom o veľkosti 256 bitov

V správe na obrázku 2.4 je vidieť zoznam rodín, u ktorých bola nájdená zhoda so skúmanou vzorkou, počet zhôd a percentuálna podobnosť s danými rodinami. Obrázok ukazuje len časť správy, kde sú vidieť len podobnosti zo sekvencií, no rovnako sú zobrazené aj podobnosti basic blokov a funkcií. Percentuálna podobnosť s určitou rodinou je vyjadrená z pomeru zhôd zo vzorky voči celkovému počtu sekvencií danej rodiny v databáze. Možnosť „show“ slúži na zobrazenie konkrétnych zhôd s príslušnou rodinou. V prípade sekvencií je to zoznam hexadecimálnych reťazcov predstavujúcich binárnu postupnosť dát. V prípade basic blokov je zobrazený zoznam blokov s ich kódom v assembly.

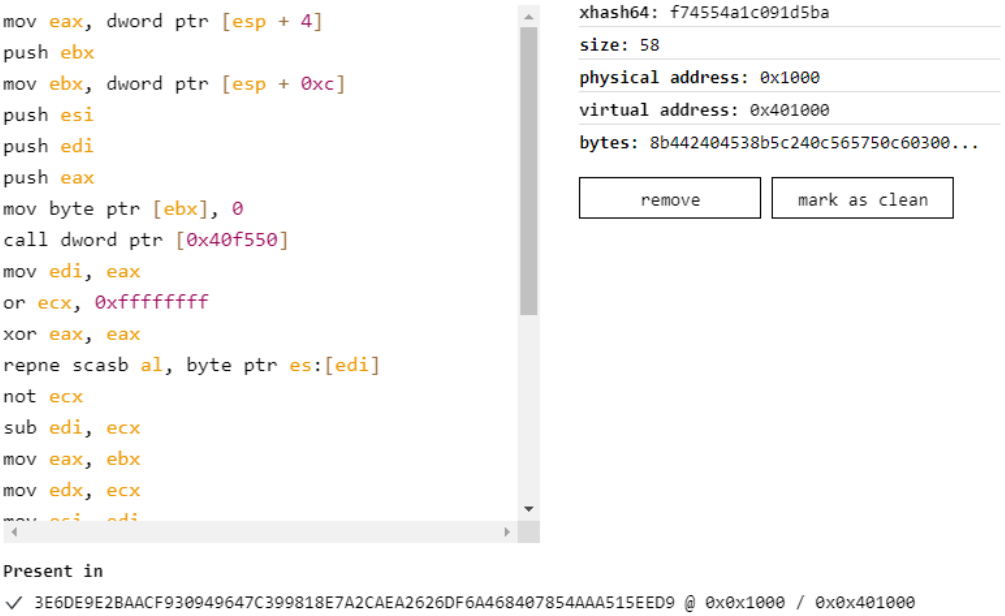


Obr. 2.5: Ukážka zobrazenia podobností basic blokov v nástroji YaraZilla

Na obrázku 2.5 je okrem kódu basic blokov uvedený SHA-256 haš vzorky, kde bola zhoda nájdená a adresy v hexadecimálnej podobe, ktoré indikujú, kde vo vzorke sa nachádza daný kód. Adresy sú užitočné najmä pri hlbšom skúmaní prekryvov kódu. Rovnakým spôsobom ako kód basic blokov je zobrazený kód funkcií.

Pri extrakcii je možné znova zadať vzorky v podobe lokálneho súboru alebo SHA-256 hašu. Pred extrakciou je možné upraviť jej parametre predspracovania. Po extrakcii je výstupom report, ktorý ukazuje informácie o získaných dátach. Extrahovaný kód je možné prefiltrovať na základe jeho výskytu v rámci vzoriek, ktoré boli zadané pri extrakcii.

Extrahované basic bloky a funkcie je možné prezerať, odstraňovať ich z výsledku, dodatočne prefiltrovať alebo označiť ako neškodlivý kód. Výsledok je možné z tohto rozhrania uložiť do databázy vytvorením novej rodiny alebo rozšírením existujúcej. Obrázok 2.6 ukazuje stránku s disassembly basic bloku pri inšpekcii.



Obr. 2.6: Ukážka zobrazenia basic bloku pri extrakcii v nástroji YaraZilla

2.3.4 Komerčné nástroje

Okrem vyššie spomenutých nástrojov existuje viacero komerčných nástrojov špecializovaných na analýzu malvéru, ktoré ponúkajú v rôznych formách aj detekciu podobnosti. Detailnejšie informácie o ich fungovaní však bežne nie sú verejne dostupné. Patria k nim aj:

- Kaspersky Threat Attribution Engine (KTAE)[24] je nástroj od spoločnosti Kaspersky Lab, ktorý pracuje na úrovni binárnych sekvencií s dĺžkou 16 bytov, ktoré nazýva *genómy*. Tieto *genómy* sú pri extrakcii prefiltrované. Nepoužíva disassembly a nepracuje na vyššej úrovni abstrakcie. To mu dovoľuje spracovať súbory určené na ľubovoľnú architektúru.
- File similarity search[25] od spoločnosti VirusTotal využíva na zistenie podobnosti hodnoty VHash, čo je hodnota vypočítaná vlastným zhlukovacím algoritmom, ktorá je rovnaká pre podobné súbory. Bližšie informácie o týchto nástrojoch nie sú verejne dostupné.
- Intezer Analyze[26] je pokročilý nástroj od spoločnosti Intezer, ktorý vyčísľuje podobnosť rodín aj na základe zhodných tzv. *génov* kódu. Takýto pomenovaním označuje bloky kódu získané pomocou disassembly. Vykonáva dynamické spúšťanie vzoriek v sandboxe, odkiaľ potom analyzuje kód v pamäti. Toto mu umožňuje skúmanie packed vzoriek. Zároveň využíva aj statickú extrakciu kódu.

[24] KTAE – https://www.kaspersky.com/enterprise-security/cyber-attack-attribution-tool
[25] File similarity search – https://support.virustotal.com/hc/en-us/articles/360001386937-File-similarity-search
[26] Intezer Analyze – https://analyze.intezer.com/

Kapitola 3

Analýza podobnosti kódu v kmeňoch malvéru

Cieľom tejto kapitoly bolo nájdenie vzťahov kódu v kmeňoch malvéru pomocou hromadnej analýzy ich podobnosti, aby mohli byť vybrané z nich podrobne preskúmané. Slúži teda ako podklad pre hĺbkovú analýzu v kapitole 4, pričom prináša potrebné experimentálne výsledky. Odhaliť vzťahy kódu umožňujú nástroje spomínané v podsekcii 2.3. Potrebné je zhodnotenie ich vlastností, aby boli vhodne používané podľa potrieb analýzy. Pri tvorbe tejto práce boli použité nástroje YaraZilla a BinDiff. Ako zdroj vzoriek malvéru slúžila Malpedia[1], vx-underground[2] a zbierka vzoriek zachytených spoločnosťou Avast.

Kapitola pozostáva z jednotlivých štúdií. Prvotne sú analyzované známe vzťahy kódu na preverenie schopností použitých nástrojov. Bolo tak nadviazané na predchádzajúci výskum kmeňov [16], kedy sa pozorovalo, či budú medzi nimi odhalené podobné vzťahy. V ďalšom kroku sa preskúmala efektivita nástrojov YaraZilla a BinDiff pri detekcii konkrétneho prekryvu malej časti kódu a dát vzoriek dvoch kmeňov. Analýza pokračovala zisťovaním vzťahov medzi viacerými verziami jednej rodiny, so zámerom preskúmania ich vývoja. S cieľom odhalenia doposiaľ neznámych vzťahov kódu boli vytvorené páry kmeňov zahrnutých v databáze nástroja YaraZilla na základe ich zistenej podobnosti.

3.1 Výber nástroja

Pre hromadné zisťovanie podobností na účely tejto prace bol zvolený nastroj YaraZilla. Je to škálovateľná služba, ktorá umožňuje porovnanie vzoriek voči celým rodinám, a to všetkým naraz, čím sa časová aj priestorová náročnosť pre výskumníka výrazne znižuje. Okrem náročnosti sa zároveň zjednodušuje interpretácia výsledkov, pretože podobnosť je vyhodnotená pre celú rodinu, ale zároveň, ak je to potrebné, pri basic blokoch a funkciách sa dá zistiť, z ktorej vzorky pochádzajú. Databáza tejto služby je naplnená vzorkami zo stránky Malpedia[3] či zbierky vzoriek zachytených spoločnosťou Avast, a tak je v službe zastúpených množstvo rodín malvéru, s ktorými dokáže tento nástroj porovnávať podobnosti.

Toto ale zároveň prináša svoje nevýhody, lebo nemôžeme špecifikovať, s ktorými konkrétnymi vzorkami z rodín budú porovnávané. Výsledky sú teda ovplyvnené tým, čo obsahuje databáza – či sú vzorky správne zaradené, či z nich boli správne extrahované sekvencie,

[1] Malpedia – https://malpedia.caad.fkie.fraunhofer.de/
[2] vx-underground – https://www.vx-underground.org/
[3] Malpedia – https://malpedia.caad.fkie.fraunhofer.de/

basic bloky a funkcie, či boli extrahované z unpacked vzoriek, či filtrácia prebehla korektne a podobne. YaraZilla zároveň pracuje nezávisle na troch rôznych úrovniach abstrakcie, čo môže poskytnúť odlišné pohľady na podobnosť skúmaných rodín, a je teda potrebné ich vhodne interpretovať v kontexte daného výskumu.

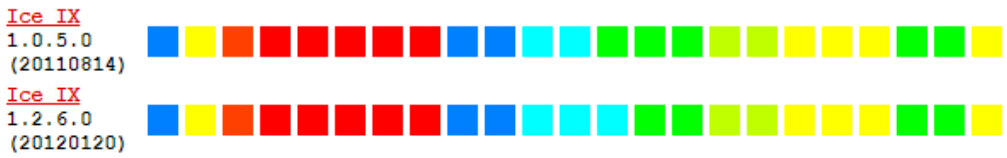
Voľne dostupné nástroje BinDiff a Diaphora sú určené na zisťovanie podobnosti dvoch binárnych súborov zanalyzovaných nástrojom IDA Pro. Pre ich porovnanie je teda potrebné spracovať v IDA Pro obe skúmané vzorky, čím sa vytvoria databázové súbory, s ktorými tieto nástroje pracujú. Táto skutočnosť je dôvodom, prečo tieto nástroje nie sú vhodné pre hromadnú analýzu rodín v rámci tejto práce. Pri spracovaní vzoriek síce môže byť práca s grafickým rozhraním vynechaná a proces automatizovaný, no časová náročnosť zostáva relatívne vysoká. Je to tak pretože všetky vzorky musia byť navzájom porovnávané po dvojiciach. Pri množstve vzoriek čo i len 100 rodín by tento proces pravdepodobne vyžadoval isté distribuovanie výpočtu a úložného priestoru. Alternatívou by bolo zredukovanie počtu analyzovaných vzoriek vhodným výberom najdôležitejších verzií z každej rodiny, no v tomto prípade by pravdepodobne nastala určitá strata informácií, ktoré by sa mohli ukázať ako kľúčové.

Z porovnania je pre nasledujúcu hromadnú analýzu najvhodnejší nástroj YaraZilla, najmä pretože dokáže skúmať celé rodiny malvéru naraz a ponúka široký rozsah rodín vo svojej databáze. Pre vybrané vzťahy ju dopĺňa nástroj BinDiff, ktorý bol použitý na overenie dosiahnutých výsledkov či vizualizáciu prekryvov kódu.

3.2 Štúdia A – Analýza kmeňov typu banker

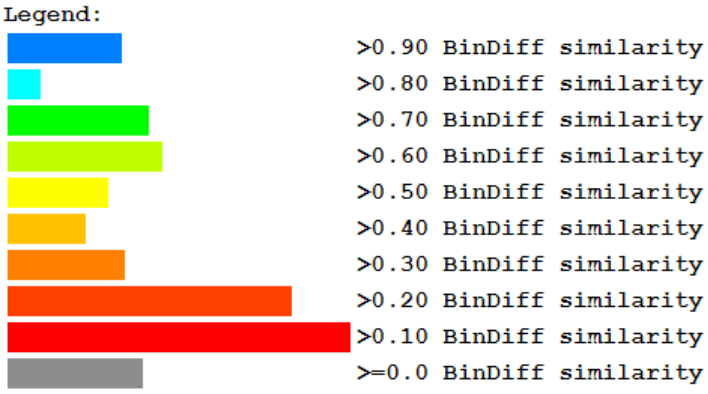
Na prvotnú analýzu s nástrojom YaraZilla boli vybrané vzorky použité v článku [16], ktorý je uvedený aj v zadaní. Vzorky pochádzajú zo stránky Malpedia[4] a patria do rodín podobných rodine Zeus. Radia sa pod typ *banker*, spomínaný v podsekcii 2.1.1, a patria k najznámejším trójskym koňom tohto druhu. Z rodiny Zeus sa po zverejnení zdrojového kódu vyvinulo množstvo nových rodín a ich variantov, ktorých vývoj sa odráža v ich vzájomnej podobnosti kódu.

Článok uvádza podobnosti medzi jednotlivými vzorkami vo forme farebne odlíšených buniek tabuľky. Na získanie týchto výsledkov bol použitý nástroj BinDiff, spomenutý podsekcii 2.3.1. Touto analýzou teda môžeme otestovať nástroj YaraZilla v praxi a zhodnotiť získané výsledky v porovnaní s uvedeným článkom.



Obr. 3.1: Časť výsledkov analýzy z referenčnej stránky [16]

[4]Malpedia – https://malpedia.caad.fkie.fraunhofer.de/



Obr. 3.2: Legenda k zafarbeniu buniek tabuľky na referenčnej stránke [16]

Obrázok 3.1 predstavuje časť tabuľky zo stránky s podobnosťami skúmaných rodín a ich verzií podľa nástroja BinDiff. Kompletná tabuľka má 98 riadkov a 98 stĺpcov vizualizujúcich percentuálne hodnoty vzájomných podobností všetkých 98 vzoriek. Tieto hodnoty sú neskôr porovnané s výsledkami analýzy nástrojom YaraZilla. Na obrázku sú riadky hodnôt pre dve verzie rodiny `Ice IX`, vyvinutej z rodiny `Zeus`. Počet stĺpcov, predstavujúcich vzorky, s ktorými sú verzie v riadkoch porovnávané, je zredukovaný pre potreby ukážky. Na obrázku 3.2 je legenda vysvetľujúca farebné odtiene buniek tabuľky, ktoré reprezentujú rozsahy získaných percentuálnych hodnôt po desiatkach percent. Veľkosť horizontálne uložených stĺpcov tu znázorňuje zastúpenie daného rozsahu podobnosti vo výsledkoch.

3.2.1 Priebeh analýzy

Dokopy bolo analyzovaných 98 rôznych verzií malvéru, z nich mnohé boli dostupné v podobe viacerých vzoriek. Z týchto vzoriek boli vybrané tie, ktoré boli označené ako „unpacked“, prípadne „dumped“, tak, aby každá verzia bola zastúpená práve jednou vzorkou. Označenie „dumped“ alebo „unpacked“, znamená, že sa jedná o „rozbalenú“ vzorku, vytvorenú výskumníkom malvéru z pôvodnej vzorky. Bližšie vysvetlenie tohoto procesu sa nachádza v podsekcii 2.2.3.

Analýza pozostávala z nasledovných krokov:

1. získanie vzájomných podobností všetkých vzoriek pomocou nástroja BinDiff,
2. zoskupenie získaných podobností do jednej sady dát,
3. získanie podobnosti každej analyzovanej vzorky s rodinami v YaraZille,
4. zoskupenie získaných podobností podľa YaraZilly pre každú vzorku do troch sád dát:
   - podobnosti vzoriek s rodinami na úrovni sekvencií,
   - podobnosti vzoriek s rodinami na úrovni basic blokov,
   - podobnosti vzoriek s rodinami na úrovni funkcií,
5. generovanie tabuliek z výsledných 4 zdrojov dát.

Pri generovaní tabuľky z údajov podľa nástroja BinDiff boli vzájomné podobnosti jednotlivých verzií rodín spriemerované pre každú rodinu, aby sa prezentovaný formát údajov priblížil tým, ktoré boli získané nástrojom YaraZilla. BinDiff totiž porovnával navzájom

jednotlivé vzorky, YaraZilla zas primárne určuje podobnosti vzoriek s celými rodinami – vzorky sa nedajú porovnať navzájom.

3.2.2 Výsledky

YaraZilla odhalila mnoho vzťahov uvedených na referenčnej stránke [16], no všeobecne boli percentuálne hodnoty podobností nižšie ako tie získané nástrojom BinDiff. Výsledky analýzy sú vizualizované prostredníctvom tabuliek v prílohe B. Zdrojové dáta pre generovanie týchto tabuliek sú obsiahnuté na priloženom médiu. Tabuľky B.2, B.3 a B.4 zobrazujú priamo hodnoty získané nástrojom YaraZilla, v tabuľke B.1 boli však podobnosti podľa nástroja BinDiff spriemerované pre rodiny, ako už bolo spomínané, a výsledkom boli vo všetkých tabuľkách riadky s verziami rodín a stĺpce s rodinami.

V tabuľkách výsledkov YaraZilly si môžeme všimnúť, že niektorým rodinám v stĺpcoch chýbajú hodnoty, čo je znázornené šedou farbou. Konkrétne sa jedná o `GameoverP2P`, `Zeus MailSniffer` a `Chthonic`. Je to preto, lebo tieto rodiny chýbajú v databáze YaraZilly. Ďalším pozorovaním je, že aj pri odhalených vzťahoch YaraZilla často zaznamenala menšie hodnoty podobností, no toto môže byť spôsobené hlavne skutočnosťou, že porovnáva vzorky voči celým rodinám.

Kvôli rozdielnej metóde určenia podobnosti aj chýbajúcim rodinám nemôžeme celkovo porovnať výsledky získané použitými nástrojmi, ako špecifický príklad preto poslúži jedna zo vzoriek. Tabuľka 3.1 slúži na ilustráciu, aké výsledky boli dosiahnuté pri mnohých vzorkách. Obsahuje hodnoty podobnosti jednej vzorky rodiny `Ice IX` verzie `1.0.5.0 (20110814)` so skúmanými rodinami.

Podobná rodina	BinDiff	YaraZilla		
		Sekvencie	Basic bloky	Funkcie
`Zeus`	98,0%	78,6%	92,5%	91,7%
`Murofet`	32,6%	0,1%	0,3%	0,2%
`GameoverP2P`	15,8%	-	-	-
`GameoverDGA`	17,7%	0,1%	0,0%	0,0%
`Ice IX`	99,0%	100,0%	100,0%	100,0%
`Citadel`	69,6%	61,6%	79,5%	80,0%
`KINS`	68,8%	37,7%	48,8%	40,3%
`VM Zeus`	64,5%	37,9%	65,4%	53,8%
`PandaBanker`	25,1%	0,1%	0,0%	0,0%
`Zeus MailSniffer`	68,6%	-	-	-
`Floki Bot`	72,6%	35,8%	18,2%	10,0%
`Zeus Sphinx`	9,0%	0,3%	0,0%	0,0%
`Chthonic`	7,2%	-	-	-

Tabuľka 3.1: Podobnosti vzorky `Ice IX 1.0.5.0 (20110814)` so skúmanými rodinami, zoradenými rovnako ako stĺpce v tabuľkách prílohy B a na referenčnej stránke [16]

Ako je vidieť v tabuľke, nástroj YaraZilla dokázal odhaliť mnoho zásadných podobností, ale zároveň pri hodnotách nižších ako 50% už výrazne zaostával za nástrojom BinDiff. A to v takej miere, že nemohla byť spôsobená tým, že sa porovnávajú vzorky voči celým rodinám. Tieto hodnoty totiž veľmi závisia od zastúpenia danej rodiny v databáze tohto nástroja. Napríklad podobnosť s rodinou `Murofet` bola odhalená, ale na tak nízkej úrovni, že dosaho-

vala desatiny percenta. Po bližšom preskúmaní bolo odhalené, že verzia `Murofet 0.0.0.27 (20120120)` dosahovala podľa BinDiff podobnosť 55,8%, ostatné iba 24,2% a 17,5%. Práve táto vzorka, ktorá sa zdá byť značne odlišná od ostatných dvoch skúmaných, nebola v databáze YaraZilly.

3.3 Štúdia B – Porovnanie vzoriek kmeňov WannaCry a Contopee

Okrem vyhodnotenia percentuálnej podobnosti skúmanej vzorky dokážu nástroje YaraZilla a BinDiff identifikovať konkrétne prekryvy v kóde, ktoré môžeme ďalej podrobne skúmať. Táto časť preto má za cieľ preveriť schopnosti týchto nástrojov detegovať známe zhody v malej časti kódu či dát.

V roku 2017 sa objavila informácia naznačujúca, že medzi rodinami WannaCry a Contopee by mohol byť určitý vzťah [6], ktorý by útoky rodiny WannaCry pripísal malvérovej skupine Lazarus zo Severnej Kórey, údajne riadenej týmto štátom. Diskusie na túto tému odštartoval príspevok[5] výskumníka Googlu na sociálnej sieti Twitter, ktorý obsahoval iba dva MD5 haše identifikujúce vzorky malvéru a dve adresy v každej z nich.

Rodina WannaCry druhu ransomware zaznamenala najväčšie šírenie v máji 2017 [1]. Pri útoku využívala zraniteľnosť operačného systému Windows nazývanú „EternalBlue“ a šírila sa pomocou počítačových sietí podobne ako malvér druhu červ. Jednalo sa o masový útok, ako už bolo spomínané v časti 2.1.1, v priebehu jedného dňa bolo zasiahnutých viac ako 230 000 zariadení po celom svete. Situáciu zhoršoval fakt, že zašifrované súbory nebolo možné obnoviť, kvôli čomu býva tento malvér označovaný aj ako *wiper*. Útočníci napriek tomu požadovali od obetí po zašifrovaní poplatok, ktorý sa s postupom času zvyšoval. Avšak keď bolo známe, že súbory neboli dešifrované ani po zaplatení, poškodené osoby nemali dôvod platiť. Hlavnou motiváciou útočníkov tak pravdepodobne neboli peniaze. Táto skutočnosť a spôsobený rozsah škôd, ktorý bol dovtedy nevídaným úkazom, teda podporuje teóriu, že sa jednalo o útok za ktorým mohol stáť útočník s dostupnými prostriedkami na úrovni štátu.

Rodina Contopee patrí k druhu backdoor. Známy aj pod názvom WHITEOUT, po napadnutí komunikuje s útočníkom pomocou vlastného šifrovacieho protokolu [5]. Umožňuje manipuláciu so súbormi, prenos dát či vzdialené spúšťanie príkazov. Použitý bol napríklad pri útokoch na bankové inštitúcie v rokoch 2015 a 2016.

Uvedená vzorka WannaCry pochádza z februára 2017, jedná sa teda ranú verziu tejto rodiny. V nasledujúcich verziách sa spomínané vzťahy s Contopee neobjavili. Vzorka tejto rodiny bola zostavená vo februári 2015.

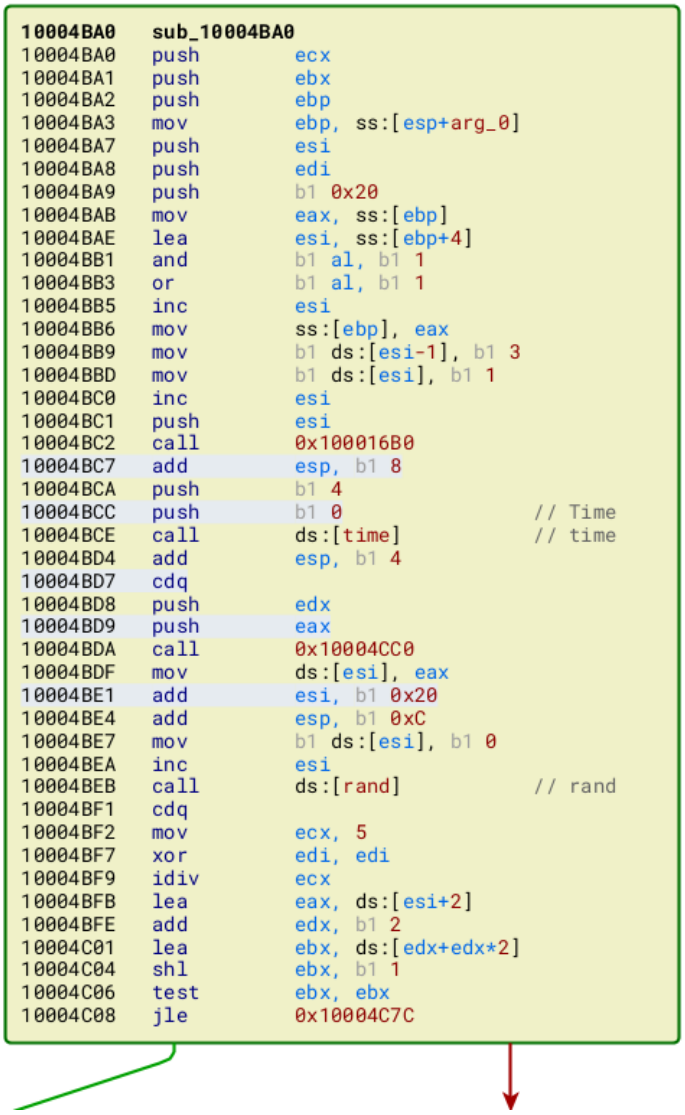
V spomínanom príspevku sú uvedené adresy v hexadecimálnej podobe, ktoré značia umiestnenie funkcií a dát v programe. Adresa `0x402560` (`0x10004ba0`) udáva pozíciu funkcie, ktorá pristupuje k dátam na adrese `0x40F598` (`0x10012AA4`). Tieto dáta sú zhodné a predstavujú šifrovacie sady, z ktorých funkcia náhodne vyberá pri *handshaku*[6] vlastnej implementácie šifrovacieho protokolu [5].

V YaraZille bola vykonaná inšpekcia vzorky WannaCry, ktorá bola porovnaná s rodinou Contopee v databáze, aj naopak, teda vzorky Contopee porovnanej s WannaCry. V oboch prípadoch YaraZilla na úrovni sekvencií ani funkcií nenašla žiadnu podobnosť, no našla 3 zhodné basic bloky, z toho 2 zo spomínanej spoločnej funkcie. Táto funkcia má 12 basic

[5] https://twitter.com/neelmehta/status/864164081116225536

[6] handshake – proces výmeny dát potrebných ku komunikácii pred jej samotným započatím

blokov, z ktorých bolo 9 odfiltrovaných pri extrakcii vzoriek rodiny kvôli príliš malej veľkosti. Nástroj BinDiff udáva podobnosť tejto funkcie na úrovni 99%, rozdiely vidí iba v prvom basic bloku. Tento basic blok na obrázku 3.3 obsahuje vo vzorke rodiny Contopee niekoľko pridaných inštrukcií.



Obr. 3.3: Odlišný basic blok vzorky Contopee, z funkcie spoločnej so vzorkou WannaCry

Na obrázku 3.3 sú modrou farbou vyznačené riadky s pridanými inštrukciami v basic bloku vzorky rodiny Contopee. Rozdiely sú spôsobené volaním odlišných funkcií, kde vo vzorke Contopee sa musia v basic bloku odstrániť parametre volanej funkcie zo zásobníka, narozdiel od vzorky WannaCry.

YaraZilla kvôli pridaným inštrukciám nevyhodnotí daný basic blok za podobný. Podobnosť funkcií sa určuje z ich basic blokov, a keďže bol tento basic blok vyhodnotený ako odlišný, funkcia je tiež. Spoločné sekvencie funkcie ani dát, ku ktorým pristupuje neboli odhalené z toho dôvodu, že rodinám v YaraZille chýbali. Po experimentovaní s rôznymi parametrami extrakcie bolo odhalené, že problémom bolo filtrovanie, konkrétne pri filtrovaní technológiou F.L.I.R.T., používanou IDA Pro aj nástrojom YaraZilla. Po vypnutí tejto

technológie boli sekvencie patriace spoločnej funkcii aj sekvencie dát zo vzoriek extrahované. Nástroj BinDiff zas pri zisťovaní podobnosti s dátami nepracuje, takže ich podobnosť nedetegoval.

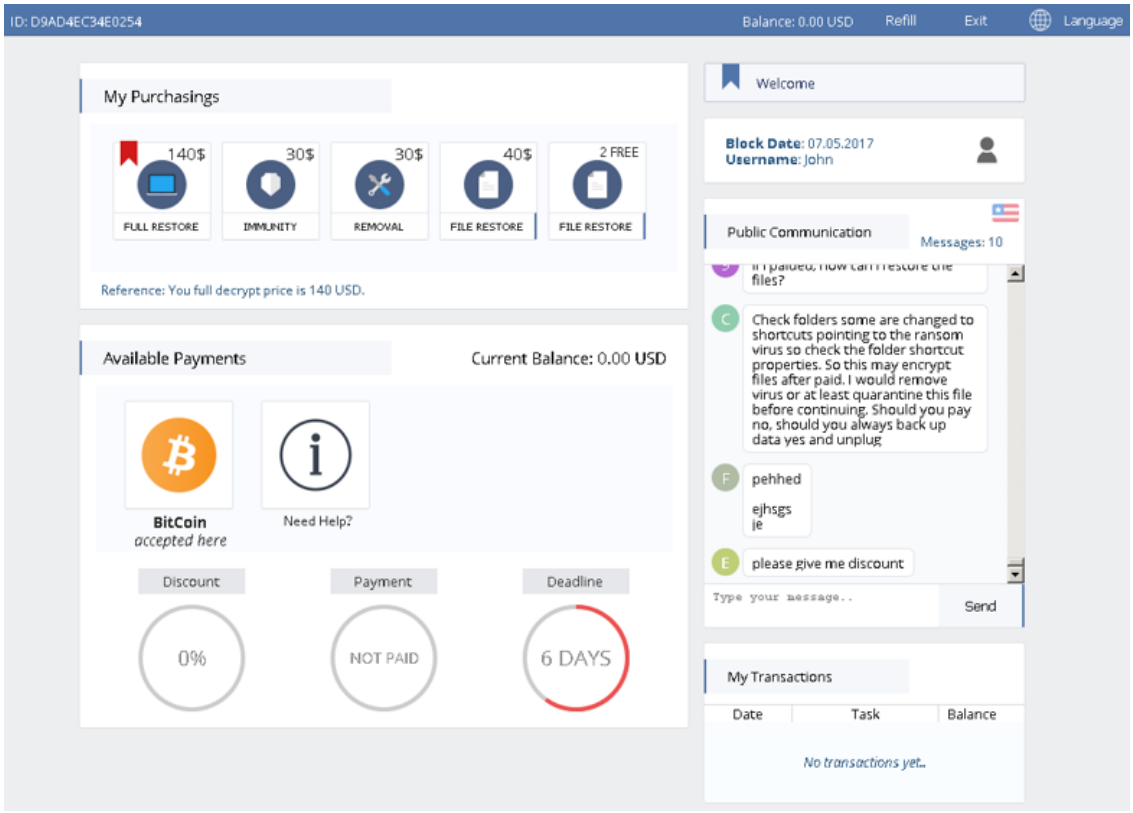
Záverom je, že tieto rodiny majú spoločnú iba jednu spomínanú funkciu a dáta, ku ktorým pristupuje. Môžeme konštatovať, že nástroj YaraZilla je náchylný nedetegovať podobnosti najmä pri zmene počtu inštrukcií čo i len v časti funkcie. Nástroj je tiež veľmi závislý od správneho fungovania použitých nástrojov pri plnení databázy, pretože prípade rodín WannaCry a Contopee chybne odfiltroval ich zhodné sekvencie patriace ako funkcii, tak aj dátam. Taktiež sa ukázalo, že skúmať vzťahy nielen v kóde malvéru, ale aj dátach môže byť užitočné.

3.4 Štúdia C – Analýza podobnosti verzií kmeňa Spora

Po analýze zameranej na väčšie množstva rodín bol zvolený iný prístup ku skúmaniu podobností medzi vzorkami malvéru – skúmanie vývoja verzií jednej rodiny. Po dohode s konzultantom zo spoločnosti Avast bola zvolená rodina Spora. Jedná sa o rodinu druhu ransomware, ktorá sa šírila v roku 2017. Analýza nadväzuje na existujúci výskum, pri ktorom boli popísané základné vlastnosti tejto rodiny [7], a identifikované jej verzie [8].

Veľké množstvo malvéru typu ransomware pochádza z Ruska, kedy sa často vyhýba útokom na domácom území a okolitých štátoch. Táto rodina je odlišná v tom, že sa začala šíriť práve v Rusku a bola zameraná hlavne na rusky hovoriace obete. K infekciám dochádzalo prostredníctvom škodlivých príloh e-mailov alebo odkazov, kedy bol prostredníctvom skriptov do zariadenia stiahnutý a spustený samotný ransomware. Popri šifrovaní súborov sa tento malvér šíri po sieťových diskoch pripojených k napadnutému zariadeniu, čo sa podobá vlastnosti červov. Jedná sa o rodinu ransomware s dômyselne implementovaným šifrovaním súborov, fungujúcim úplne bez nutnosti pripojenia k internetu. To je vyžadované až po zašifrovaní súborov, kedy sa obeť pomocou odkazu v *ransom note* môže dostať na stránku s platobným portálom.

Na platobnom portáli je obeti vypočítaná personalizovaná výška poplatku za odstránenie malvéru a dešifrovanie súborov, a zároveň sú ponúknuté aj lacnejšie možnosti, ako samotné odstránenie malvéru alebo dokonca imunizácia proti útoku. Možnosť kúpenia imunizácie – ochrany proti budúcej infekcii rovnakým malvérom – od tvorcov ransomware nie je vôbec bežným úkazom. Ukážka takejto stránky je na obrázku 3.4. Na stránke je tiež vidieť, že platby prebiehajú prostredníctvom kryptomeny Bitcoin, a dobíja sa nimi konto obete, z ktorého sú strhnuté poplatky za požadované služby. Na platbu má človek obmedzený čas, čo ho má priviesť k rozhodnutiu zaplatiť. Využitie kryptomien či časového limitu nie je pri kmeňoch ransomware nič nezvyčajné, no možnosti platobného portálu a jeho vizuálna prezentácia sú výnimočným úkazom.



Obr. 3.4: Ukážka platobnej stránky pre ransomware Spora z analýzy z mája 2017 [7]

3.4.1 Priebeh analýzy

Na analýzu bolo zvolených 18 verzií tejto rodiny, pričom každá bola reprezentovaná jednou vzorkou. Verzie boli identifikované dátumom zostavenia v hlavičke vzorky. Skúmané neboli vzorky, ktoré boli pôvodne zachytené, ale tie z nich „rozbalené“ pomocou metód *unpackingu* spomínaných v časti 2.2.3. Pre analýzu museli byť vzorky jednotlivých verzií extrahované a uložené v databáze YaraZilly podobne, akoby to boli samostatné rodiny. V praxi to umožnilo pracovať v YaraZille s podobnosťami jednotlivých vzoriek medzi sebou, narozdiel od analýzy bankerov v sekcii 3.2. Rovnako analýza pracovala iba s vybranou sadou vzoriek, nie s existujúcim obsahom databázy. Výsledky sa teda v tomto prípade dajú priamo porovnávať s nástrojom BinDiff.

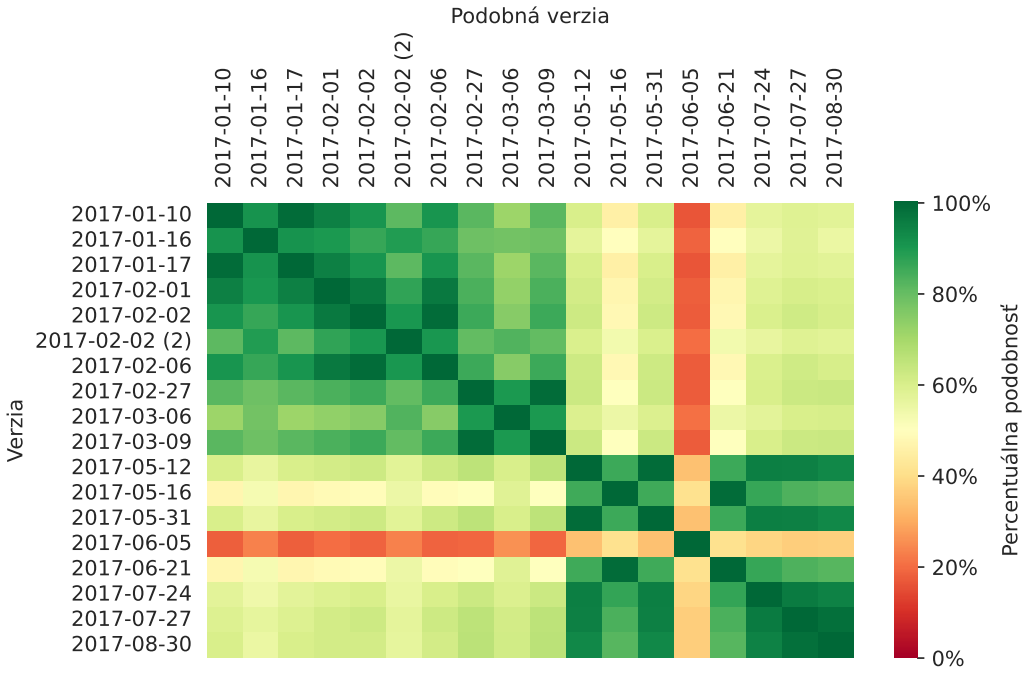
Analýza prebiehala nasledovne:

1. získanie vzájomných podobností všetkých vzoriek pomocou nástroja BinDiff,
2. zoskupenie získaných podobností do jednej sady dát,
3. extrakcia a uloženie dát zo vzoriek v YaraZille – do samostatných rodín,
4. získanie podobnosti všetkých vzoriek navzájom – inšpekciou v YaraZille,
5. zoskupenie získaných dát do troch sád podľa úrovní – sekvencie, basic bloky, funkcie,
6. generovanie 4 tabuliek z dát získaných nástrojmi BinDiff a YaraZilla.

Oproti analýze v sekcii 3.2 teda obsahovala dva kroky naviac – extrakciu a následné uloženie dát v YaraZille. Extrakcia prebiehala prostredníctvom grafického rozhrania YaraZilly, a inšpekcia pomocou HTTP požiadavkov na API.

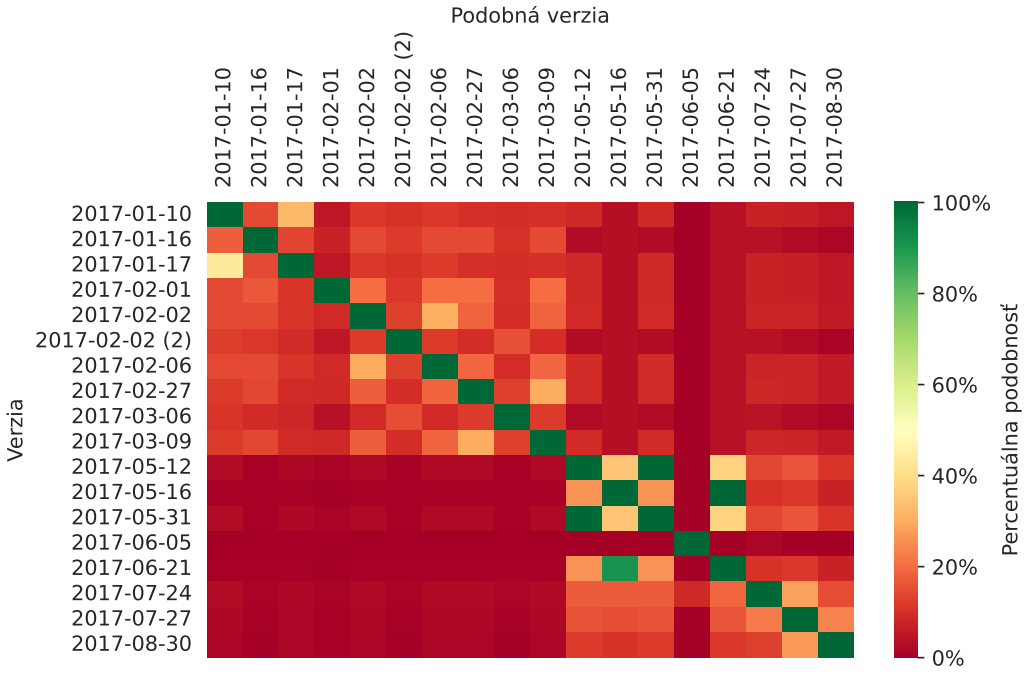
3.4.2 Výsledky

Získané výsledky, ktoré najlepšie zobrazuje tabuľka 3.5 priniesli tri hlavné zistenia. Prvým je skutočnosť, že verzie s dátumom skorším ako 2017-05-12 sú si navzájom viac podobné ako s ostatnými verziami, čo taktiež platí o verziách s neskorším dátumom. V tejto časti výskumu zatiaľ nebolo jasné, aké zmeny nastali vo verzii s týmto dátumom, no je zrejmé, že sa jednalo o druhý najväčší odklon od pôvodnej implementácie. Druhé zistenie je, že verzia 2017-06-05 obsahuje práve najväčšie rozdiely oproti ostatným verziám. Pri týchto rozdieloch sa núka otázka, či sa naozaj jedná o vzorku patriacu tejto rodine. Odpovede aj na túto otázku sa snaží nájsť hĺbková analýza verzií Spory v sekcii 4.1.



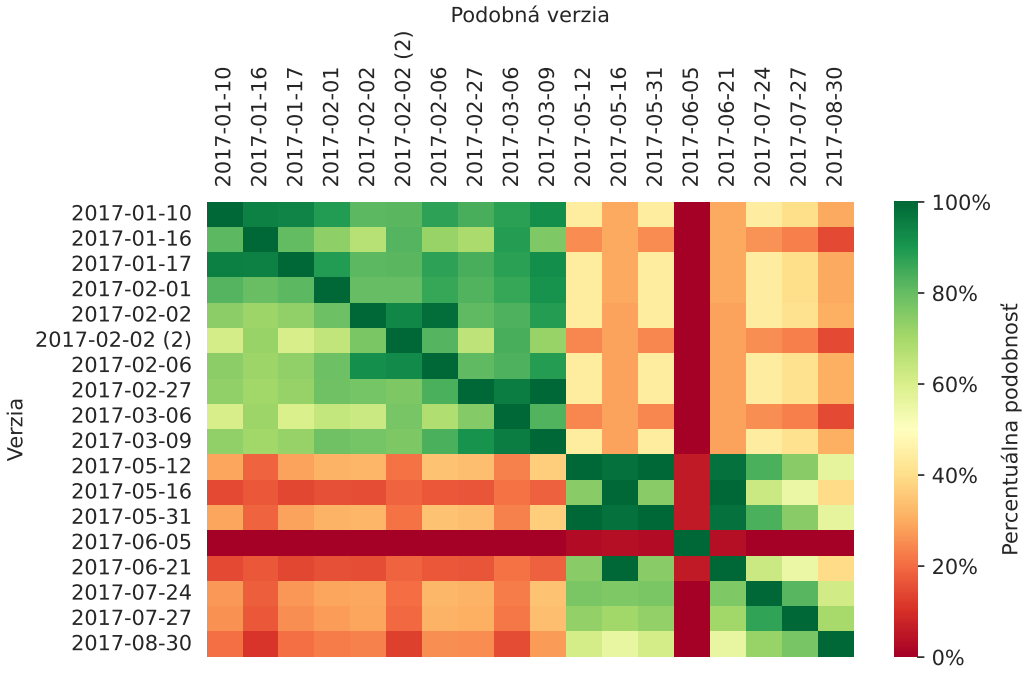
Obr. 3.5: Tabuľka podobnosti verzií podľa nástroja BinDiff

Ďalším hlavným zistením je, že nástroj YaraZilla dokázal odhaliť rovnaké vzťahy medzi verziami ako BinDiff, no často v nižšej miere. V tomto ohľade dopadla najhoršie analýza na úrovni sekvencií, ako je vidieť v tabuľke 3.6. Pri vizualizácii s lineárnou škálou zafarbenia mnoho podobností priam zaniká, pretože sa jedná o príliš nízke hodnoty.

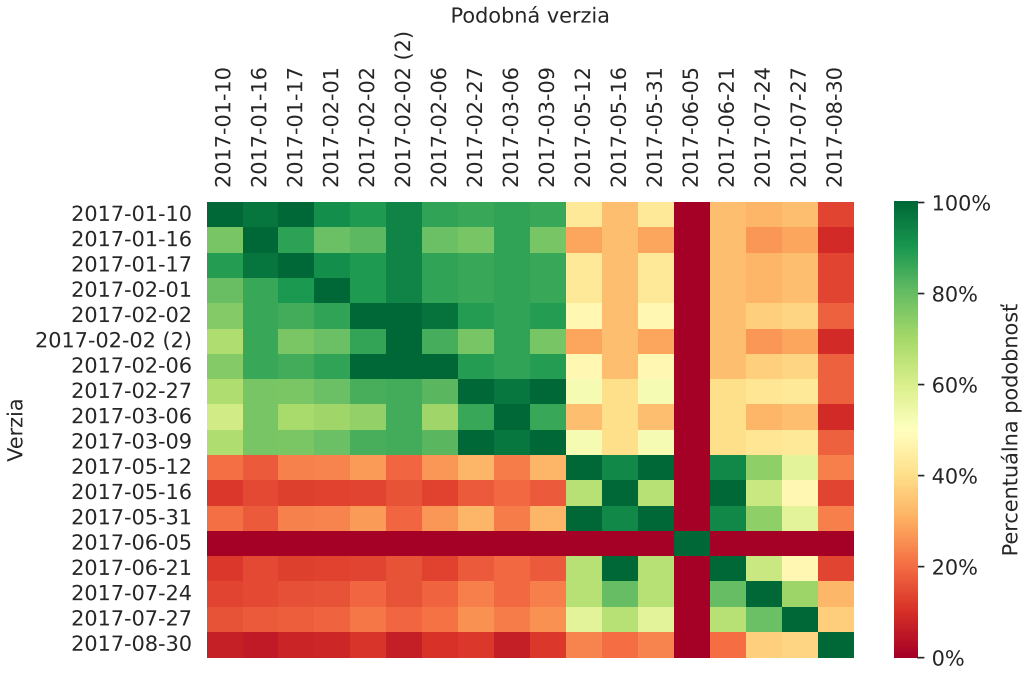


Obr. 3.6: Tabuľka podobnosti verzií na úrovni sekvencií podľa nástroja YaraZilla

Narozdiel od sekvencií sa analýza na úrovni basic blokov a funkcií menej líšila od podobností podľa nástroja BinDiff. Pri takomto použití mu môže YaraZilla konkurovať, čo je vidieť aj v tabuľkách 3.7 a 3.8. Rovnako je tu znova jasne vidieť zlom v podobnosti pri verzii 2017-05-12.



Obr. 3.7: Tabuľka podobnosti verzií na úrovni basic blokov podľa nástroja YaraZilla



Obr. 3.8: Tabuľka podobnosti verzií na úrovni funkcií podľa nástroja YaraZilla

Na zistenia tejto analýzy nadväzuje sekcia 4.1 v nasledujúcej kapitole, celá venovaná hĺbkovej analýze verzií tejto rodiny. Objasňuje funkcionalitu každej verzie a implementačné rozdiely medzi nimi, z ktorých vyplývajú namerané hodnoty podobnosti.

3.5 Štúdia D – Hľadanie doposiaľ neznámych vzťahov medzi kmeňmi

V nasledujúcom kroku slúžil nástroj YaraZilla na výber rodín pre bližšiu analýzu. Na tento účel sa dá využiť funkcionalita API, ktoré ponúka možnosť získania celkového počtu sekvencií ľubovoľnej rodiny a počtu ich prekryvov s inými rodinami. Týmto spôsobom boli zistené vzťahy medzi rodinami, z ktorých bol vytvorený zoznam párov podobných rodín.

Podobnosť na úrovni sekvencií je v tomto prípade vhodnou voľbou, pretože pri zisťovaní podobností na vyššej úrovni, teda s využitím disassembly, boli zaznamenané problémy v konzistentnosti dát. Tie boli spôsobené spôsobom implementácie ukladania basic blokov a funkcií v databáze. Podobnosti by sa teda museli zisťovať pre každú vzorku zvlášť.

3.5.1 Proces tvorby párov

Tvorba párov podobných rodín podľa spoločných sekvencií prebiehala nasledovne:

1. získanie zoznamu všetkých rodín,
2. pre každú rodinu s neprázdnym zoznamom prekryvov sekvencií:
   (a) získanie zoznamu rodín so spoločnými sekvenciami a ich počtom,
   (b) výpočet relatívnej podobnosti – voči pôvodnej rodine a podobnej rodine zvlášť – z počtu spoločných sekvencií a všetkých sekvencií rodiny,

3. vytvorenie množiny párov zo zoznamu rodín a ich prekryvov sekvencií,
4. zoradenie párov podľa súčtu podobností,
5. exportovanie párov do tabuľky.

V priebehu tohto procesu boli zhromaždené detaily o 614 rôznych rodinách, z ktorých bolo vytvorených 307 párov, v rámci ktorých boli odhalené prekryvy sekvencií. Na automatizáciu bol vytvorený skript v jazyku Python s využitím knižnice pandas[7]. Získavanie zoznamu rodín a ich štatistík o sekvenciách prebiehalo pomocou HTTP požiadavkov na API. Odpovede boli ukladané, zhromažďované a manipulované s pomocou dátového typu slovník (dict), ktorý korešponduje odpovedi vo formáte JSON[8]. Nakoniec boli páry exportované do CSV[9] súboru pomocou knižnice pandas. V takomto formáte môžu byť výsledky jednoducho vizualizované vo forme tabuľky.

3.5.2 Výsledky

V čase výskumu tvorilo obsah databázy YaraZilla 614 rodín. Odhalením vzájomných vzťahov ich sekvencií vzniklo celkovo 307 párov rodín. Z výsledkov analýzy prvotne zaujali podobnosti sekvencií na úrovni blízkej 100%. Takéto hodnoty môžu okrem skutočnej identickosti značiť nesprávne zaradenie vzoriek k rodine či už len v databáze YaraZilly alebo iných nástrojov Avastu, prípadne chybnú extrakciu sekvencií. Najväčšia časť párov sa ale zhromaždila pri hodnotách blízkych nule, čo bolo očakávané vzhľadom na množstvo odlišných rodín.

Veľkú časť dvojíc s vysokou podobnosťou tvorili duplicity, teda páry tých istých rodín uložených viackrát pod iným názvom. Kvôli duplicitám bola tvorba párov tiež viac komplikovaná ako sa uvádza v podsekcii 3.5.1, bolo totiž potrebné vytvoriť rôzne zoznamy rodín, jeden s pôvodnými názvami rodín a druhý so zlúčenými duplicitami. Následne boli aj získané prekryvy sekvencií rodín zlučované.

[7] pandas – https://pandas.pydata.org/
[8] JSON (JavaScript Object Notation) – formát výmeny dát, používaný najmä vo webových aplikáciách
[9] CSV (comma-separated values) – formát textového súboru, používaný v tabuľkových procesoroch

Rodina	Podobná rodina	Podobnosť	
		voči pôvodnej rodine	voči podobnej rodine
atmappin	atmii	100,0%	100,0%
babyshark	grease	100,0%	100,0%
kraton	padodor	100,0%	100,0%
dropshot	stonedrill	100,0%	99,5%
bitconnect	bansomqare	99,7%	99,7%
bitconnect	opsvenezuela	99,6%	99,6%
bansomqare	opsvenezuela	99,6%	99,6%
warzone	ave maria	100,0%	79,6%
ice ix	zeus	87,8%	72,4%
dropshot	turnedup	84,4%	49,9%
stonedrill	turnedup	83,9%	49,9%
ice ix	citadel	68,1%	59,3%
zeus	citadel	57,9%	61,2%
...			

Tabuľka 3.2: Ukážka párov rodín na základe percentuálnej podobnosti sekvencií

V tabuľke 3.2 sa nachádza ukážka vytvorených párov zoradených podľa podobnosti. Vzhľadom na stovky rodín uložených v databáze YaraZilly obsahuje tabuľka len malú časť získaných výsledkov. Kompletné výsledky sú súčasťou priloženého média. Vytvorené páry slúžili ako zdroj pre výber vzoriek na hĺbkovú analýzu v kapitole 4. Objavujú sa v nej aj rodiny, ktorým sa venuje sekcia 3.2. Pozoruhodná bola podobnosť rodín Padodor a Kraton na úrovni 100%, pretože tieto boli v rámci Avastu považované za rozdielne rodiny. Hĺbkovej analýze vzoriek týchto rodín sa venuje sekcia 4.2.

3.5.3 Výsledky po aktualizácii YaraZilly

Počas výskumu bola vo vývoji verzia 2.0 tohto nástroja, ktorá riešila aj nedostatky odhalené pri tvorbe tejto práce, najmä pri prvej iterácii tvorby párov. Aktualizácia bola sprístupnená vo februári 2022, kedy sa uskutočnila druhá iterácia tvorby párov, ktorá zúžitkovala vylepšenia novej verzie.

Touto verziou bolo opravené uloženie dát z disassembly v databáze, čo umožnilo pomocou API zisťovať vzťahy medzi rodinami nielen na úrovni sekvencií, ale aj basic blokov a funkcií. Vďaka tomu boli páry rodín vytvorené podľa podobností na všetkých úrovniach.

Aktualizácia nástroja priniesla tiež štandardizované pomenovanie rodín identifikátorom, ktorý má predpísané časti:

```
{zdroj}.{názov rodiny}.{cieľová architektúra}.{verzia}
```

kde zdroj značí pôvod vzorky, architektúra na akú architektúru procesora bol malvér vyvíjaný a verzia voliteľné označenie prípadnej verzie rodiny (predvolene nastavené na „default“). Vhodné rodiny tak môžu byť rozdelené do viacerých verzií so zachovaním konzistencie pomenovania, ktoré sa líši iba v poslednej časti. Tiež sa tým vyriešil problém duplicít, ak pri tvorbe rodín v YaraZille predpokladáme dodržanie správneho vyplnenia častí už spomenutého identifikátora rodiny.

Proces tvorby párov bol po tejto aktualizácii zjednodušený, lebo sa medzi rodinami neobjavovali duplicitné rodiny, ktoré sa líšili v pomenovaní. Stačilo teda odstrániť páry so zhodným *názvom rodiny* v identifikátore.

Tiež bola implementovaná automatická extrakcia nových vzoriek rodín zo zbierky Malpedia[10]. Vzorky sú automaticky zaradené k rodinám a je im priradená prípadná verzia, pod ktorou boli označené na stránke Malpedia. Vďaka tomu je databáza YaraZilly priebežne napĺňaná rodinami zo spoľahlivého zdroja a pri skúmaných vzorkách je tak čoraz väčšia šanca na odhalenie podobnosti.

Keďže bola databáza nestále napĺňaná množstvom nových vzoriek z rôznych kmeňov, tentoraz tvorilo jej obsah 2 046 rodín alebo ich variantov. Z nich vzniklo 3 085 na úrovni sekvencií, 12 911 párov na úrovni basic blokov a 3 556 párov podľa podobnosti na úrovni funkcií. Všetky výsledné páry sa nachádzajú na priloženom médiu.

Získané výsledky boli z analýz na úrovni basic blokov a funkcií boli oproti sekvenciám viac prekvapivé, pretože obsahovali príliš veľa vzťahov rodín s vysokým podielom prekryvov. Sekcia 4.4 sa bližšie zameriava na dôvod týchto zistení.

3.6 Zhodnotenie

Z výsledkov práce s nástrojom YaraZilla vyplýva, že nemôže úplne nahradiť nástroje na porovnávanie binárnych súborov ako BinDiff či Diaphora, spomínané v podsekcii 2.3, ale môže ich dopĺňať. BinDiff a Diaphora sú skôr vhodné pri podrobnej analýze podobnosti jednotlivých súborov či konkrétne vzoriek malvéru, kde dokážu lepšie vyhodnotiť nielen podobnosti v nich, ale aj rozdiely, ako ukázala sekcia 3.3. Pri hromadnej analýze malvéru ako v sekcii 3.5 alebo zisťovaní vzťahov kódu vzorky s neznámymi rodinami zas vyniká YaraZilla. Každý nástroj má svoje prednosti a nedostatky, preto je dôležité ich využiť čo najefektívnejšie podľa situácie.

Analýza rodín typu banker v sekcii 3.2 poslúžila ako prvý krok výskumu, ktorý odhalil limitácie YaraZilly, najmä čo sa týka práce s existujúcim obsahom databázy. Analýza známeho vzťahu rodín WannaCry a Contopee v sekcii 3.3 naznačila, že úroveň basic blokov v YaraZille by mohla byť najspoľahlivejšou voľbou, pretože narozdiel od funkcií, pri zmenách len určitých častí spoločnej funkcie dokáže stále odhaliť podobnosť basic blokov v nej.

Skúmanie podobnosti verzií rodiny Spora v sekcii 3.4 odhalilo dve hlavné skupiny verzií a jednu osamotenú verziu. Vlastným výberom a extrakciou zo vzoriek pre dané použitie sa touto analýzou ukázal potenciál nástroja YaraZilla, kedy konkuroval nástroju BinDiff. Iný spôsob použitia bol aplikovaný pri tvorbe párov v sekcii 3.5, a to hromadná analýza podobnosti rodín. Na výsledky spomenutých analýz nadväzuje nasledujúca kapitola.

[10] Malpedia – https://malpedia.caad.fkie.fraunhofer.de/

Kapitola 4

Hĺbková analýza

Na základe výsledkov analýzy v predošlej kapitole boli vybrané vzorky, ktoré boli následne podrobené hĺbkovej analýze. Táto kapitola podrobne popisuje jej zistenia, pričom počas jej priebehu boli použité hlavne techniky statickej a dynamickej analýzy spomínané v podsekcii 2.2 a nástroje ako IDA Pro, BinDiff a OllyDbg.

Odhalené vzťahy medzi verziami rodiny Spora sú podrobne preskúmané a je objasnený pôvod nielen prekryvov kódu ale aj odlišností v implementácii. Z párov rodín, ktoré boli vytvorené v sekcii 3.5 je analyzovaný vzťah medzi rodinami Padodor a Kraton, medzi ktorými bola nameraná podobnosť 100%. Ďalšou analyzovanou dvojicou sú vzorky rodín LockBit a Zeoticus, kedy je cieľom identifikovať účel spoločných funkcií. Posledná časť kapitoly odhaľuje príčinu odhalenia veľkého množstva vzťahov medzi rodinami, ktorou je nedostatočné filtrovanie staticky linkovaného kódu vo vzorkách.

4.1 Analýza vývoja verzií rodiny Spora

Analýza podobnosti verzií rodiny Spora v YaraZille odhalila, že všetky vzorky mali navzájom určitú úroveň podobnosti, no nezistila okolnosti týchto vzťahov. K výberu tejto rodiny prispelo aj použitie zaujímavých techník, akou je schopnosť šírenia po sieťových diskoch. Bolo zhodnotené, že preskúmanie pôvodu prekryvov kódu a jeho odlišností, a tým pádom detailov vývoja rodiny počas niekoľkých mesiacov, môže byť užitočné nielen pre nadobudnutie znalostí ohľadom tvorby tohto malvéru, ale aj budúcich hrozieb.

YaraZilla odhalila, že ako sa Spora vyvíjala, postupne sa odlišovala od prvej verzie. Podľa podobnosti môžeme rozdeliť verzie do dvoch skupín – pred verziou 2017-05-12 a od tejto verzie. Z tohto trendu najviac vybočovala verzia 2017-06-05, ktorá mala relatívne nižšiu podobnosť s predošlými aj nasledujúcim verziami. Pre odhalenie príčin týchto rozdielov je potrebné jednotlivé verzie dôkladne preskúmať metódami reverzného inžinierstva.

Táto sekcia podrobne zaznamenáva vývoj vzoriek rodiny Spora postupne podľa dátumu zostavenia v hlavičke vzorky. Toto však nutne nemusí znamenať, že sa daná verzia v tom čase aktívne šírila. Zároveň boli zachytené verzie s dátumami medzi spomínanými, ale tieto neobsahovali žiadne zmeny oproti predošlým. Analýza nadväzuje na existujúci výskum vlastností [7] a verzií [8] tejto rodiny, pričom ho výrazne rozširuje detailnejším preskúmaním vzťahov verzií a používaných techník.

4.1.1 Prvá verzia – 2017-01-10

Táto verzia je prebraná najdetailnejšie, pretože ďalšie verzie sa vyvinuli z nej a až na výnimky bola v nich funkcionalita postupne redukovaná. Detailne sú preskúmané použité techniky aj kryptografická schéma vzorky tejto verzie.

Špecifické techniky

Spora využíva viaceré techniky, ktoré sú špecifické pre skúmanú rodinu, no hlavne pre staršie verzie, pretože v neskorších verziách sú mnohé z nich zmenené, či úplne odstránené. Ich pochopenie je potrebné pre podrobnú analýzu popísanú v tejto sekcii.

Vo vzorkách je uložený *identifikátor verzie* – 10-miestny reťazec typicky hexadecimálnych číslic – ktorý pravdepodobne slúži útočníkom na zvolenie príslušných ciest vymáhania poplatku či dešifrovania súborov. V prvej verzii je to `D283C31972`.

Pri hľadaní súborov na šifrovanie sú priečinky s nasledovnými názvami ignorované:

- windows,
- program files,
- program files (x86),
- games.

Z obsahu prechádzaných priečinkov Spora cieli len na určité typy súborov na základe ich prípony. Podľa nej ich rozdeľuje do kategórií:

1. dokumenty – xls, doc, xlsx, docx, rtf, odt;
2. prezentácie – pdf;
3. grafické súbory – psd, dwg, cdr;
4. databázové súbory – cd, mdb, 1cd, dbf, sqlite, accdb;
5. obrázky – jpg, jpeg, tiff;
6. archívy – zip, rar, 7z, backup.

Tento malvér pracuje s konceptom stavov výpočtu, ktoré predstavujú jednotlivé fázy útoku na zariadenie. Tieto stavy tvoria jednoduchý stavový automat s 8 stavmi. Priebeh spustenia sa teda riadi týmito stavmi, ktoré sú priebežne zapisované do súboru, ktorý by sa dal pomenovať ako *stavový súbor*. Toto programu umožňuje pri spustení čítať zapísaný stavu v súbore, ak taký existuje, a podľa neho riadiť tok programu. Tým zabezpečí, že bude útok bude pokračovať od poslednej dokončenej fázy, ak bol skôr prerušený.

S číselným označením stavov sa pracuje ako s bezznamienkovými číslami. Pri spustení sa kontroluje, či je stav menší alebo rovný 7, ak to neplatí, vynuluje sa. Preto existuje nasledujúcich 8 stavov:

0. vytvorenie *stavového súboru*,
1. vytváranie zoznamu súborov na zašifrovanie,
2. generovanie počtov súborov patriacich každej kategórii, vytvorenie súboru so *záznamom údajov o obeti*,
3. šifrovanie súborov,
4. generovanie odkazu pre obeť (*ransom note*) a súborov so zoznamom šifrovaných súborov,

5. kontrola oprávnení procesu, prípadné spustenie s administrátorskými oprávneniami,
6. otvorenie HTML dokumentu s odkazom pre obeť,
7. šírenie malvéru prostredníctvom odkazov na lokálnych a sieťových diskoch.

Tieto stavy a priebeh spustenia je detailne popísaný v prílohe C. Stav 7 sa do stavového súboru nikdy priamo nezapisuje, ale je platný. Po tomto stave sa program ukončí, čiže vytvorením stavového súboru v správnom umiestnení s korektne zapísaným stavom 7 by sa teoreticky dalo predísť útoku. Všetky časti stavového súboru sú ale pred zapisovaním zašifrované funkciou `CryptProtectData`[1], čo komplikuje takýto spôsob ochrany.

Stavový súbor sa nachádza v priečinku, na ktorý odkazuje premenná prostredia s názvom `APPDATA`, konkrétne v operačnom systéme Windows 10 je to štandardne cesta: `C:\Users\{Meno používateľa}\AppData\Roaming`, s menom prihláseného používateľa. Súbor nemá príponu, je pomenovaný ako decimálny zápis *Volume Serial Number* systémového disku, čo je hexadecimálne číslo priradené disku pri jeho formátovaní. Obsah stavového súboru sa delí na 4 časti:

0. stav,
1. zoznam šifrovaných súborov,
2. verejný kľúč z vygenerovaného páru *lokálnych* RSA-1024 kľúčov,
3. *identifikátor obete*.

Význam verejného kľúča bude ozrejmený v rámci popisu kryptografickej schémy. Číselné označenie zodpovedá hodnotám, ktoré používajú funkcie, ktoré zabezpečujú čítanie a zápis do stavového súboru. Každá časť je zapísaná vo forme štruktúry `DATA_BLOB`, ktorá je definovaná nasledovne:

```
typedef struct _CRYPTOAPI_BLOB {
    DWORD cbData;
    BYTE *pbData;
} DATA_BLOB;
```

kde `cbData` udáva veľkosť dát, na ktoré ukazuje `pbData`.

Záznam údajov o obeti je dôležitý pre dešifrovanie súborov, ale môže byť tiež použitý na personalizáciu výšky vymáhaného poplatku na základe počtov zašifrovaných súborov, či krajiny pôvodu. Skladá sa z:

```
-----BEGIN RSA PRIVATE KEY-----
{súkromný RSA-1024 kľúč kódovaný v Base64}
-----END RSA PRIVATE KEY-----
{deň}.{mesiac}.{rok}|{meno používateľa}|{skratka krajiny}|{
identifikátor verzie}|{počty súborov}
```

Súkromný kľúč patrí k vygenerovanému páru *lokálnych* RSA-1024 kľúčov. Dátum je aktuálny v čase infikovania zariadenia, meno patrí prihlásenému používateľovi a skratka krajiny sa určí podľa predvoleného jazyka OS. Identifikátor verzie je v tejto vzorke `D283C31972`. Jednotlivé počty súborov sú znova oddelené znakom „|“ a zodpovedajú šiestim spomínaným kategóriám. Takýto záznam sa ukladá do globálnej premennej a zároveň do súborov s príponou `KEY`.

[1] `CryptProtectData` – https://docs.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptprotectdata

Zo záznamu údajov o obeti sa generuje *identifikátor obete*, ktorý obsahuje veľké písmená a číslice, po piatich znakoch oddelené čiarkou. Prvé dva znaky tvorí skratka krajiny podľa predvoleného jazyka, ďalších 5 znakov je začiatok kontrolného súčtu MD5[2] vypočítaného zo *záznamu o obeti*. Zvyšné znaky tvoria kódované počty súborov jednotlivých kategórií, zoskupené vždy po piatich znakoch. Posledná pätica v prípade potreby obsahuje výplň v podobe znaku „`Y`“. Spôsob kódovania znakov ukazuje tabuľka 4.1.

Pôvodný znak	Kódovaný znak
0	Z
1	X
2	R
3	O
4	A
5	H
6	F
7	G
8	E
9	K
|	T
výplň	Y

Tabuľka 4.1: Kódovanie znakov v identifikátore obete

Identifikátor obete tvorí názov súborov s príponou KEY, LST a HTML dokumentu s odkazom pre obeť (*ransom note*).

Malvér nešifruje napádané súbory hneď, ale najprv si cesty k nim ukladá do zoznamu. Zoznam je vo forme *streamu*, implementovaným rozhraním `IStream`[3] z Win32 API[4]. Toto rozhranie umožňuje pracovať s dátami v pamäti podobne ako so súborom – zapisovanie, čítanie a posúvanie pozície ukazovateľa. Zapisuje sa dĺžka cesty a následne samotný reťazec s cestou. Do časti 1 stavového súboru sa ukladá priamo obsah *streamu*, ale okrem toho zoznam súborov zapisuje aj ako reťazce oddelené novými riadkami do súborov s príponou `LST`.

HTML dokument s odkazom pre obeť (*ransom note*) je uložený v globálnej premennej programu. Obsahuje zástupné reťazce `{key}` a `{data}`, ktoré sú neskôr nahradené *identifikátorom obete* a *záznamom údajov o obeti* kódovaným v Base64[5]. Tieto informácie sú odoslané útočníkom pomocou HTTP požiadavkov. Inštrukcie sú v prvej verzii tejto rodiny iba v ruskom jazyku a odkazy smerujú na proxy domény ako `https://spora.bz`, ktoré sprostredkúvajú komunikáciu cez sieť Tor[6]. Cieľovou stránkou je platobný portál, popísaný v sekcii 3.4, na ktorom môže obeť vykonať platbu kryptomenou Bitcoin.

Zaujímavou technikou tohto malvéru je šírenie pomocou odkazov. Po zašifrovaní súborov vytvára na lokálnych, ale hlavne aj na sieťových diskoch súbory s príponou `lnk`, čo je formát odkazu vo Windows. Do koreňových priečinkoch diskov a na pracovnú plochu malvér skopíruje svoj spustiteľný súbor a vytvorí odkazy na vnorené priečinky. Tento súbor sa kopíruje ako skrytý, aby na prvý pohľad nevzbudil podozrenie. Z rovnakého dôvodu sa

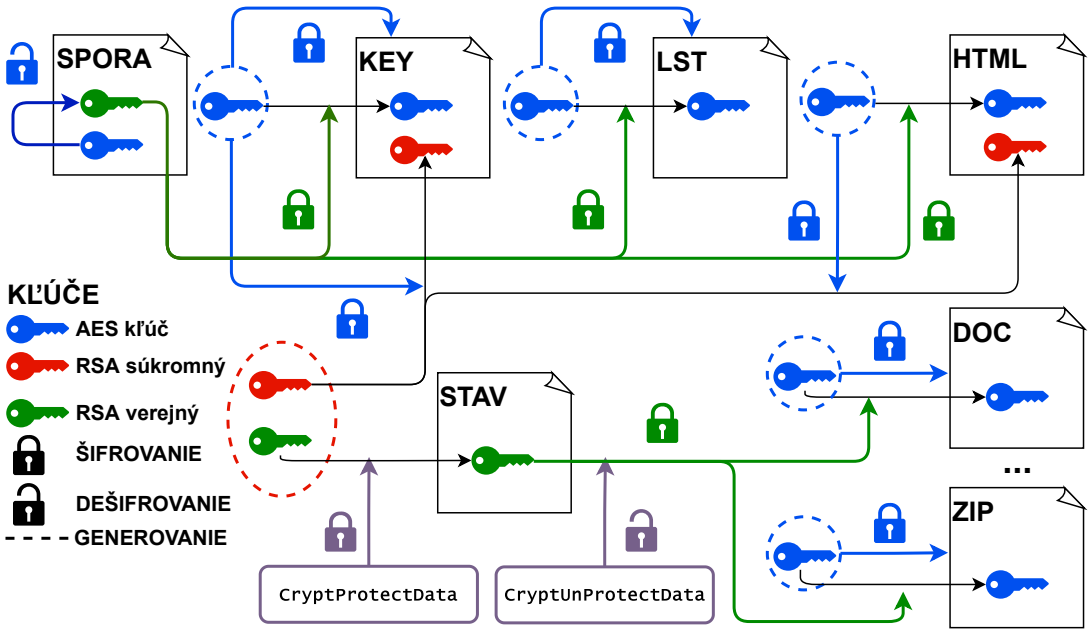
[2]MD5 (message-digest algorithm) – kryptografická hašovacia funkcia s výstupom o veľkosti 128 bitov
[3]IStream – https://docs.microsoft.com/en-us/windows/win32/api/objidl/nn-objidl-istream
[4]Win32 API – 32-bitové API pre Windows
[5]Base64 – formát kódovania binárnych dát pomocou 64 znakov ASCII
[6]Tor – systém pre anonymizáciu internetovej komunikácie

skryjú odkazované priečinky a všetkým odkazom sa skryje predvolená ikona, ktorá indikuje, že sa jedná o odkaz. To má za následok, že odkazy napodobňujú pôvodné priečinky a bežný používateľ si nemusí všimnúť rozdiel. Odkazy okrem otvorenia priečinku spustiteľný súbor skopírujú do `%tmp%`[7] a spustia ho, pričom `%tmp%` je premenná prostredia definujúca cestu k priečinku s dočasnými súbormi. Otvorením takéhoto falošného „priečinka“ sieťového disku z iného zariadenia spôsobí jeho infikovanie. Šírenie tohto druhu je typickým znakom malvéru typu červ.

Kryptografická schéma

Rodina Spora patrí k druhom ransomware, ktoré šifrujú súbory na disku napadnutého zariadenia a používa na to kombináciu algoritmov symetrickej a asymetrickej kryptografie. Symetrická je zastúpená algoritmom AES-256[8], a asymetrická RSA-1024[9]. Na generovanie, importovanie, exportovanie kľúčov, či šifrovanie súborov sa využíva Microsoft CryptoAPI[10], ktoré je bežne zahrnuté v operačnom systéme Windows.

Práca so spomínanými algoritmami tvorí vo vzorkách komplexnú kryptografickú schému. Prepracovanosť šifrovania potvrdzuje skutočnosť, že všetky súbory vytvárané programom okrem HTML dokumentu sú zašifrované, a to unikátnymi kľúčmi, ktoré sú tiež zašifrované. Na obrázku 4.1 je schéma znázornená pomocou piktogramov.



Obr. 4.1: Kryptografická schéma rodiny Spora

[7] `%tmp%` – predvolene `%USERPROFILE%\AppData\Local\Temp`

[8] AES-256 (Advanced Encryption Standard) – algoritmus symetrickej blokovej šifry s kľúčom o veľkosti 256 bitov

[9] RSA-1024 (Rivest–Shamir–Adleman) – asymetrický šifrovací algoritmus s kľúčom o veľkosti 1024 bitov

[10] CryptoAPI – https://docs.microsoft.com/en-us/windows/win32/seccrypto/cryptoapi-system-architecture

Stavový súbor, popísaný v predošlej časti, sa skladá z dátových položiek, ktoré zodpovedajú štruktúre `DATA_BLOB`. Tieto dáta sú pri zápise šifrované funkciou `CryptProtectData`[11], ktorá šifruje algoritmom používajúcim údaje prihláseného používateľa a identifikátor zariadenia tak, aby sa dali dešifrovať len rovnakým používateľom na rovnakom zariadení pomocou `CryptUnprotectData`. `pOptionalEntropy` je voliteľný parameter týchto funkcií, ktorý toto zabezpečenie rozširuje o heslo, v prípade skúmanej vzorky sa používa už spomínané *Volume Serial Number* systémového disku. Toto heslo je potom potrebné použiť vždy pri čítaní dát.

Dva kľúče sú uložené priamo v globálnych premenných programu – *hlavný* RSA-1024 verejný kľúč a AES-256 kľúč, ktorým sú zašifrované ďalšie globálne premenné. Jednou z nich je aj spomenutý *hlavný* RSA-1024 kľúč, HTML dokument s odkazom pre obeť (*ransom note*) a *identifikátor verzie*.

Dôležitú úlohu zohráva práve spomínaný *hlavný* verejný kľúč, ku ktorému patriaci súkromný kľúč by mal mať uložený útočník. Program používa ešte jeden, *lokálny* RSA-1024 pár kľúčov, z ktorého súkromný kľúč po vygenerovaní zapíše do *záznamu údajov o obeti* a verejný kľúč do časti 2 v *stavovom súbore*.

Súbor so záznamom údajov o obeti a súbor so zoznamom šifrovaných súborov sú šifrované unikátnymi AES-256 kľúčmi, ktoré sú po zašifrovaní *hlavným* verejným RSA-1024 kľúčom zapísané na koniec súboru.

Samotné napadnuté súbory sú šifrované kľúčmi AES-256, ktoré sú unikátne, vygenerované pre každý súbor zvlášť. AES-256 kľúč je potom zašifrovaný *lokálnym* verejným RSA-1024 kľúčom a z výsledných dát sa vypočíta kontrolný súčet CRC-32[12]. Na koniec napadnutého súboru sa postupne zapíše zašifrovaný kľúč a kontrolný súčet.

Pre dešifrovanie súborov je potrebné poznať útočníkov súkromný, *hlavný* RSA-1024 kľúč, pretože ním je zašifrovaný AES-256 kľúč na konci súboru so *záznamom údajov o obeti*, ktorým je zašifrovaný samotný súbor. Ako už bolo vysvetlené, tento súbor obsahuje súkromný *lokálny* RSA-1024 kľúč, ktorým sú zašifrované AES-256 kľúče napadnutých súborov. Po získaní tohto kľúča je možné dešifrovať súbory, odstrániť dáta zapísané na ich konci, a tým ich navrátiť do pôvodnej podoby.

4.1.2 Odstránenie ukladania zoznamu súborov – 2017-02-01

Táto vzorka obsahuje *identifikátor verzie* `51CA8D5791`. Už nevytvára súbory s príponou `LST` – so zoznamom šifrovaných súborov. Už predtým sa do časti 1 stavového súboru ukladal stream s cestami jednotlivých súborov, takže vytváranie týchto súborov bolo pravdepodobne redundantné.

HTML dokument s odkazom pre obeť a súbory so *záznamom údajov o obeti* sa už nekopírujú do priečinka `TEMPLATES`. Celkovo sa teda umiestnia do koreňových priečinkov pevných diskov, na pracovnú plochu, do priečinka `APPDATA` a HTML dokument ešte do `STARTUP`. Súborom s príponou `KEY` sa už nenastavuje atribút „iba na čítanie“.

4.1.3 Drobné zmeny – 2017-02-02

Identifikátor verzie sa zmenil na `2BBD38C833`. Zmeny nastali v uložení súborov so *záznamom údajov o obeti*, ktoré už nemajú príponu `KEY` – sú bez prípony, a po vytvorení v `APPDATA`

[11] `CryptProtectData` – https://docs.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptprotectdata

[12] CRC-32 (cyclic redundancy check) – funkcia kontrolného súčtu používaná na kontrolu integrity dát

sa už nikam nekopírujú. Prípona dokumentu s odkazom pre obeť (*ransom note*) sa zmenila z `HTML` na `html`, pričom tento súbor sa z `APPDATA` skopíruje iba na koreňové priečinky pevných diskov a do `STARTUP`.

Program už po vymazaní *Volume Shadow Copies* nemodifikuje nastavenia *Windows Error Recovery* režimu. Spúšťaný príkaz teda vyzerá nasledovne:

```
wmic.exe process call create
"cmd.exe /c vssadmin.exe delete shadows /quiet /all"
```

Pri vytváraní odkazov nastali dve zmeny. Bol skrátený názov kopírovaného spustiteľného súboru, ktorý už neobsahuje toľko číslic kontrolného súčtu CRC-32 z *Volume Serial Number*. Samotný odkaz má mierne pozmenený spúšťaný príkaz, no jeho funkcionalita je identická:

```
cmd.exe /c explorer.exe "{priečinok}" &
type "{názov}" > "%temp%\{názov}" &&
%temp%\{názov}
```

4.1.4 Odstránenie ukladania záznamu údajov o obeti – 2017-02-27

Verzia s *identifikátorom* `BCC3C5D2F6` nepoužíva súbory so *záznamom údajov o obeti*. Záznam je rovnako ako predtým, pomocou Base64 zakódovaný v HTML dokumente s odkazom pre obeť (*ransom note*). Teraz sa však ukladá iba tam. Je teda pozmenený stav 2, pretože po zašifrovaní záznamu AES-256 kľúčom a tohto kľúča *hlavným* RSA-1024 verejným kľúčom ich nezapisuje do súboru, ale rovno kóduje v Base64, takto necháva v pamäti, odkiaľ sa v tom istom stave zapíše do HTML dokumentu. To má za následok, že stav 4 nie je potrebný.

Odstránením pôvodného stavu 4 je počet stavov zredukovaný z 8 na 7, pričom stavy 5-7 sú označené číslami 4-6. Pre úplnosť, označenie stavov tejto verzie je nasledovné:

0. vytvorenie *stavového súboru*,
1. vytváranie zoznamu súborov na zašifrovanie,
2. generovanie *záznamu údajov o obeti* a odkazu pre obeť (*ransom note*),
3. šifrovanie súborov,
4. kontrola oprávnení procesu, prípadné spustenie s administrátorskými oprávneniami,
5. otvorenie HTML dokumentu s odkazom pre obeť,
6. šírenie malvéru prostredníctvom odkazov na lokálnych a sieťových diskoch.

Šifruje sa viacero druhov súborov, pretože ku kategórii prezentácií pribudli rozšírenia `ppt` a `pptx`, k archívom `sql` a `bak`. Šifrovanie súborov prebieha s jedným pokusom na súbor miesto piatich, čo v prípade, že nie je vždy úspešné urýchľuje celý beh malvéru.

4.1.5 Prepracovanie celého programu – 2017-05-12

Verzia 2017-05-12 s *identifikátorom* `AEFF1820BA` prináša odklon od predošlých implementácií, ktoré pracovali s konceptom stavov a *stavovým súborom*. Okrem toho nepoužíva *stream* na ukladanie zoznamu súborov a *záznam údajov o obeti* sa nezapisuje do HTML odkazu pre obeť *ransom note*. *Stavový súbor* sa vytvára, ale nič sa doňho nezapisuje, nastavuje sa pri ňom vlastnosť „iba na čítanie“ po dokončení útoku malvéru. Celý priebeh programu je tak zjednodušený, no keďže sa nepoužívajú stavy, proces útoku sa nemôže reštartovať od posledného dokončeného stavu.

Drobnou zmenou v tejto verzii je, že medzi preskakované priečinky je pridaný názov „appdata“. Malvér tiež nepoužíva nikde *Volume Serial Number*, čo znamená že aj názov kopírovaného spustiteľného súboru pri šírení je iný. Predtým vychádzal z opakovaného kontrolného súčtu CRC-32 z tohto čísla, teraz sa počíta z 0, čo znamená, že názov je pevný: `df696522.exe`.

Program pracuje s *príznakom šifrovania* ktorý má uložený globálnej premennej, inicializovaný na 0. Tento *príznak* mu dovoľuje používať rovnakú funkciu na šifrovanie súborov, zapisovanie HTML dokumentov s odkazom pre obeť aj generovanie počtov kategórií súborov. Táto funkcia zabezpečuje šifrovanie súborov alebo inkrementovanie počítadiel – ak *príznak šifrovania* nie je nastavený, šifrovanie a generovanie HTML dokumentov sa nevykoná.

Táto verzia si však stále zachováva veľmi podobnú implementáciu mnohých funkcií použitých v jednotlivých fázach útoku. To dokazuje aj fakt, že 18/25 funkcií je úplne identických. Okrem zvyšných zmenených funkcií sú najväčšie rozdiely v ich použití, čiže vysvetlenie priebehu spustenia znova nadväzuje na predošlé verzie.

Priebeh spustenia

Rovnako ako predtým, program kontroluje, či je spustený s parametrom `/u`, ak nie, vytvára *mutex*, ale tentoraz má pevný názov: „`iISp4fhZ`“. Ak je spustený s týmto parametrom, správa sa tiež podobne ako v predošlých verziách, kedy najprv známym spôsobom maže *Volume Shadow Copies*, svoj `Zone.Identifier`, a šíri sa pomocou odkazov. Rovnako odstraňuje hodnotu `IsShortcut` z kľúča `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\lnkfile` v *registri*. Po týchto úkonoch však končí, a teda pri spustení s `/u` nikdy nedochádza k šifrovaniu súborov.

Pri bežnom spustení otvára súbor v priečinku `APPDATA` na čítanie aj písanie, s názvom rovnakým ako má *mutex*. Do tohto *stavového súboru* sa už však nič nezapisuje a uzatvára sa jeho *handle*. Malvér potom maže svoj `Zone.Identifier` a šíri sa odkazmi totožne ako predtým. Ak by však otváraný *stavový súbor* mal nastavený atribút „iba na čítanie“, malvér by po šírení skončil. V tomto momente ešte program upravuje kľúč v *registri* pre súbory s príponou `lnk`.

Identicky ako minulej verzii iteruje najprv cez lokálne pevné a prenosné disky, neskôr cez sieťové disky, a hľadá na nich súbory s podporovanými rozšíreniami, ktoré sa nenachádzajú v jednom z ignorovaných priečinkov. Pri vnorených priečinkoch sa správa rovnako – rekurzívne volá funkciu pre daný priečinok. Pri súboroch však ich cesty nikam nezapisuje, len inkrementuje počítadlá príslušných kategórií. Robí to pomocou funkcie na šifrovanie, ktorá nevykoná šifrovanie, keďže *príznak šifrovania* v globálnej premennej ešte nebol nastavený. Ak je počet súborov v hociktorého počítadla väčší ako 0, program pokračuje ďalej.

Importuje sa *hlavný* RSA-1024 verejný kľúč z globálnej premennej, v ktorej je uložený ako nezašifrovaný reťazec kódovaný v Base64. V tomto momente sa vygeneruje *lokálny* RSA-1024 pár kľúčov. Generuje *záznam údajov o obeti*, ktorý už neobsahuje skratku krajiny podľa predvoleného jazyka vo Windows. Okrem toho sú malé rozdiely vo formátovaní dát – *lokálny* RSA-1024 súkromný kľúč nie je ohraničený hlavičkami `---BEGIN RSA PRIVATE KEY---` a `---END RSA PRIVATE KEY---`, neobsahuje oddeľovače riadkov a jednotlivé údaje *záznamu* sú oddelené znakom „;“ miesto „|“. Tento reťazec (v jednom riadku) tak vyzerá nasledovne:

```
{súkromný RSA-1024 kľúč kódovaný v Base64, bez oddeľovačov riadkov};
{deň}.{mesiac}.{rok};{meno};{identifikátor verzie};{počty súborov}
```

Zhodne s predošlou implementáciou vygeneruje AES-256 kľúč, ktorým zašifruje *záznam údajov o obeti*, kľúč zašifruje *lokálnym* RSA-1024 verejným kľúčom, no *záznam* ponechá v binárnej podobe, nekóduje ho v Base64. V tomto momente program nastaví *príznak šifrovania*, rovnako ako pri počítaní súborov prechádza cez pripojené disky, no tentoraz ich hneď šifruje.

Pred šifrovaním jednotlivých súborov sa zarovnanie veľkosti výsledného zašifrovaného súboru počíta odlišne – podľa veľkosti súboru ju inak počíta pre súbory do 1 GB, 100 MB, 10 MB, 1 MB, 100 kB, 10 kB. Súbory menšie ako 2048 B nešifruje vôbec. Samotné šifrovanie prebieha totožne – obsah súboru sa namapuje do pamäti pomocou `CreateFileMappingA` a `MapViewOfFile` a zašifruje sa vygenerovaným AES-256 kľúčom, ktorý sa zašifruje *lokálnym* RSA-1024 verejným kľúčom. Narozdiel od predchádzajúcej verzie sa najskôr na koniec súboru sa zapíše šifrovaný *záznam údajov o obeti* s jeho zašifrovaným AES-256, dĺžka týchto dát, a až potom zašifrovaný AES-256 kľúč súboru nasledovaný jeho kontrolným súčtom CRC-32.

Taktiež pridanou funkcionalitou je vytváranie HTML súboru s odkazom pre obeť *ransom note* pri šifrovaní súborov. V každom priečinku, kde sa šifrujú súbory sa vytvorí takýto dokument s názvom `HELP_iISp4fhZ.html`, vlastnosťou „iba na čítanie“ a nasledujúcim obsahom:

```
<meta http-equiv='refresh' content='0; url=http://190.115.19.190' />
```

Tento dokument teda zabezpečuje iba presmerovanie na danú IP adresu.

Následne program skontroluje, či beží na Windows verzie Vista (resp. Server 2008) alebo vyššej a či beží s administrátorskými oprávneniami známymi metódami. Ak áno, vymaže *Volume Shadow Copies*, ak nie, spustí sa s parametrom `/u` ako administrátor.

Vo finálnej fáze znova nájdeme viacero zmien. Rovnaký ako už spomenutý HTML dokument `HELP_iISp4fhZ.html` sa vytvorí v priečinku `STARTUP` a pokúsi sa ho 10-krát otvoriť prostredníctvom `ShellExecuteW`. Tento súbor by sa tak mal otvoriť automaticky aj pri každom prihlásení používateľa. Nakoniec sa nastaví prázdnemu *stavovému súboru* atribút „iba na čítanie“, čo značí, že bol ukončený celý proces napadnutia zariadenia a naozaj končí.

4.1.6 „Minimalistická“ verzia – 2017-06-05

Verzia 2017-06-05 je výnimkou vo vývoji rodiny Spora, pretože hneď nasledujúca verzia sa vracia k predošlej implementácii. Narozdiel od ostatných verzií sa nejedná o samostatne fungujúci malvér, pretože sa riadi parametrami pri spustení – s (generovanie zoznamu napadnutých diskov), k (generovanie kľúčov), e (šifrovanie súboru), bez nich neškodne končí. Nevie sa šíriť, vie zašifrovať iba jeden súbor pri jednom spustení, nezanecháva *ransom note*. Počet funkcií je zredukovaný na 5, preto verziu môžeme pomenovať ako „minimalistickú“. Z hľadiska podobnosti jednotlivé funkcie obsahujú črty predošlých verzií.

Spustenie s parametrom „`k`“ vyžaduje 3 ďalšie parametre. Prvý z nich je *hlavný* RSA-1024 verejný kľúč, druhý je reťazec, ktorý sa pripája na koniec zašifrovaného súboru (pravdepodobne na jeho identifikáciu), tretí je cesta súboru, do ktorého sa uložia kľúče. Vygeneruje sa *lokálny* pár RSA-1024 kľúčov, z nich súkromný zakóduje v Base64 a pripojí k nemu reťazec v druhom parametri po `k`. Vygeneruje AES-256 kľúč, ktorým zašifruje súkromný RSA-1024 kľúč spolu s reťazcom. AES-256 kľúč potom zašifruje *hlavným* RSA-1024 kľúčom. Vytvorí súbor s cestou danou parametrom, zapíše doňho postupne *lokálny* RSA-1024 verejný kľúč v binárnej podobe, *lokálny* RSA-1024 súkromný kľúč šifrovaný AES-256 kľúčom a AES-256 kľúč šifrovaný *hlavným* RSA-1024 verejným kľúčom.

Spustenie s parametrom „s“ potrebuje 1 ďalší parameter – cestu súboru, do ktorého sa uloží zoznam napadnutých sieťových diskov. Program funkciou `WNetEnumResourceW` iteruje cez nájdené sieťové zariadenia a v prípade sieťového disku ho pripojí k zoznamu. Nakoniec zoznam zapíše do súboru danom v parametri.

Spustenie s parametrom „e“ vyžaduje 2 ďalšie parametre. Obsahujú cestu súboru na zašifrovanie a súboru s kľúčmi. Po kontrole parametrov sa otvára súbor na zašifrovanie a súbor s kľúčmi. Prečíta *lokálny* RSA-1024 verejný kľúč a potom aj zvyšok súboru s kľúčmi. Pri určovaní zarovnania veľkosti šifrovaného súboru používa rovnaký algoritmus ako predošlá verzia rodiny. Napadnutý súbor sa šifruje na mieste, mapovaním obsahu do pamäti pomocou `CreateFileMappingA` a `MapViewOfFile`. Pre súbor sa vygeneruje AES-256 kľúč, ktorým ho zašifruje. Tento kľúč zašifruje *lokálnym* RSA-1024 verejným kľúčom. Na koniec súboru sa zapíše obsah súboru s kľúčmi okrem RSA-1024 verejného kľúča a veľkosť týchto dát. Nakoniec sa zapíše zašifrovaný AES-256 kľúč pre napadnutý súbor.

4.1.7 Návrat k predošlej verzii, pozmenené šifrovanie – 2017-07-24

V porovnaní s 2017-05-12 prináša táto verzia menšie zmeny. Jednou z nich je skutočnosť, že *identifikátor verzie* už neexistuje, zmenil sa však názov *mutexu* na „`eGUiKSAmJi`“. Súbory s odkazom pre obeť teraz majú inštrukcie iba v ruskom jazyku, neobsahujú spomenutý *identifikátor verzie*, majú príponu `hta` miesto `html` a názov `SPORA_eGUiKSAmJi` miesto `HELP_iISp4fhZ`.

Výraznou zmenou pri šifrovaní je fakt, že nedochádza ku kontrole, či súbor nebol už predtým šifrovaný. Môže teda dôjsť k opätovnému zašifrovaniu, aj keď predísť by mu malo znemožnenie opätovného spustenia útoku vďaka atribútu *stavového súboru* – „iba na čítanie“ (takisto ako v predošlej verzii). Na koniec šifrovaných súborov sa preto nezapisuje kontrolný súčet CRC-32, ale iba záznam údajov o obeti, údaj o jeho dĺžke a *lokálnym* RSA-1024 kľúčom zašifrovaný AES-256 kľúč súboru.

Šírenie odkazov cieli odteraz iba na sieťové disky. Predtým prebiehalo aj na lokálnych diskoch, kde sa môžeme domnievať že to nebolo príliš užitočné, keďže cez sieťové disky môže malvér napadnúť iné zariadenie, cez lokálne iba znovu napadnúť to isté. Spustiteľné súbory, ktoré odkazy kopírujú majú pevné pomenovanie, zhodné s názvom *mutexu*: `eGUiKSAmJi.exe`.

4.1.8 Uloženie údajov do HTML odkazu – 2017-07-27

Oproti predošlej verzii sa názov *mutexu* nezmenil, názov súboru s odkazom pre obeť (*ransom note*) však áno: `HELP_eGUiKSAmJi.hta`. Samotný súbor zaberá 6-násobne viac miesta, pretože znova obsahuje zakódovaný *záznam údajov o obeti*, ktorý sa posiela cez HTTP POST požiadavok. Súbor obsahuje parameter `key=BASE64_CODE`, kde sa za `BASE64_CODE` nahradí Base64 kódovaný *záznam*.

4.1.9 Verzia s ladiacimi výpismi – 2017-08-30

Názov mutexu sa tentoraz zmenil na „`sTlLoTpq`“. Okrem toho vo vzorke tvorcovia malvéru nechali príkazy, ktoré vypisujú ladiace informácie pomocou funkcie `OutputDebugStringA`. Výpisy sú veľmi časté – zaznamenávajú aktuálne volané funkcie, fázy šifrovania a cesty jednotlivých šifrovaných súborov. Výpis 4.1 obsahuje ukážku súboru so záznamom ladiacich výpisov. Kompletný súbor je zahrnutý na priloženom pamäťovom médiu.

```
Process start
Sub procedure: Get OS version OK
Getting OS                          // získanie verzie Windows
Create mutex                        // vytvorenie mutexu
Start Spreading                     // šírenie
Start Local Disk Enumerate          // vymenovanie lokálnych diskov
...
Sub procedure: Checking directory // kontrola názvu priečinka pre ignorovanie
$AAA                                // názov priečinka
...
Sub procedure: Getfilegroup         // určenie kategórie súboru podľa prípony
foobar.doc                          // názov súboru
...
Checking files count                // kontrola, či boli nájdené súbory
...
Generate Keypair                    // generovanie RSA-1024 páru kľúčov
...
-ENCRYPT STAGE: LOCAL FILES-        // šifrovanie lokálnych súborov
...
Sub procedure: Getfilegroup         // určenie kategórie súboru podľa prípony
foobar.doc                          // názov súboru
Sub procedure: Encrypt File         // šifrovanie súboru
...
-ENCRYPT STAGE: REMOTE FILES-       // šifrovanie vzdialených súborov
...
Checking admin rights               // kontrola oprávnení
Admin granted: delete shadows       // mazanie Volume Shadow Copies
...
Setting finish mark                 // označenie súboru
Encrypt procedure finished
Cleaning up
Exit process
```

Výpis 4.1: Skrátený súbor s ladiacimi výpismi vytvorený Spora verziou 2017-08-30

4.1.10 Zhodnotenie vývoja ransomware rodiny Spora

Počas analýzy bol spozorovaný zlom vo vývoji pri verzii 2017-05-12. Predošlé verzie vychádzali z pôvodnej a iteratívne zmenšovali komplexitu, no táto verzia priniesla výrazné zmeny, a to aj v implementácii viacerých techník špecifických pre túto rodinu. Verziu 2017-06-05 môžeme považovať za odnož tejto rodiny, takpovediac slepú uličku vývoja, na ktorú ďalej nebolo nadviazané, alebo akúsi testovaciu verziu použitú pri vývoji rodiny Spora.

Okrem vzoriek, ktoré boli v tejto sekcii spomenuté, bolo odhalených viacero verzií, ktoré oproti predošlej verzii prinášali vždy iba jednu výraznú zmenu – odstránenie šírenia pomocou odkazov. Vo verziách, ktoré im nasledovali, bol zaznamenaný návrat k tejto funkcionalite, čo sa vymyká inak viditeľnému priebehu vývoja verzií a naznačuje skôr, že sa jedná o verzie upravené pre špecifické použitie. Medzi takéto verzie patria tie s dátumami:

- 2017-01-16,
- 2017-02-02 (2) – s rovnakým dátumom, ale odlišná oproti analyzovanej verzii,
- 2017-03-06,
- 2017-05-16,
- 2017-06-21.

Celkovo si rodina zachovávala základnú implementáciu zachovanej funkcionality – metódu šifrovania súborov, vymenovania sieťových a lokálnych diskov, hľadania súborov na šifrovanie, či vytvárania odkazov na šírenie. Toto zistenie podporuje fakt, že nástroj BinDiff pri 14 z 23 funkcií verzie 2017-08-30 vyhodnotil ich podobnosť s príslušnými funkciami prvej verzie na aspoň 90%. Po preskúmaní sa skutočne jednalo o funkcie, ktoré plnili vo vzorkách rovnaký účel. Tieto zistenia spolu s predchádzajúcou analýzou nástrojom YaraZilla v časti 3.4 sú zhodnotené v sekcii 5.1.

4.2 Rodiny Padodor a Kraton

Vzťah medzi rodinami Padodor a Kraton bol odhalený skúmaním podobnosti párov vzoriek popísaných v sekcii 3.5. Po dohovore s konzultantom bolo na analýzu určených 1 535 vzoriek poskytnutých spoločnosťou Avast, z ktorých všetky analyzované vykazovali 100% vzájomnú podobnosť sekvencií. Preto boli z nich vybrané vzorky rovnomerne podľa ich dátumu zachytenia, postupne od prvej z 30.4.2017 až po poslednú z 2.3.2020. Vzorky rodiny Kraton boli zachytené iba medzi 6.2. a 14.2.2018. Dokopy ich bolo vybraných 7 z rodiny Padodor, zaradenej pod typ backdoor a 2 z rodiny Kraton, zaradenej pod typ trojan.

Vzorky boli vo forme DLL súborov, ktoré pri načítaní vykonávali vždy tú istú funkciu, ktorej kód by sa dal zjednodušene popísať nasledovne:

```
void Execute()
{
    if((hMutex = OpenMutex("karton")) != NULL)
    {
        CloseHandle(hMutex);
        return;
    }
    sprintf(szCommand, "%s\\%s", szSystemDir, "Ckmonm32");
    WinExec(szCommand);
}
```

Premenná `hMutex` predstavuje *handle*[13] vytváraného zdieľaného objektu – mutexu, spomínaného v podsekcii 2.1.2. Identifikátor `szCommand` predstavuje reťazec, ktorý obsahuje cestu k spustiteľnému súboru v systémovom priečinku v operačnom systéme Windows, ktorého cesta je uložená v reťazci `szSystemDir`.

Funkcia `OpenMutex` vytvára mutex a funkcia `WinExec` spúšťa príkaz s cestou danou reťazcom `szCommand`. Volaním funkcie na otvorenie mutexu a následným porovnaním návratovej hodnoty s `NULL` sa v tomto prípade docieli kontroly, či mutex s takýmto názvom už neexistuje, a ak áno, program ukončí svoju činnosť. Takto s veľkou pravdepodobnosťou program kontroluje, či systém nebol už skôr infikovaný malvérom, ktorý mutex s daným menom vytvára.

Analyzované vzorky zdieľali rovnakú funkcionalitu, rozdiely boli iba v názvoch mutexov či spustiteľných súborov v systémovom priečinku. Okrem volania vyššie popísanej funkcie vzorky neobsahovali inú funkcionalitu. Obsahovali ale veľa zbytočných inštrukcií, čo by sa dalo považovať za primitívnu metódu obfuskácie, ako sa spomína v podsekcii 2.1.2. Na obrázku 4.2 je vidieť, ako vyzerá úryvok disassemblovaného kódu jednej zo vzoriek, aj s veľkým počtom nepotrebných inštrukcií.

[13] handle – unikátny identifikátor zdroja spravovaného jadrom Windows

```
mov     eax, ebx
xor     eax, ebx
mov     ebx, eax
xor     ebx, 696Ah
sub     ebx, 7163h
add     ebx, 67B9h
mov     eax, 2062h
mul     ebx
mov     [ebp+var_120], eax
mov     ebx, eax
mov     eax, 7179h
mul     ebx
mov     [ebp+var_124], eax
mov     ebx, eax
sub     ebx, 4117h
xor     ebx, 2315h
mov     eax, 739Eh
mul     ebx
mov     [ebp+var_128], eax
mov     ebx, eax
add     ebx, 4A95h
mov     eax, ebx
add     eax, ebx
mov     ebx, eax
mov     eax, 6510h
mul     ebx
mov     [ebp+var_12C], eax
mov     ebx, eax
xor     eax, ebx
mov     ebx, eax
add     ebx, 25FEh
mov     eax, ebx
sub     eax, ebx
mov     ebx, eax
push    offset Name     ; "karton"
push    0               ; bInheritHandle
push    MUTEX_ALL_ACCESS ; dwDesiredAccess
call    OpenMutexA
mov     [ebp+hMutex], eax
or      eax, eax
jz      loc_100014E4
```

Obr. 4.2: Úryvok kódu vzorky rodiny `Padodor`

Ako najviac pravdepodobná sa javí hypotéza, že skúmané vzorky týchto rodín sú jednou súčasťou väčšieho celku malvéru, v rámci ktorého iba spúšťajú iné spustiteľné súbory. Tento malvér, a hlavne jeho ostatné súčasti ale neboli identifikované.

Záverom je, že identickosť vzoriek bola potvrdená metódami reverzného inžinierstva, a teda patria do toho istého kmeňa. V rámci nástrojov Avastu boli tieto rodiny zlúčené do rodiny Padodor, čo prispelo k presnejšej klasifikácii existujúcich aj prípadných novo zachytených vzoriek. Na základe analýzy bolo tiež navrhnuté detekčné pravidlo popísané v časti 5.3.1.

4.3 Rodiny LockBit a Zeoticus

Tieto rodiny boli vybrané pri analýze vzoriek zo zbierky vx-underground[14], zároveň sa ale nachádzajú aj v zbierke vzoriek na stránke Malpedia[15]. Vo všetkých zdrojoch je k rodine LockBit zaradená vzorka s nasledujúcou hodnotou hašu SHA-256:

`9feed0c7fa8c1d32390e1c168051267df61f11b048ec62aa5b8e66f60e8083af`

[14] vx-underground – https://www.vx-underground.org/
[15] Malpedia – https://malpedia.caad.fkie.fraunhofer.de/

Medzi touto rodinou a Zeoticus neexistuje žiaden známy vzťah, no nástroj YaraZilla pri inšpekcii odhalil podobnosti na úrovni uvedenej v tabuľke 4.2.

Podobná rodina	YaraZilla		
	Sekvencie	Basic bloky	Funkcie
`Zeoticus`	13% (6 538 / 51 088)	20% (56 / 277)	15% (7 / 47)
`LockBit (Avast)`	0% (101 / 52 635)	2% (2 / 214)	1% (1 / 78)
`LockBit (Malpedia)`	0% (101 / 93 266)	1% (4 / 432)	1% (1 / 75)

Tabuľka 4.2: Podobnosti skúmanej vzorky `LockBit (Malpedia)` s inými rodinami

Vzorka rodiny LockBit mala prekvapujúco nízku podobnosť s touto rodinou a nezanedbateľnú podobnosť s ransomware Zeoticus. Lockbit aj Zeoticus sú rodiny typu ransomware, ktoré šifrujú súbory na disku napadnutého zariadenia. Obe rodiny zašifrujú súbory aj bez pripojenia k internetu, ale Zeoticus funguje plne offline, nekomunikuje teda so žiadnym riadiacim serverom. Spoločným znakom je, že sa vyhýbajú aktivite v rusky hovoriacich a im susedných štátoch.

Tieto rodiny na šifrovanie používajú kombináciu algoritmov symetrickej a asymetrickej kryptografie. Obe využívajú algoritmus Curve25519[16] na generovanie páru kľúčov používaných pri šifrovaní [20] [15]. Zeoticus pracuje s výberom algoritmov zameraných na rýchlosť – okrem Curve25519 používa Poly1305[17] a Salsa20[18].

LockBit používa viacero metód obfuskácie. Jednou z nich je použitie zašifrovaných názvov importovaných knižníc a funkcií, ktoré sa pri spustení dešifrujú vlastným algoritmom. Vzorka rodiny Zeoticus používa podobnú techniku, no dešifrovanie reťazcov prebieha algoritmami XTEA[19] alebo Murmur2[20]. LockBit implementuje aj techniku *anti-debuggingu*, kedy pri detekcii, že beží v debuggeri skočí do nekonečného cyklu.

Vzorky oboch rodín boli analyzované pomocou techník statickej aj dynamickej analýzy. Nasledujúca podsekcia sa zaoberá zisteniami analýzy spoločného kódu, ktorý predstavoval iba malú časť celkovej implementácie vzoriek, pričom čerpá najmä zo statickej analýzy.

4.3.1 Analýza spoločného kódu

Zo 7 spoločných funkcií odhalených pomocou YaraZilly sa podarilo identifikovať 3 funkcie implementujúce kryptografické algoritmy. Takéto algoritmy je možné niekedy identifikovať pomocou konštánt či postupností inštrukcií, ktoré využívajú. Patria medzi ne nasledujúce:

`salsa_20_1`

Funkcia implementuje algoritmus Salsa20, čo dokazujú napríklad použité konštanty [2]:

- `0x61707865`,
- `0x3320646e`,
- `0x79622d32`,
- `0x6b206574`.

[16] Curve25519 – kryptografický algoritmus eliptickej krivky používaný na rôzne účely
[17] Poly1305 – univerzálna hašovacia funkcia
[18] Salsa20 – kryptografický algoritmus prúdovej šifry
[19] XTEA (eXtended TEA) – bloková šifra so 128-bitovým kľúčom
[20] Murmur2 – nekryptografická hašovacia funkcia

Pre Salsa20 je typické využitie bitových rotácií doľava o 7, 9, 13, 18 bitov a operácie sčítania, rotácie, exkluzívneho OR [2]. Ako je vidieť na obrázku 4.3, tieto sa vo funkcii taktiež vyskytujú, v tomto prípade sa používa miesto rotácie doľava o 18 bitov rotácia doprava o 14 bitov, čo je pri 32-bitových registroch ekvivalentná operácia.

```
loc_408A56:
mov     eax, [ebp+var_24]
add     eax, esi
mov     ecx, [ebp+var_24]
rol     eax, 7
xor     [ebp+var_14], eax
mov     eax, [ebp+var_14]
add     eax, esi
rol     eax, 9
xor     [ebp+var_4], eax
mov     eax, [ebp+var_4]
add     eax, [ebp+var_14]
rol     eax, 13
xor     ecx, eax
mov     eax, [ebp+var_4]
add     eax, ecx
mov     [ebp+var_24], ecx
ror     eax, 14
xor     esi, eax
mov     ecx, [ebp+var_18]
```

Obr. 4.3: Časť implementácie funkcie `salsa_20_1`

`salsa_20_2`

Druhá funkcia implementujúca algoritmus Salsa20 pri šifrovaní pracuje s väčším objemom dát, jedná sa teda pravdepodobne o takto špecializovanú variantu predchádzajúcej funkcie. Tiež v nej nájdeme konštanty `0x61707865`, `0x3320646e`, `0x79622d32` a `0x6b206574`, rovnako tak postupnosť rotácií, sčítania a exkluzívneho OR ako je na obrázku 4.4.

```
loc_408CC3:
mov     ecx, [ebp+var_10]
mov     eax, [ebp+var_C]
add     eax, ecx
rol     eax, 7
xor     edx, eax
lea     eax, [edx+ecx]
mov     ecx, [ebp+var_28]
rol     eax, 9
xor     [ebp+var_24], eax
mov     eax, [ebp+var_24]
add     eax, edx
rol     eax, 13
xor     [ebp+var_C], eax
mov     eax, [ebp+var_C]
add     eax, [ebp+var_24]
ror     eax, 14
xor     [ebp+var_10], eax
mov     eax, [ebp+arg_0]
```

Obr. 4.4: Časť implementácie funkcie `salsa_20_2`

`blake2b`

Viaceré funkcie, ako `blake2b_initialize`, `blake2b_update` a `blake2b_final` implementujú algoritmus BLAKE2b[21], vychádzajúci z ChaCha, ktorý je zas odvodený zo Salsa20. Tieto tri funkcie sú podobné v rodinách LockBit a Zeoticus, ale analýzou v YaraZille bola

[21] BLAKE2b – kryptografická hašovacia funkcia

odhalená iba zhoda v `blake2b_update`. S identifikáciou funkcií pomohli Lumina metdáta v nástroji IDA Pro. Pri inicializácii algoritmu sa používajú konštanty zhodné s algoritmami SHA [17]:

- `0x6A09E667F3BCC908`,
- `0xBB67AE8584CAA73B`,
- `0xA54FF53A5F1D36F1`,
- `0x510E527FADE682D1`,
- `0x3C6EF372FE94F82B`,
- `0x9B05688C2B3E6C1F`,
- `0x1F83D9ABFB41BD6B`,
- `0x5BE0CD19137E2179`.

4.3.2 Zhodnotenie

Analýzou spoločného kódu vzoriek LockBit a Zeoticus bol odhalený jeho účel – bol súčasťou rôznych kryptografických algoritmov, ale jeho pôvod zostával otázkou. Spomínané kryptografické algoritmy sú totiž súčasťou implementácie mnohých knižníc s týmto zameraním, alebo je k nim dokonca dostupná vzorová implementácia, ako v prípade BLAKE2b [17]. Možnosťou, ako odhaliť, či sa jedná o kód špecifický pre tieto rodiny, je zistenie výskytu spoločného kódu v rozsiahlom súbore rôznych vzoriek, a práve tomu sa venuje sekcia 5.2. Na základe celkovej analýzy vzorky z rodiny Zeoticus boli navrhnuté detekčné pravidlá popísané v časti 5.3.2.

4.4 Rodiny s podobnosťami v staticky linkovanom kóde

Pri analýze mnohých vzoriek pomocou nástroja YaraZilla sa opakovali niektoré podobné rodiny. Počty podobných funkcií a basic blokov sa často pohybovali v desiatkach až stovkách, čo vzbudzovalo podozrenie voči správnosti týchto dát. Rodiny pochádzali z párov vytvorených v sekcii 3.5, ale aj zo zbierky vx-underground[22]. Ako sa ukázalo pri analýze metódami reverzného inžinierstva, takéto prípady boli spôsobené použitím rovnakej knižnice, *frameworku* či vývojového prostredia pri tvorbe daného malvéru. Kód z takejto knižnice bol staticky linkovaný k binárnemu spustiteľnému súboru, takže ho nebolo možné ľahko odlíšiť od vlastného kódu. V niektorých prípadoch boli funkcie takéhoto kódu označené nástrojom IDA Pro ako neidentifikovaná staticky linkovaná funkcia s názvom `unknown_libname`. Vo väčšine prípadov ale tento kód nebol ľahko detegovateľný. Vtedy indície k použitiu určitej knižnice mohli ponúkať prípadné reťazce zahrnuté v binárnom súbore, ako napríklad *copyright* deklarácia s menom autora, alebo výpisy používané pri behu funkcií. Tento spôsob ale nedokázal odhaliť, ktoré konkrétne funkcie pochádzali z danej knižnice.

Jednalo sa o všeobecný problém, ktorý komplikoval výber vzoriek pre hĺbkovú analýzu. Rozsah týchto zistení sa nedal vyčísliť kvôli jeho komplikovanej detekcii. Medzi odhalenými knižnicami sa objavili systémové knižnice Delphi, C++, Go, knižnice MFC, OpenSSL, zlib, libcurl a mnohé iné. V nasledujúcich podsekciách budú popísané dva špecifické výskyty tohto problému.

[22] vx-underground – https://www.vx-underground.org/

4.4.1 Skupina rodín využívajúcich knižnicu MFC

Najčastejšími zástupcami medzi rodinami používajúcimi knižnicu MFC boli rodiny Nemim, WinMM, Kurton, Leash a Tapaoux. Spoločným znakom spomínaných rodín bolo použitie knižnice MFC[23] a zostavenie pomocou Microsoft Visual C++. Knižnica MFC je určená na tvorbu aplikácií pre operačný systém Windows v jazyku C++, s využitím princípov objektovo orientovaného programovania. Postavená je na Windows API, pričom nad ním tvorí tzv. *wrapper*[24].

Pri analýze spoločného kódu rodín WinMM a Nemim bolo možné identifikovať množstvo funkcií pochádzajúcich z knižnice MFC priamo v nástroji IDA Pro, no rozsah tohto zistenia zostával otázkou. Práve na tento účel bola vytvorená jednoduchá aplikácia využívajúca MFC, ktorej funkcionalita zahŕňala iba zobrazenie dialógového okna pri spustení.

Inšpekcia tejto aplikácie v nástroji YaraZilla odhalila zhody určitých častí kódu u 54 rodín. Na úrovni sekvencií nebola zaznamenaná žiadna podobnosť vyššia ako 1%, no až 35 z 50 rodín prekonalo túto hranicu na úrovni basic blokov a funkcií. Najpodobnejšie boli práve rodiny WinMM a Nemim. Tieto zistenia sú zdokumentované v tabuľke 4.3. Tabuľka obsahuje identifikátory rodín predstavené v časti 3.5.3, pozostávajúce zo zdroja, názvu rodiny, cieľovej architektúry a verzie rodiny, oddelených bodkou.

[23] MFC – https://docs.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications
[24] wrapper – kód zaobaľujúci existujúcu implementáciu s určitou úrovňou abstrakcie

Podobná rodina	YaraZilla	
	Basic bloky	Funkcie
malpedia.WinMM.x86_32.default	100,0% (1 / 1)	92,5% (49 / 53)
malpedia.nemim.x86_32.default	65,5% (167 / 255)	86,7% (111 / 128)
malpedia.pngdowner.x86_32.default	34,2% (26 / 76)	60,0% (12 / 20)
malpedia.leash.x86_32.2015-02-25	24,7% (56 / 227)	58,8% (57 / 97)
malpedia.rikamanu.x86_32.2013-06-20	32,3% (10 / 31)	50,0% (9 / 18)
malpedia.Leash.x86_32.default	24,6% (47 / 191)	57,3% (59 / 103)
malpedia.kurton.x86_32.2012-05-24	26,2% (56 / 214)	47,2% (50 / 106)
malpedia.httpbrowser.x86_32.2015-06-30	27,5% (11 / 40)	45,0% (9 / 20)
malpedia.leash.x86_32.2016-06-26	21,7% (61 / 281)	47,1% (57 / 121)
malpedia.tapaoux.x86_32.default	22,9% (59 / 258)	42,6% (52 / 122)
malpedia.htran.x86_32.2011-05-20	21,7% (10 / 46)	39,1% (9 / 23)
malpedia.rarstar.x86_32.default	14,3% (9 / 63)	38,1% (8 / 21)
malpedia.paladin.x86_32.default	14,3% (11 / 77)	31,0% (9 / 29)
malpedia.miragefox.x86_32.2018-06-08	17,9% (33 / 184)	23,0% (14 / 61)
malpedia.kuaibu8.x86_32.2017-01-30	10,9% (64 / 588)	25,9% (56 / 216)
malpedia.wormhole.x86_32.2014-06-10	28,1% (9 / 32)	5,9% (1 / 17)
malpedia.naikon.x86_32.default	10,7% (19 / 178)	22,0% (13 / 59)
malpedia.typehash.x86_32.default	13,6% (12 / 88)	17,3% (9 / 52)
zoo.Ipsee.x86_32.default	8,8% (88 / 1 004)	18,0% (92 / 510)
zoo.ZombieBoy.x86_32.2019-09-17	6,8% (19 / 280)	18,4% (95 / 517)
malpedia.mbrlock.x86_32.2018-02-04	7,2% (96 / 1 341)	14,4% (88 / 610)
zoo.Termite.x86_32.default	7,1% (84 / 1 179)	13,8% (88 / 640)
malpedia.flystudio.x86_32.default	5,4% (112 / 2 084)	12,4% (93 / 753)
malpedia.hoplight.x86_32.default	4,8% (12 / 249)	10,6% (10 / 94)
malpedia.volgmer.x86_32.2016-03-30	4,0% (10 / 247)	10,8% (8 / 74)
...		

Tabuľka 4.3: Podobnosti MFC dialógovej aplikácie s rodinami v databáze YaraZilla

Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name
1.00	0.90	00411A31	sub_00411A31	Normal	00425AB0	AfxGetAfxWndProc
1.00	0.96	00415017	sub_00415017	Normal	0043489F	AfxGetThreadState
1.00	0.96	0041526F	sub_0041526F	Normal	00434B55	AfxGetModuleThreadState
1.00	0.99	00405E55	sub_00405E55	Normal	00401320	AfxGetMainWnd
1.00	0.99	00413984	sub_00413984	Normal	00423C3C	AfxGetThread
1.00	0.99	00415249	executed?A...	Normal	00434B2F	AfxGetModuleState
1.00	0.99	00412D38	?AfxGetPare...	Normal	004270C2	AfxGetParentOwner

Obr. 4.5: Ukážka funkcií z MFC vo vzorke rodiny WinMM

Obrázok 4.5 obsahuje ukážku funkcií z nástroja BinDiff, ktoré boli zhodné s funkciami rodín Nemim a WinMM. Majú predponu `Afx`, čo značí, že patria aplikačnému frameworku knižnice MFC [11]. Položka „Normal“ v stĺpci „Type“ v rozhraní nástroja BinDiff značí, že danú funkciu považuje za normálnu funkciu, nepatriacu do knižnice, lebo v tom prípade by ju označil ako „Library“. Skutočnosť, že aj BinDiff nedokázal takto označiť funkcie z tak rozšírenej knižnice akou je MFC len zdôrazňuje zložitosť problému detekcie staticky linkovaného kódu.

4.4.2 Rodíny písané v jazyku Go

Pri analýze malvéru naprogramovanom v jazyku Go[25] je výhodou, že takto vytvorené spustiteľné súbory si zachovávajú informácie o názvoch funkcií. Taktiež sú všetky funkcie pochádzajúce z rovnakého zdroja usporiadané priamo za sebou. Pri disassembly stačí nájsť adresu prvej a poslednej funkcie, ktorá pochádza z daného balíčka[26] a je jasné, že všetky funkcie v tomto rozsahu z neho tiež pochádzajú. Vďaka tomu je relatívne jednoduché overiť, či sa nachádzajú zhody aj vo vlastnom kóde vyfiltrovaním basic blokov alebo funkcií z príslušného rozsahu adries.

Vzťahy sa našli napríklad medzi rodinami Pysa Ransomware a Klingon RAT, a medzi Hive Ransomware a JCry. Vzorky rodín Pysa Ransomware a Hive Ransomware pochádzajú zo zbierky vx-underground[27], Klingon RAT a JCry boli súčasťou databázy YaraZilla.

Similarity	Confidence	Address	Primary Name
1.00	0.99	0052EC50	crypto_ecdsa_PrivateKey_Add
1.00	0.99	0052ED90	crypto_ecdsa_PrivateKey_IsOnCurve
1.00	0.99	0052EF20	crypto_ecdsa_PrivateKey_ScalarMult
1.00	0.99	00535CF0	crypto_rand_init_0
1.00	0.99	0053E340	crypto_des_ksRotate
1.00	0.99	00540620	crypto_sha1___ptr_digest__Sum
1.00	0.99	005449F0	crypto_sha256_appendUint64
1.00	0.99	0054C500	crypto_md5_appendUint64
1.00	0.99	005C8E70	crypto_tls_prf12
1.00	0.99	005CE070	crypto_tls_newConstantTimeHash_func1
1.00	0.99	00518560	crypto_elliptic___ptr_p256Curve__Add
1.00	0.99	00518690	crypto_elliptic___ptr_p256Curve__Double
1.00	0.99	00542360	crypto_sha1_blockAVX2
1.00	0.99	00510690	crypto_elliptic_p224ToAffine
1.00	0.99	005123A0	crypto_elliptic_maybeReduceModP

Obr. 4.6: Spoločné kryptografické funkcie vzoriek rodín Pysa Ransomware a Klingon RAT

Na obrázku 4.6 je časť spoločných funkcií vzoriek rodín Pysa Ransomware a Klingon RAT, ktoré boli označené nástrojom BinDiff ako identické. Patria k nim funkcie z `crypto`[28] a jemu podradeným balíčkom, ktoré sú súčasťou štandardnej knižnice jazyka Go a implmentujú rôzne šifrovacie algoritmy. Na obrázku tiež môžeme vidieť funkcie, ktoré sú súčasťou implementácie algoritmov ECDSA, DES, SHA-1, SHA-256, MD5 či protokolu TLS.

[25] Go – moderný programovací jazyk vyvinutý v Google – https://go.dev/
[26] balíček (*package*) – celok zdrojových súborov Go nachádzajúcich sa v jednom priečinku
[27] vx-underground – https://www.vx-underground.org/
[28] crypto package – https://pkg.go.dev/crypto

Similarity	Confidence	Address	Primary Name
1.00	0.99	00423A70	runtime_stdcall1
1.00	0.99	00401EC0	runtime_internal_atomic_Or8
1.00	0.99	004030E0	sync_atomic_StorePointer
1.00	0.99	004475C0	runtime_aeshashbody
1.00	0.99	00401CF0	runtime_internal_atomic_Loaduintptr
1.00	0.99	004474C0	runtime_abort
1.00	0.99	00401220	sync_atomic_CompareAndSwapUintptr
1.00	0.99	00448760	runtime__div64by32
1.00	0.99	00401250	sync_atomic_LoadUint32
1.00	0.99	00426D60	runtime_Gosched
1.00	0.97	004478A0	runtime_return0
1.00	0.97	004251E0	runtime_getargp
1.00	0.99	00448360	runtime_sigtramp

Obr. 4.7: Spoločné funkcie vzoriek rodín Hive Ransomware a JCry

Obrázok 4.7 obsahuje ukážku spoločných funkcií vzoriek rodín Hive Ransomware a JCry, zobrazených v nástroji BinDiff. Vyskytujú sa tu funkcie zo štandardnej knižnice jazyka Go, konkrétne z balíčkov `runtime`[29] a `sync`[30].

Žiadne zo skúmaných vzoriek neobsahovali spoločný kód mimo funkcií z balíčkov štandardnej knižnice. Dá sa konštatovať, že sa teda jedná o rozdielne rodiny, ktorých spoločným znakom je, že sú napísané v programovacom jazyku Go s využitím štandardnej knižnice.

4.4.3 Zhodnotenie

Pri vzorkách využívajúcich knižnicu MFC bolo možné vytvoriť veľmi podobnú aplikáciu, ktorá ju tiež využívala. Basic bloky a funkcie, u ktorých sa našla zhoda s touto aplikáciou boli vďaka tomu označené v nástroji YaraZilla ako neškodlivý kód a boli tak prečistené dáta u 54 rodín. V tomto prípade bola táto metóda efektívna, no jej rozsiahlejšie použitie nie je realistické vzhľadom na množstvo rodín v databáze YaraZilly.

Určitý výskyt staticky linkovaného kódu v podobnostiach odhalených nástrojom YaraZilla bol očakávaný. Aj keď by takýto kód mala identifikovať technológia F.L.I.R.T, spomínaná v časti 2.3.3, nedá sa predpokladať, že bude dokonale účinná. Nástroj BinDiff zas neponúka filtráciu, no pri funkciách pochádzajúcich z knižníc by ich mal byť schopný označiť, čo sa často nepotvrdilo. Použité metódy pre detekciu takého kódu teda neboli efektívne.

Ukázalo sa tak, že problém detekcie staticky linkovaného kódu je dnes stále aktuálny a predstavuje komplikáciu pri hľadaní vzťahov v kóde malvéru. Pre nájdenie doposiaľ neznámych vzťahov v rámci tejto práce sa problém ukázal ako kritický, pretože jeho výskyt prekryl možné vzťahy škodlivého kódu medzi rodinami.

[29] runtime package – https://pkg.go.dev/runtime
[30] sync package – https://pkg.go.dev/sync

Kapitola 5

Vyhodnotenie výsledkov výskumu

Pre zdokonaľovanie klasifikácie a zároveň ochrany voči aktuálnym aj budúcim hrozbám v podobe malvéru sa javí ako cenné zhromažďovanie informácií nielen o ich správaní, šírení, vývoji, ale aj vzájomných vzťahoch medzi nimi. Doposiaľ nadobudnuté výsledky priniesli viaceré takéto poznatky, no aby ich bolo možné neskôr zúžitkovať je potrebná ich správna interpretácia. O to sa pokúša táto kapitola, ktorá zhŕňa a vyvodzuje závery zo zistení výskumu.

V nasledujúcej sekcii je celkovo zhodnotená analýza verzií rodiny Spora spojením poznatkov nadobudnutých pri zisťovaní podobnosti v sekcii 3.4 a hĺbkovej analýze v časti 4.1. Prináša nový pohľad na dosiahnuté výsledky, a to ako na vývoj verzií rodiny, tak aj na prácu s nástrojom YaraZilla.

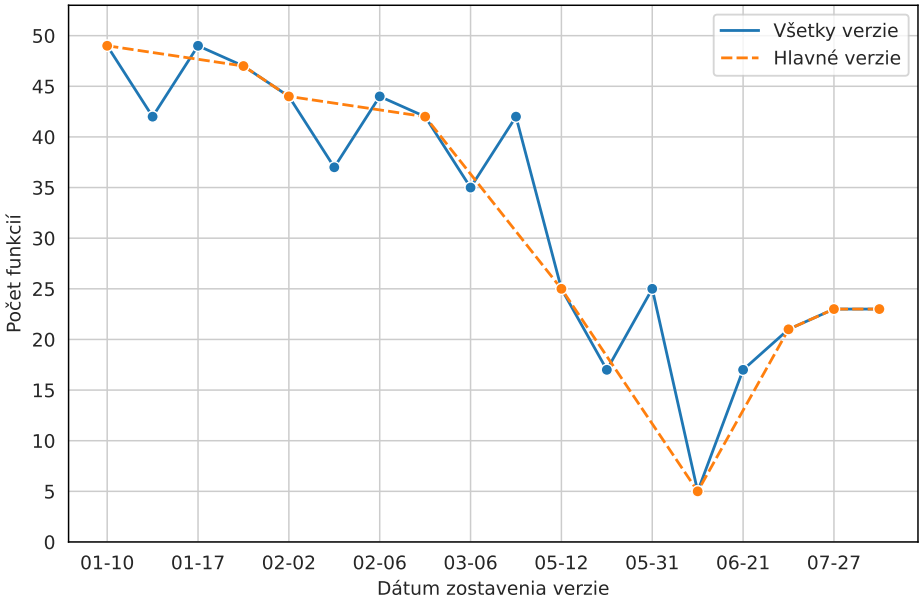
Skúmaním vzťahu rodín Lockbit a Zeoticus bolo odhalené, že spoločný kód bol implementáciou šifrovacích algoritmov, ale nebol odhalený jeho pôvod. Sekcia 5.2 sa práve pokúša odhaliť pôvod kódu prostredníctvom zistenia jeho výskytu vo vzorkách malvéru a bežného softvéru. Sú pri tom použité detekčné pravidlá vygenerované zo spoločného kódu vzoriek. Pre klasifikáciu vzoriek kmeňov, u ktorých bol odhalený doposiaľ neznámy vzťah sú navrhnuté detekčné pravidlá v časti 5.3 na základe zistení hĺbkovej analýzy.

Práca s nástrojom YaraZilla tiež poukázala na jeho početné nedostatky, z ktorých boli mnohé napravené. Ich úspešné odhalenie prispelo k vylepšeniu nástroja. Zaznamenaným problémom a celkovému zhodnoteniu výskumu sa venujú záverečné časti tejto kapitoly.

5.1 Zhodnotenie analýzy verzií rodiny Spora

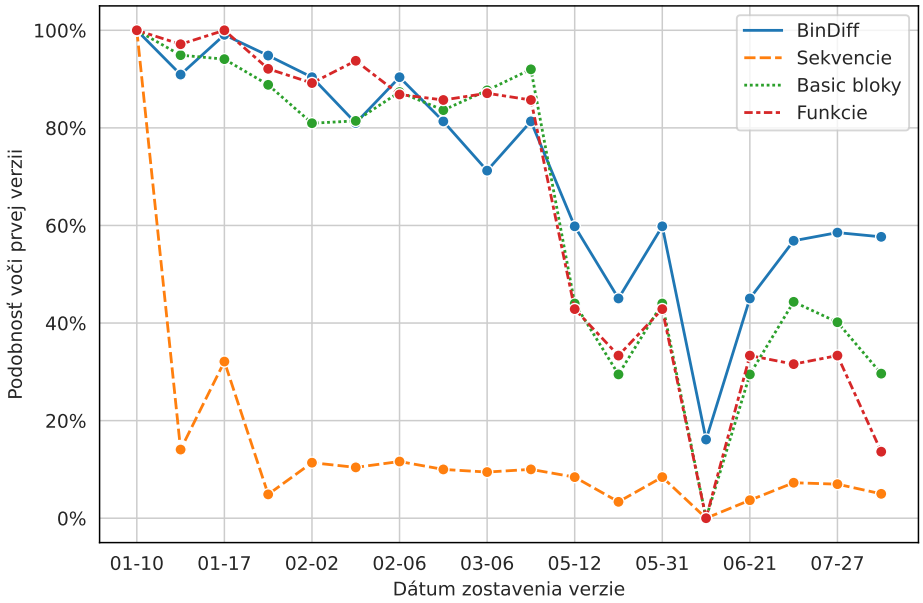
Postupná analýza v sekciách 3.4 a 4.1 poskytla komplexný pohľad na vývoj rodiny Spora. Hĺbková analýza objasnila v čom spočívajú vzťahy medzi jej jednotlivými verziami – rozdiely a podobnosti. No už analýza podobností naznačila, ako postupoval vývoj odhalením miery podobnosti týchto verzií. Zároveň sa preverili schopnosti nástroja YaraZilla, keď bol priamo porovnaný so zaužívaným nástrojom BinDiff.

Fenoménom, ktorý sme mohli pozorovať bolo postupné redukovanie komplexnosti vzoriek počas vývoja rodiny. To sa odrazilo na znižujúcom sa počte funkcií. Nie je známe, prečo sa autori malvéru rozhodli pre zjednodušovanie implementácie, no motiváciou mohlo byť zmenšenie spustiteľného súboru. To môže uľahčiť šírenie v situáciách, kedy je veľkosť obmedzená, napríklad vložením do iného spustiteľného súboru či skriptu. Verzia 2017-06-05 s veľkosťou 5 120 bytov a dokopy 5 funkciami dotiahla toto redukovanie najďalej, čo je viditeľné aj na grafe 5.1.

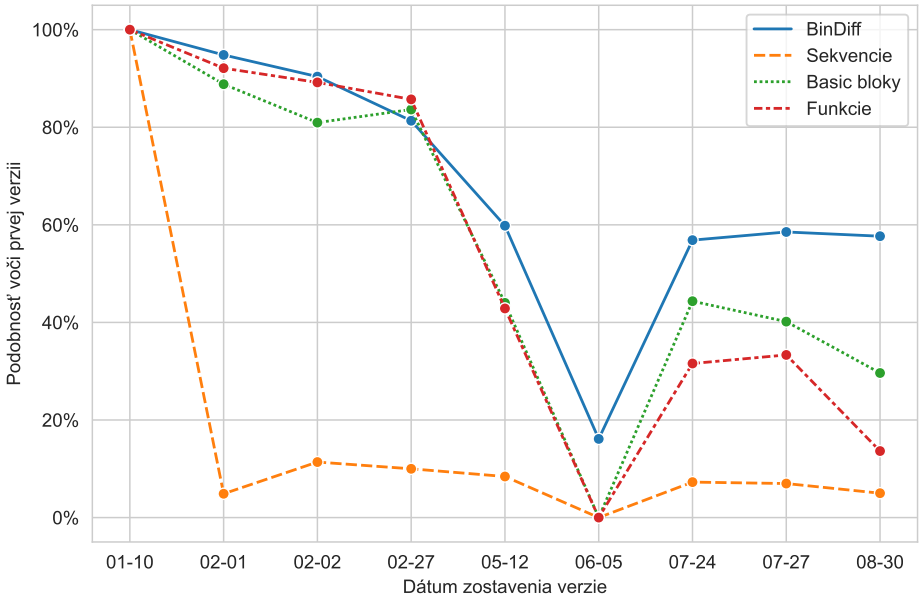


Obr. 5.1: Graf vývoja počtu funkcií jednotlivých verzií rodiny Spora

Pojmom *hlavné verzie* sú označené všetky verzie, ktoré upravovali implementáciu malvéru aj iným spôsobom, ako len odstránením funkcionality šírenia. Vzorky bolo takto možné rozdeliť vďaka zisteniam hĺbkovej analýzy v podsekcii 4.1. Spora sa postupne vyvíjala, čím sa prevažne vzďaľovala od implementácie prvej verzie, ale verzie s odstráneným šírením sa vymykajú tomuto vývoju. Táto skutočnosť sa odzrkadlila v kolísavom klesaní podobnosti na grafe 5.2 a postupnom klesaní na grafe 5.3.

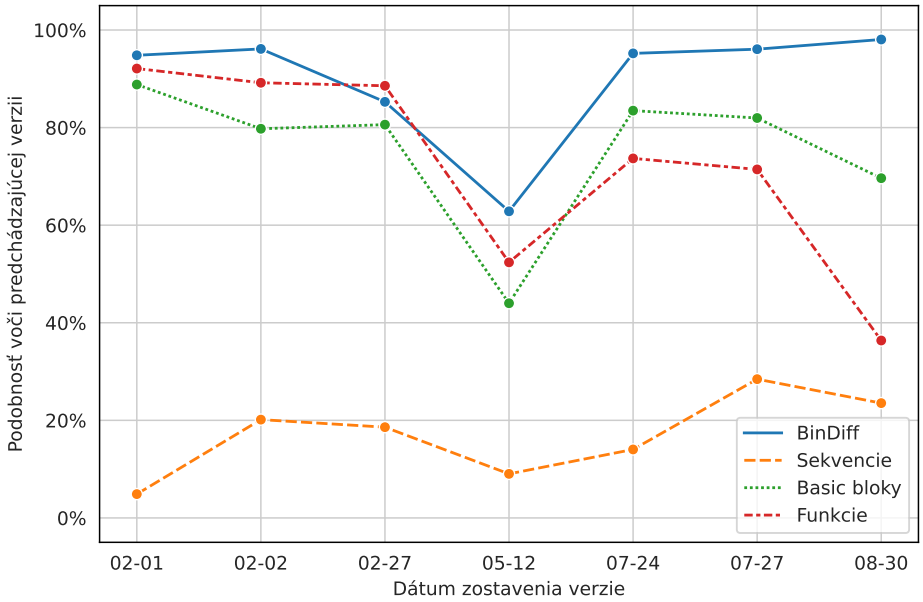


Obr. 5.2: Graf vývoja podobnosti voči prvej verzii rodiny Spora



Obr. 5.3: Graf vývoja podobnosti *hlavných verzií* voči prvej verzii rodiny Spora

Ako bolo spomenuté pri hĺbkovej analýze v časti 4.1.6, verzia 2017-06-05 – „minimalistická“ verzia najviac vybočuje z vývoja rodiny, čo je vidieť na grafe 5.3, kde je podobnosť s prvou verziou podľa nástroja YaraZilla takmer nulová. Nepredstavovala jeden z vývojových stupňov a teda pri vizualizácii tohto vývoja na grafe 5.4 je vhodné ju vynechať.



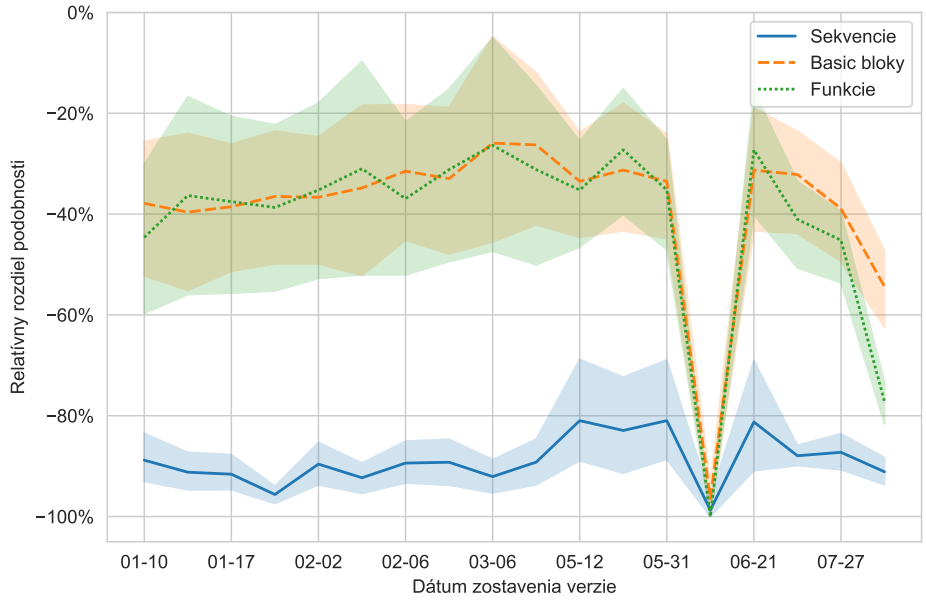
Obr. 5.4: Graf vývoja podobnosti verzií voči predchádzajúcej verzii

Tento graf ukazuje, ako sa líšila daná verzia od svojej predchádzajúcej. Obsahuje podobnosti *hlavných verzií* rodiny, bez „minimalistickej“ verzie, a teda sa zameriava len na hlavnú

vývojovú vetvu rodiny. Vizualizácia súhlasí so zisteniami hĺbkovej analýzy, kde rozsah zmien v implementácii je postupný, až na verziu 2017-05-12, ktorá priniesla výrazné zmeny.

Zaujímavým zistením je aj fakt, že percentuálna podobnosť poslednej vzorky – či už s predchádzajúcou alebo prvou verziou – poklesla na všetkých úrovniach nástroja YaraZilla. Ako bolo objasnené počas hĺbkovej analýzy v podsekcii 4.1.9, táto verzia oproti predošlej pridáva iba vykonávanie ladiacich výpisov. Tieto výpisy sa uskutočňujú prostredníctvom funkcie `OutputDebugStringA`, čo znamená, že pri každom výpise oproti predchádzajúcej verzii pribudnú minimálne dve nasledujúce inštrukcie: `PUSH` pre umiestnenie parametra funkcie – adresy vypisovaného reťazca – na zásobník a `CALL` pre zavolanie funkcie. Po pridaní inštrukcií YaraZilla neoznačí zasiahnuté funkcie a basic bloky za podobné a tým sa zníži percentuálna hodnota podobnosti, čo sa odrazilo na grafoch.

Zároveň na grafoch 5.2, 5.3 a 5.4 tiež môžeme vidieť, že YaraZilla na úrovni basic blokov aj funkcií do istej miery kopíruje trend podobnosti podľa nástroja BinDiff. Podobnosť na úrovni sekvencií je výrazne nižšia, a zároveň sa líši aj trend podobnosti. Relatívny pokles percentuálnej podobnosti oproti nástroju BinDiff vykresľuje graf 5.5.



Obr. 5.5: Graf relatívnych rozdielov hodnôt podobností podľa nástroja BinDiff a jednotlivých úrovní v YaraZille

Graf 5.5 znázorňuje pokles podobnosti voči nástroju BinDiff na jednotlivých úrovniach abstrakcie s 95% intervalom spoľahlivosti. Úroveň basic blokov sa preukázala ako najbližšia výsledkom nástroja BinDiff, čo dokazujú aj priemery poklesu podobnosti v tabuľke 5.1. Výhodou úrovne sekvencií je, že tieto hodnoty prejavujú najnižšiu mieru odchýlky.

Úroveň	Priemer	Smerodajná odchýlka
Sekvencie	-88,90%	13,33%
Basic bloky	-38,45%	31,43%
Funkcie	-40,97%	35,83%

Tabuľka 5.1: Štatistické údaje relatívnych rozdielov hodnôt podobností voči BinDiff

5.2 Zisťovanie výskytu spoločného kódu rodín LockBit a Zeoticus

Analýza vzťahov rodín LockBit a Zeoticus v sekcii 4.3 odhalila, že prekryvy kódu boli súčasťou implementácie kryptografických algoritmov. Nebolo však jasné, či sa jedná o vlastný kód od tvorcov malvéru alebo prebratý z bežne rozšírenej knižnice či iného voľne dostupného zdroja. Možnosťou, ako sa priblížiť k pravde bolo zistenie výskytu spoločného kódu vo vzorkách iných kmeňov malvéru a neškodlivého softvéru. Nástroj Retrohunt od spoločnosti VirusTotal môže byť v takomto prípade veľmi užitočný. Umožňuje totiž vyhľadať vzorky pridané do rozsiahlej databázy VirusTotal za posledných 90 dní pomocou statických YARA pravidiel.

Na tento účel bolo vygenerované YARA pravidlo zo zachytených prekryvov kódu v YaraZille. Pre generovanie pravidiel bol použitý nástroj yaragen, vyvinutý a používaný interne v spoločnosti Avast. Vytvorené boli pravidlá zo spoločných sekvencií, basic blokov a funkcií. Vo výpise 5.1 je uvedené iba pravidlo vygenerované zo sekvencií, ostatné sú obsahom priloženého pamäťového média. Pravidlá obsahovali iba hexadecimálne reťazce prislúchajúce spoločnej časti kódu a podmienku na výskyt určitého počtu reťazcov. Pravidlo vygenerované zo sekvencií malo najprísnejšiu podmienku, pretože požadovalo výskyt 10 zo 14 reťazcov, ďalšie pravidlá požadovali výskyt 4 z 9 basic blokov a 2 zo 6 funkcií a boli v nich nahradené konkrétne registre, adresy a ofsety v operandoch inštrukcií zástupnými znakmi.

```
rule rule_2022_03_08_21_19_30_yarazilla_seqs
{
    strings:
        $seq_0_0 = { 00 8D 57 FF 8D 8C 1D 7B FF FF FF 03 D3 3B F9 77 }
        $seq_0_1 = { 04 07 8D 7F 01 32 47 FF 41 88 44 37 FF 33 C0 3B }
        $seq_0_2 = { 28 E6 89 95 40 FF FF FF 89 95 38 FF FF FF 83 C2 }
        $seq_0_3 = { 2F 77 04 85 F6 74 29 8D 8D 34 FF FF FF 2B D9 0F }
        $seq_0_4 = { 36 26 33 FF F1 80 65 FF 29 79 4A FF EC 4E 9B 00 }
        $seq_0_5 = { 4D C4 89 48 34 5F 5E 5B 8B E5 5D C3 CC 55 8B EC }
        $seq_0_6 = { 57 8B 7D 10 0B C7 89 5D DC 89 75 B8 89 BD 7C FF }
        $seq_0_7 = { 72 09 83 FA 40 0F 83 4E FE FF FF 89 7D E4 85 C0 }
        $seq_0_8 = { 88 04 0B 83 FA 40 72 EA 8B 7D BC 8B 5D E4 EB 41 }
        $seq_0_9 = { 88 5E 0C 8B C2 88 4E 13 C1 F8 07 88 46 14 8B C2 }
        $seq_0_10 = { 8B C1 89 4D DC C1 F8 1A 03 D8 8B CB 89 5D D0 8B }
        $seq_0_11 = { 8B C2 C1 F8 0D 88 46 08 8A C3 C0 E0 05 C1 FA 15 }
        $seq_0_12 = { 8D 42 01 0F 45 C2 8B 55 DC 83 C2 C0 89 41 34 8B }
        $seq_0_13 = { FB 13 0A D8 88 56 09 8B 55 F0 8B C1 C1 F8 02 88 }
        $seq_0_14 = { FB 83 C4 04 83 F7 07 4F C1 EF 1F 8B 4D F8 8D 96 }
    condition:
        10 of ($seq_0_*)
}
```

Výpis 5.1: YARA pravidlo vygenerované zo sekvencií spoločného kódu LockBit a Zeoticus, použité pre zisťovanie výskytu kódu nástrojom Retrohunt

Všetky vytvorené pravidlá boli použité v nástroji Retrohunt, pričom najviac vzoriek bolo zachytených pravidlom vygenerovanom z funkcií. Zoznam zachytených vzoriek je zahrnutý na priloženom pamäťovom médiu. Dokopy bolo zachytených 621 vzoriek, ktoré sa dali podľa množstva antivírusových detekcií rozdeliť do dvoch hlavných skupín – malvér a ostatné. Za malvér sa dá považovať 187 vzoriek, ktoré boli identifikované nasledovne:

- LockBit (48),
- Zeoticus (14),

- neidentifikovaný (10),
- poškodený (115).

Najväčšiu časť vzoriek tvorili poškodené škodlivé súbory, u ktorých by sa dalo polemizovať, či naozaj patria medzi malvér, keďže vďaka poškodeniu nepredstavujú hrozbu. 10 vzoriek sa nepodarilo identifikovať, no bolo pri nich detegované správanie typické pre malvér. Ako bolo očakávané, zachytené boli vzorky rodín LockBit a Zeoticus, vrátane tých, ktoré boli skúmané v rámci tejto práce. Z výsledkov vyplýva, že hľadaný kód bol z identifikovaných kmeňov malvéru zaznamenaný iba vo vzorkách týchto dvoch rodín.

Zvyšných 434 vzoriek s najväčšou pravdepodobnosťou nepatrí medzi malvér. Zachytené boli najmä vzorky rôznych nástrojov. Podľa názvu boli identifikované ako:

- Hyperview(362),
- DWS inštalátor a klient (55),
- Roblox (7),
- SciChart DLL (4),
- eduVPN DLL (2),
- dnscryptproxy (1),
- Bitrix24 (1),
- BDisk (1),
- TinyPNG PhotoShop plug-in DLL (1).

Najväčšie zastúpenie mal nástroj Hyperview[1] pre dátové centrá, no objavili sa tu aj niektoré verzie hry Roblox[2]. Implementácia kryptografických funkcií teda nie je unikátna pre rodiny malvéru LockBit a Zeoticus, ale objavuje sa v množstve neškodlivého softvéru. Zároveň to znamená, že sa nejedná o rovnaké rodiny a pravdepodobne medzi nimi nie je hlbší vzťah ako použitie kódu prebratého z rovnakého zdroja. YaraZilla síce odhalila pri analýze v sekcii 4.3 medzi vzorkou LockBit a rodinou Zeoticus najväčšiu zhodu, no keďže bolo pomocou nástroja Retrohunt zachytených 48 vzoriek rodiny LockBit, záverom je, že tieto vzorky jednoducho nie sú zastúpené v databáze nástroja YaraZilla.

5.3 Návrh detekčných pravidiel

Vďaka zisteniam hĺbkovej analýzy bolo možné vytvoriť detekčné pravidlá v jazyku YARA, ktorý je popísaný v časti 2.2.4. Boli navrhnuté pre kmene s doposiaľ neznámymi vzťahmi, preskúmané v kapitole 4. Pre kmene Padodor a Kraton bolo navrhnuté jedno pravidlo, lebo na základe ich analýzy boli zlúčené do kmeňa Padodor. Rozsiahlejšie pravidlá boli navrhnuté pre rodinu Zeoticus, ktorá predtým nemala pravidlo v rámci klasifikácie vzoriek spoločnosti Avast. Tieto pravidlá dokážu efektívne detegovať nielen skúmané vzorky, ale aj tie, ktoré sú im podobné, a prispieť tak k zlepšeniu ich klasifikácie a detekcie.

5.3.1 Rodina Padodor

Na základe analýzy vzoriek rodiny Padodor v sekcii 4.2 bolo vytvorené behaviorálne YARA pravidlo. Zachytáva všetku odhalenú funkcionalitu skúmaných vzoriek.

[1]Hyperview – https://www.hyperviewhq.com/
[2]Roblox – https://www.roblox.com/

```
rule padodor_known_behavior
{
    condition:
        (
            cuckoo.sync.mutex(/^QKK_HT$/) or // názvy mutexov
            cuckoo.sync.mutex(/^kkcc$/) or
            cuckoo.sync.mutex(/^karton$/) or
            cuckoo.sync.mutex(/^Engel$/) or
        ) and
        cuckoo.process.executed_command(/:\\Windows\\System32\\[A-Z]{1}[a-z]{5,7}[0-9]{0,2}$
            /i)
}
```

Výpis 5.2: Behaviorálne YARA pravidlo pre rodinu Padodor

Analyzované vzorky rodiny Padodor obsahovali mutexy s názvami `QKK_HT`, `kkcc`, `karton` alebo `Engel`. Spúšťaný súbor v systémovom priečinku mal názov pozostávajúci zo 6 až 8 písmen, z ktorých bolo začiatočné písmeno veľké, prípadne nasledovaných dvoma číslicami.

5.3.2 Rodina Zeoticus

```
import "cuckoo"

rule zeoticus_known_behavior
{
    condition:
        (
            (
                cuckoo.filesystem.file_write(/^C:\\READ_?ME\.html$/i) or // zápis odkazu
                cuckoo.filesystem.file_write(/\\Desktop\\READ_?ME\.html$/i)
            ) and
            (
                cuckoo.filesystem.file_write(/\.zeoticus2$/) or // prípona šifrovaného súboru
                cuckoo.filesystem.file_write(/\.young$/) or
                cuckoo.filesystem.file_write(/\.pandora$/) or
                cuckoo.filesystem.file_write(/\.2020END$/)
            )
        ) or
        (
            cuckoo.filesystem.file_access(/.{1,70}\.exe:Zone\.Identifier$/i) and
            cuckoo.process.executed_command(/\\cmd\.exe \/c ping localhost -n [0-9]{1,10} >
                nul & del \"?.{1,70}\.exe\"?$/i) and
            cuckoo.registry.key_read(/^HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\
                Control\\Nls\\CustomLocale$/i) and
            cuckoo.registry.key_access(/^HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\
                CurrentVersion\\Run$/i) and
            cuckoo.registry.key_write(/^HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\
                CurrentVersion\\Run\\.{1,70}\.exe$/i)
        ) or
        (
            cuckoo.process.executed_command(/^\"?C:\\Windows\\.{1,70}\.exe\"? p r i v e t$/)
                 and
            cuckoo.process.executed_command(/^\"?C:\\Windows\\.{1,70}\.exe\"? p r i v e t1$/
                )
        )
}
```

Výpis 5.3: Behaviorálne YARA pravidlo pre rodinu Zeoticus

Behaviorálne pravidlo 5.3 bolo vytvorené najmä na základe dynamickej analýzy vzorky Zeoticus. Všetky časti podmienky pravidla využívajú regulárne výrazy. Prvá časť sa zameriava na zachytenie zapisovania súboru s odkazom pre obeť (*ransom note*) s danou cestou a šifrovaných súborov, ktoré môžu mať jednu z uvedených prípon. Ďalšia časť pravidla odráža viaceré techniky používané počas behu, ako je mazanie `Zone.Identifier` vlastnému spustiteľnému súboru, ktorý indikuje, že súbor bol stiahnutý z internetu. Detekuje tiež vymazanie spustiteľného súboru pomocou príkazu `ping`. Malvér zisťuje predvolený jazyk z kľúča v *registri* a pre spustenie pri prihlásení používateľa zapisuje reťazec so svojou cestou do kľúča `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`. Poslednou časťou pravidla je detekcia spustenia s parametrami, ktoré sú špecifické pre túto rodinu, ako „`p r i v e t`“ a „`p r i v e t1`“.

```
rule zeoticus_known_sequences
{
    strings:
        $s0_0 = "/c ping localhost -n 3 > nul & del %s" // príkaz na zmazanie vzorky
        $s0_1 = "p r i v e t1" wide // špecifický parameter spustenia
        $s0_2 = "p r i v e t2" wide // špecifický parameter spustenia
        $s1_0 = "ZEOTICUSV2"
        $s1_1 = "README.html" wide // názov ransom note
        $s1_2 = "%s%x%x%x%x.zeoticus2" wide // formátovací reťazec šifrovaného súboru
        $s1_3 = ".%s.%s%s" wide // alternatívny formátovací reťazec šifrovaného súboru
        $s1_4 = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion" wide // kľúč v registri
        $s2_0 = "Dear %s" wide // odkaz pre obeť aplikovaný na pracovnú plochu
        $s2_1 = " All your files has been encrypted " wide
        $s2_2 = " Toss a coin to us and get decryptor tool" wide
        $s2_3 = " All information how to make it you can find in " wide
        $s2_4 = " README file" wide
        $s3_0 = "zeoticus@tutanota.com" wide // e-mailové adresy útočníkov
        $s3_1 = "zeoticus@aol.com" wide
        $s3_2 = "zeoticus@protonmail.com" wide
        $s3_3 = "immunityyoung@aol.com" wide
        $s3_4 = "immunityyoung@tutanota.com" wide
        $s3_5 = "immunityyoung@protonmail.com" wide
        $s3_6 = "outsourse@tutanota.com" wide
        $s3_7 = "outsourse@cock.li" wide
        $s3_8 = "anobtanium@tutanota.com" wide
        $s3_9 = "anobtanium@cock.li" wide
    condition:
        EXE and
        (
            all of ($s0_*) or
            3 of ($s1_*) or
            all of ($s2_*) or
            2 of ($s3_*)
        )
}
```

Výpis 5.4: Statické YARA pravidlo pre rodinu Zeoticus

Statické pravidlo 5.4 sa zameriava na nezašifrované reťazce obsiahnuté v skúmanej vzorke. Medzi nimi sú aj e-mailové adresy, ktoré sa používajú v názve zašifrovaných súborov a v odkaze pre obeť (*ransom note*). Obsahuje tiež príkaz používaný na zmazanie spustiteľného súboru, parametre spustenia a formátovacie reťazce pre premenovanie šifrovaných súborov. Pravidlo funguje iba pre spustiteľné súbory, aby sa zamedzilo detekcii v iných súboroch akým je aj *ransom note*.

```
rule zeoticus_ransom_note_known_sequences
{
    strings:
        $s0_00 = "----===Zeoticus========" // začiatok ransom note staršej verzie
        $s0_01 = "All your data are encrypted."
        $s0_02 = "Only we can decrypt your data, write to the original mails specified in
            this file, otherwise you will become a~victim of scammers"
        $s0_03 = "Be carefully, recovery companies usually require more than we, and act as
            middleman"
        $s0_04 = "Contact and send this file to us:"
        $s0_05 = "zeoticus@tutanota.com"
        $s0_06 = "zeoticus@aol.com"
        $s0_07 = "zeoticus@protonmail.com"
        $s1_00 = "----===Zeoticus 2.0=======" // začiatok ransom note novšej verzie
        $s1_01 = "WARNING!"
        $s1_02 = "I am truly sorry to inform you that all your important files are crypted."
        $s1_03 = "If you want to recover your encrypted files you need to follow a few steps
            ."
        [                                VYNECHANÉ REŤAZCE                                ]
        $s1_20 = "immunityyoung@aol.com" // e-mailové adresy útočníkov
        $s1_21 = "immunityyoung@tutanota.com"
        $s1_22 = "immunityyoung@protonmail.com"
        $s1_23 = "outsourse@tutanota.com"
        $s1_24 = "outsourse@cock.li"
        $s1_25 = "anobtanium@tutanota.com"
        $s1_26 = "anobtanium@cock.li"
    condition:
        not EXE and
        (
            5 of ($s0_*) or
            20 of ($s1_*)
        )
}
```

Výpis 5.5: Statické YARA pravidlo pre *ransom note* rodiny Zeoticus

Statické pravidlo 5.5 je určené na detekciu *ransom note* súboru rodiny Zeoticus, funguje preto iba pre súbory, ktoré nie sú spustiteľné. Obsahuje reťazce s inštrukciami pre vykonanie vymáhanej platby a e-mailové adresy útočníka. Uvedené pravidlo je skrátené o reťazce `$s1_04` až `$s1_19`, ktoré obsahujú celú dĺžku inštrukcií k platbe.

5.4 Zhodnotenie analýzy nástrojom YaraZilla

Analýza nástrojom YaraZilla priniesla hodnotné výsledky z hľadiska hľadania podobnosti v kóde malvéru, prezentované v predošlých častiach práce. Pri práci s nástrojom boli tiež úspešne odhalené mnohé problémy nástroja, ktoré boli prešetrené a následne vyriešené. Niektoré z nich však pretrvávali počas celého výskumu a komplikovali jeho priebeh. Nasledujúce časti problémy popisujú spolu s návrhmi na ich riešenie a celkovo hodnotia dosiahnuté výsledky.

5.4.1 Odhalené problémy pri analýze nástrojom YaraZilla

Pri analýze rodín bankerov v sekcii 3.2 boli podobnosti niektorých vzoriek na úrovni sekvencií prvotne výrazne nižšie ako očakávané, aj vzhľadom k úrovni basic blokov a funkcií. Po prešetrení súčastí nástroja YaraZilla bol odhalený problém s filtrovaním sekvencií. Metóda

používaná pri predspracovaní na odfiltrovanie neškodlivých (*clean*) sekvencií, spomínaná v časti 2.3.3, v niektorých prípadoch označila väčšinu sekvencií vzorky ako *clean*. To sa stávalo aj keď takmer identická vzorka nemala podobný problém. Na základe tohto problému musela byť analýza rodín bankerov zopakovaná po vypnutí tejto filtračnej metódy. Problém bol prešetrený a bolo odhalené, že dôvodom bolo označenie kódu všetkých poškodených spustiteľných súborov ako *clean*. Kód, ktorý sa nachádzal v poškodenom malvéri bol tak odfiltrovaný a nevyskytoval sa vo výsledku analýzy. Ako riešenie bolo navrhnuté kontrolovať okrem výskytu aj prevalenciu kódu medzi čistými vzorkami. Do doby vyriešenia problému bola filtračná metóda vypnutá.

Problém s iným druhom filtrácie sa objavil pri analýze vzoriek rodín WannaCry a Contopee v sekcii 3.3. YaraZilla nedokázala odhaliť podobnosť na úrovni sekvencií, pretože tieto rodiny neobsahovali sekvencie spoločnej funkcie. Ako bolo pri analýze spomenuté, problém spôsobilo filtrovanie prostredníctvom F.L.I.R.T, spomenuté v časti 2.3.3. Príčinou bola konkrétne chybná signatúra použitá pri filtrovaní, čo bolo následne napravené.

Istý problém vo filtrácii sa prejavil aj pri extrakcii verzií rodiny spora v sekcii 3.4, kedy jedna verzia rodiny v YaraZille obsahovala z neznámeho dôvodu „funkciu“, ktorá skutočnosti bola extrahovaná z dát.

```
add byte ptr [eax], al
add byte ptr [eax], al
add byte ptr [eax], al
add byte ptr [eax], al
add byte ptr [eax], al
add byte ptr [eax], al
add byte ptr [eax], al
```

Výpis 5.6: Nezmyselný kód časti „funkcie“ extrahovanej z dát vzorky Spora

Extrahovaná „funkcia“ mala 412 bytov, začiatok z jej kódu je na výpise 5.6. Uvedené inštrukcie nie sú súčasťou žiadneho algoritmu, ale vznikli chybnou interpretáciou dát. Inštrukcie `add byte ptr [eax], al` majú binárnu hodnotu `0x0000` v hexadecimálnej podobe. Obrázok 5.6 ukazuje začiatok týchto dát nachádzajúcich sa na adrese `0x406c08`.

```
.text:00406C08 ; void *(__cdecl *memcpy)(void *, const void *Src, size_t Size)
.text:00406C08 memcpy          dd 0                    ; DATA XREF: sub_40575E+3B↑r
.text:00406C08                                         ; start+A5↑w
.text:00406C0C ; void *(__cdecl *memset)(void *, int Val, size_t Size)
.text:00406C0C memset          dd 0                    ; DATA XREF: sub_404E93+15↑r
.text:00406C0C                                         ; sub_404F25+1D↑r ...
.text:00406C10 ; int RtlComputeCrc32
.text:00406C10 RtlComputeCrc32 dd 0                    ; DATA XREF: sub_406118+D9↑r
.text:00406C10                                         ; sub_406118+1B8↑r ...
.text:00406C14 ; HCRYPTKEY hKey
.text:00406C14 hKey            dd 0                    ; DATA XREF: sub_406118+188↑r
.text:00406C14                                         ; start+192↑w ...
.text:00406C18 ; HCRYPTKEY phKey
.text:00406C18 phKey           dd 0                    ; DATA XREF: sub_40533C+76↑r
.text:00406C18                                         ; sub_406073+77↑o ...
.text:00406C1C ; DWORD VolumeSerialNumber
.text:00406C1C VolumeSerialNumber dd 0                 ; DATA XREF: start+4F↑o
.text:00406C1C                                         ; start+BC↑r ...
.text:00406C20 ; HANDLE hObject
.text:00406C20 hObject         dd 0                    ; DATA XREF: sub_405D62+12↑r
.text:00406C20                                         ; sub_405D62+33↑r ...
```

Obr. 5.6: Časť dát vzorky Spora, odkiaľ bola extrahovaná „funkcia“

Pri procese tvorby párov, ktorý opisuje sekcia 3.5, bolo zaznamenané množstvo duplicitných rodín. Túto skutočnosť zapríčinil spôsob vkladania vzoriek do databázy nástroja YaraZilla, pretože vzorky pochádzajú z rôznych zdrojov, kde ich rodiny môžu mať aj rôzne pomenovanie. Na nedostatok bolo týmto spôsobom upozornené a začalo sa pracovať na jeho riešení. Verzia 2.0 zredukovala tento problém na minimum, pretože priniesla ucelený spôsob pomenovania rodín identifikátorom, ktorý je popísaný v podsekcii 3.5.3.

Hlavným problémom pri pároch bol ale staticky linkovaný kód, ktorý nebolo možné odfiltrovať. Príkladom sú rodiny využívajúce knižnicu MFC, spomenuté v časti 4.4.1, ale objavili sa viaceré ktoré obsahovali systémové knižnice jazykov Delphi, C++, knižnice OpenSSL, libcurl alebo zlib. Staticky linkovaný kód by mal byť aspoň sčasti odfiltrovaný technológiou F.L.I.R.T., no to sa ukázalo ako nedostatočné. To bolo prekážkou pri odhalení väčšieho množstva neznámych prekryvov rodín. Takýto kód nielen skresľuje výsledky pri hľadaní podobností vo vlastnom kóde – pretože niekedy sa nedá presne určiť pôvod, ako tomu bolo pri rodinách LockBit a Zeoticus – ale často aj samotné množstvo zhodných funkcií komplikuje hĺbkovú analýzu, a tým je ťažšie odhaliť medzi nimi vlastný kód.

Vylepšenie detekcie staticky linkovaného kódu by mohla priniesť integrácia súčastí dekompilátora RetDec[3], ktoré sú schopné ho detegovať. Pri použití nástroja RetDec boli totiž zaznamenané lepšie experimentálne výsledky oproti aktuálnej implementácii. Rovnako ako YaraZilla, je vyvíjaný v spoločnosti Avast, takže pri spolupráci vo vývoji sa môžu tieto nástroje navzájom zlepšovať. Integrácia dekompilátora je plánovaná do ďalšej verzie YaraZilly. Plánovaná je tiež možnosť označenia sekvencií, basic blokov alebo funkcií v YaraZille ako súčasť staticky linkovaného kódu, ktorá by doplnila možnosti označenia neškodlivého (*clean*) kódu a úplného odstránenia z databázy.

5.4.2 Zhrnutie

V priebehu celého výskumu bolo možné pozorovať, že na úrovni sekvencií nástroja YaraZilla boli zaznamenané najnižšie hodnoty podobností, čo poukazuje na nedostatočnú filtráciu. Príčinou mohli byť aj rôzne parametre extrakcie pri napĺňaní databázy, ktoré spôsobili rozdiely v počte a unikátnosti sekvencií rodiny. Príkladom môže byť extrakcia viacerých vzoriek, kedy bol zvolený len prienik ich sekvencií, čo by ich zredukovalo na spoločné – charakteristické sekvencie rodiny, a extrakcia jednej vzorky, z ktorej boli vybrané všetky jej sekvencie. Kolekcia sekvencií takýchto rodín by vznikla iným spôsobom a rozdiely by sa prejavili pri zisťovaní podobnosti. V prípade analýzy rodiny Spora v štúdii C – sekcia 3.4 – bola každá verzia uložená ako samostatná rodina, čiže pri extrakcii boli vždy vybrané všetky sekvencie. To kvôli nedostatočnej filtrácii spôsobilo, že verzie rodiny obsahovali množstvo nadbytočných sekvencií, medzi ktorými nebola nájdená podobnosť. Priestor na zlepšenie je teda vo filtrácii sekvencií a zjednotení parametrov extrakcie pri napĺňaní databázy.

Preukázalo sa, že YaraZilla je na úrovni funkcií citlivá na zmeny počtu inštrukcií, čo bolo pozorované pri analýze kmeňov WannaCry a Contopee alebo aj verzie rodiny Spora, ktorá obsahovala ladiace výpisy. Dôvodom bolo, že štrukturálna podobnosť funkcií sa v súčasnosti určuje zo zhodných basic blokov. Vhodným vylepšením by bolo, ak by sa určovala z postupnosti inštrukcií, kde by sa drobné zmeny neodrazili v takom výraznom prepade podobnosti, ako pri poslednej verzii rodiny Spora.

Najspoľahlivejšie sa ukázali byť podobnosti na úrovni basic blokov, pretože v porovnaní so sekvenciami prešli väčšou mierou filtrácie, a nie sú tak citlivé na drobné zmeny ako na

[3]RetDec (Retargetable Decompiler) – https://retdec.com/

úrovni funkcií, čo sa potvrdilo pri skúmaní vzťahu rodín WannaCry a Contopee v štúdii B aj najmenšou odchýlkou od nástroja BinDiff pri analýze verzií Spora.

Jedným zo zistení celkovej práce s nástrojom YaraZilla je, že nájsť známe vzťahy medzi rodinami nie je väčšinou problém, ale hľadať neznáme vzťahy je už ťažšie. Spôsobil to hlavne spomínaný problém detekcie staticky linkovaného kódu. Jeho odhalenie vzbudilo diskusiu, či by takýto kód mal vôbec byť v databáze YaraZilly. A to aj napriek tomu, že existujúce metódy nástroja takýto kód úplne odfiltrujú. Je faktom, že pri podobnosti binárnych spustiteľných súboroch zohráva veľkú úlohu, ale ak sa zameriame na malvér, prináša viac problémov ako úžitku. Ak je predmetom skúmania kód vlastnej implementácie, skresľuje dosiahnuté výsledky. Kompromisom bol návrh už spomenutého rozšírenia, ktoré by umožnilo takýto kód označiť ako staticky linkovaný. Skutočným riešením by sa však mohla ukázať integrácia súčastí dekompilátora RetDec.

Táto práca mala hlavne výrazný prínos pre vývoj nástroja YaraZilla, pretože upozornila na jeho nedostatky a podnietila snahu o ich riešenie. Uskutočnené aj naplánované vylepšenia sú správnym krokom do budúcnosti, aby tak nástroj mohol lepšie preukázať svoj potenciál v praxi.

Kapitola 6

Záver

Cieľom tejto práce bolo hĺbkovo analyzovať vzťahy kódu vo vybraných kmeňoch malvéru pomocou metód reverzného inžinierstva a pokúsiť sa odhaliť ich pôvod. To vyžadovalo hromadnú analýzu podobnosti malvéru, na základe ktorej sa uskutočnil výber týchto kmeňov. Deklarované ciele boli touto prácou splnené a v niektorých ohľadoch presiahnuté.

Na začiatku výskumu bolo kľúčové sa oboznámiť s problematikou malvéru, jeho analýzy a dostupnými nástrojmi schopnými detekcie podobnosti binárneho kódu. Pre účely tejto práce bol zvolený nástroj YaraZilla, ktorý bol vyvinutý v rámci diplomovej práce v spolupráci so spoločnosťou Avast, a teda táto práca na ňu týmto spôsobom nadväzuje.

Prvým krokom analýzy pomocou nástroja YaraZilla bolo preskúmanie jeho použitia v praxi pri analýze známych vzťahov medzi kmeňmi malvéru. ajskôr sa zamerala na 98 vzoriek rôznych verzií kmeňov odvodených zo Zeus, pričom pozorovala, či boli medzi nimi nájdené podobné vzťahy ako pri existujúcom výskume. Ďalej bol analýze podrobený publikovaný vzťah medzi kmeňmi WannaCry a Contopee, kedy bolo predmetom záujmu odhalenie konkrétnych prekryvov v dvojici skúmaných vzoriek. Skúmaním známych vzťahov bolo zistené, že YaraZilla ich tiež dokáže odhaliť, no zároveň boli identifikované limitácie nástroja ako obsah databázy rodín, či citlivosť detekcie podobnosti pri pridaní a odobraní inštrukcií.

Nasledujúcim použitím YaraZilly bolo hľadanie vzťahov, z ktorých boli niektoré vybrané pre hĺbkovú analýzu. Spočívalo z dvoch častí – podobnosti obmedzené na určitú skupinu vzoriek a vzorky všetkých kmeňov databázy. Pre prvú časť bola určená ransomware rodina Spora, kedy sa skúmal vývoj ich verzií naprieč 18 rôznymi vzorkami. V druhej časti sa hľadali vzťahy v kmeňoch zastúpených v databáze nástroja YaraZilla, pričom z nich boli vytvorené páry na základe podobnosti. Pri prvej iterácii bolo spracovaných 614 kmeňov, pri druhej sa ich počet vyšplhal na 2 046. Okrem toho boli s obsahom databázy porovnávané vzorky z viacerých zbierok malvéru. Zo všetkých spomínaných zdrojov potom boli vybrané vzorky pre hĺbkovú analýzu.

Následná hĺbková analýza najskôr skúmala prekryvy vo vzorkách rodiny Spora, z ktorých bolo 9 hlavných verzií podrobne popísaných. Analýza odhalila vzťahy medzi týmito verziami vo vlastnom kóde a priniesla tak komplexnejšie pochopenie vývoja rodiny a zmýšľania autorov. Toto sú poznatky, ktoré sa môžu ukázať ako prínosné aj pri skúmaní budúcich hrozieb. Vo vzorkách kmeňov Padodor a Kraton bola odhalená kompletná zhoda, na základe čoho boli zlúčené do jedného kmeňa. Tiež bol zistený prekryv kódu vo vzorkách rodín LockBit a Zeoticus, ktorý spočíval v podobnej implementácii kryptografických algoritmov. Tento kód nemá známy pôvod, ale bol zrejme prebratý, keďže bolo zistené, že sa vyskytoval v množstve čistých spustiteľných súborov. Pre Padodor a Zeoticus boli na základe zis-

tení hĺbkovej analýzy navrhnuté detekčné vzory, ktoré dokážu efektívne detegovať skúmané a taktiež aj podobné novo zachytené vzorky.

Počas analýzy nástrojom YaraZilla bolo úspešne odhalených viacero nedostatkov, ktoré boli prešetrené a niektoré z nich aj napravené. U mnohých párov kmeňov vytvorených pri analýze bolo zistené, že dôvod ich podobnosti spočíva v použití rovnakých knižníc, ktorých kód bol staticky prilinkovaný. To predstavovalo problém, ktorého postupné riešenie bolo naplánované do budúcnosti. Kvôli rozsahu problému nebolo možné z párov vybrať ďalšie doposiaľ neznáme vzťahy, pretože takýto kód vo veľkej miere skresľuje výsledky.

Pri pohľade do budúcnosti sa na základe súčasného vývoja dá očakávať neustály prísun nových kmeňov malvéru či ich variantov. Nadobúdanie poznatkov aj o ich vzájomných vzťahoch sa ukazuje ako prínosné pri zdokonaľovaní klasifikácie a zároveň ochrany voči aktuálnym aj novo odhaleným hrozbám. Možnosť pokračovania tejto práce teda vidíme najmä vo vylepšovaní nástroja YaraZilla. Hlavným cieľom sa javí zlepšenie detekcie staticky linkovaného kódu a filtrovania extrahovaných sekvencií. Podobnosť na úrovni sekvencií sa ukázala ako najmenej efektívna pre odhalenie vzťahov rôznych kmeňov, ale na druhej strane netrpela v takej miere problémom so staticky linkovaným kódom. Obe vylepšenia majú potenciál výrazne zväčšiť efektivitu YaraZilly. Tieto návrhy boli prediskutované s autorom nástroja a boli naplánované kroky pre ich uskutočnenie.

Literatúra

[1] Avast. *Remembering WannaCry* [online]. 2020 [cit. 2022-01-22]. Dostupné z: https://blog.avast.com/remembering-wannacry-avast.

[2] Bernstein, D. J. The Salsa20 Family of Stream Ciphers. In: *New Stream Cipher Designs: The eSTREAM Finalists*. 1. vyd. Berlin, Heidelberg: Springer, 2008, s. 84–97. ISBN 978-3-540-68350-6.

[3] Cohen, R. a Walkowski, D. *Banking Trojans: A Reference Guide to the Malware Family Tree* [online]. 2019 [cit. 2021-12-31]. Dostupné z: https://www.f5.com/labs/articles/education/banking-trojans-a-reference-guide-to-the-malware-family-tree.

[4] Eilam, E. *Reversing: Secrets of Reverse Engineering*. 1. vyd. Indianapolis: Wiley, 2005. ISBN 978-0-7645-7481-8.

[5] FireEye. *Report APT38 – Un-usual Suspects* [online]. 2018 [cit. 2022-04-29]. Dostupné z: https://content.fireeye.com/apt/rpt-apt38.

[6] Kaspersky. *WannaCry and Lazarus Group – the missing link?* [online]. 2017 [cit. 2022-04-29]. Dostupné z: https://securelist.com/wannacry-and-lazarus-group-the-missing-link/78431/.

[7] Kironský, E. a Křoustek, J. *Spora: New Kid On The Block*. CARO2017, Kraków, máj 2017 [cit. 2022-04-29].

[8] Kironský, E. a Křoustek, J. *Spora: The Saga Continues*. VB2017, Madrid, október 2017 [cit. 2022-04-29].

[9] Kubov, P. *Scalable Binary Executable File Similarity*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

[10] Liska, A. a Gallo, T. *Ransomware: Defending Against Digital Extortion*. 1. vyd. Sebastopol: O'Reilly Media, 2016. ISBN 978-1-491-96788-1.

[11] Microsoft. *Application Information and Management* [online]. 2021 [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/cpp/mfc/reference/application-information-and-management?view=msvc-170.

[12] Microsoft. *Mutex Objects* [online]. 2021 [cit. 2021-12-16]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/sync/mutex-objects.

[13] Microsoft. *PE Format – Win32 apps* [online]. 2021 [cit. 2022-01-17]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

[14] Microsoft. *OSVERSIONINFOEXW structure* [online]. 2022 [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_osversioninfoexw.

[15] Pasca, V. *A Detailed Analysis of The LockBit Ransomware* [online]. 2022 [cit. 2022-04-29]. Dostupné z: https://lifars.com/wp-content/uploads/2022/02/LockBitRansomware_Whitepaper.pdf.

[16] Plohmann, D. *The Big Zeus Family Similarity Rundown v2 (2018-05-22, by pnX)* [online]. 2018 [cit. 2021-11-28]. Dostupné z: https://pnx.tf/slides/zeus_similarity_showdown.html.

[17] Saarine, M.-J. a Aumasson, J.-P. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)* [Internet Requests for Comments]. RFC 7693. RFC Editor, február 2015. Dostupné z: https://www.rfc-editor.org/rfc/rfc7693.html.

[18] Sikorski, M. a Honig, A. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. 12. vyd. San Francisco: No Starch Press, 2012. ISBN 978-1-59327-290-6.

[19] Szor, P. *The Art of Computer Virus Research and Defense*. 1. vyd. Boston: Addison-Wesley, 2005. ISBN 0-321-30454-3.

[20] Walter, J. *WannaCry and Lazarus Group – the missing link?* [online]. 2021 [cit. 2022-04-29]. Dostupné z: https://www.sentinelone.com/labs/zeoticus-2-0-ransomware-with-no-c2-required/.

[21] *YARA 4.1.3 documentation* [online]. 2021 [cit. 2022-01-17]. Dostupné z: https://yara.readthedocs.io/en/v4.1.3/.

Príloha A

Obsah priloženého pamäťového média

Priložené médium má nasledujúci obsah:

- `latex` – priečinok s LaTeX zdrojovými súbormi práce,
- `logs` – priečinok so súborom so záznamom ladiacich výpisov vzorky Spora,
- `reports` – priečinok s Cuckoo reportami pre behaviorálne pravidlá,
- `retrohunt` – priečinok s Retrohunt YARA pravidlom a zoznamom vzoriek,
- `rules` – priečinok s YARA pravidlami pre vzorky,
- `samples` – priečinok so zbalenými vzorkami malvéru,
- `similarities` – priečinok s tabuľkami podobností,
- `yara` – priečinok s YARA spustiteľným súborom pre Windows,
- `README.md` – súbor s inštrukciami a bližším popisom súborov,
- `rules_test.py` – Python skript na predvedenie funkčnosti YARA pravidiel,
- `xvosci00-Hloubkova-analyza-podobnosti-kodu-v-malware-kmenech.pdf` – PDF dokument práce,
- `xvosci00-Hloubkova-analyza-podobnosti-kodu-v-malware-kmenech_print.pdf` – PDF dokument práce určený na vytlačenie.

Príloha B

Tabuľky výsledkov analýzy kmeňov typu banker

Táto príloha obsahuje tabuľky výsledkov analýzy podobností z časti 3.2. Na obrázku B.1 sú vizualizované podobnosti podľa nástroja BinDiff. Obrázok B.2 zobrazuje podobnosti podľa nástroja YaraZilla na úrovni sekvencií, obrázok B.3 výsledky na úrovni basic blokov a B.4 na úrovni funkcií. Všetky obrázky obsahujú legendu k zafarbeniu jednotlivých políčok a prípadné chýbajúce údaje sú označené šedou farbou.

Obr. B.1: Tabuľka podobnosti rodín podľa nástroja BinDiff

Obr. B.2: Tabuľka podobnosti rodín na úrovni sekvencií podľa nástroja YaraZilla

Obr. B.3: Tabuľka podobnosti rodín na úrovni basic blokov podľa nástroja YaraZilla

Obr. B.4: Tabuľka podobnosti rodín na úrovni funkcií podľa nástroja YaraZilla

Príloha C

Detailný popis stavov a priebehu spustenia malvéru Spora

Predstavené stavy boli v texte práce iba stručne popísané. Pre analýzu verzií je však potrebné hlbšie porozumenie fáz útoku tohto malvéru. V tejto časti sú detailne popísané stavy, ktorými sa riadi prvá analyzovaná verzia rodiny Spora.

Stav 0

Tento stav nastáva vždy pri spustení programu. Program zisťuje písmeno systémového disku tým, že nájde cestu k `ntdll.dll`, ktoré sa bežne nachádza v `C:\Windows\SYSTEM32`. Pomocou `GetVolumeInformationW` získa *Volume Serial Number* tohto disku, s ktorým neskôr pracuje.

Súbor `ntdll.dll` využije ešte pri jednej činnosti – zisťovaní verzie Windows. Pomocou `GetFileInfoW` získa informácie o verzii súboru, z ktorých funkciou `VerQueryValueW` s parametrom `"\\"` ako `lpSubBlock` dostane `VS_FIXEDFILEINFO` štruktúru. Z nej vyberie `dwProductVersionMS`, čo je horných 32 bitov verzie produktu tejto systémovej knižnice, ktorá by zároveň mala zodpovedať verzii operačného systému. Z nej zodpovedá horných 16 bitov `dwMajorVersion` a spodných 16 bitov `dwMinorVersion` z dokumentácie Microsoftu [14]. Program skombinuje spodných 8 bitov a horných 24 bitov operáciou OR nasledovne:

```
return dwProductVersionMS >> 8 | BYTE(dwProductVersionMS);
```

Pre Windows 10 by podľa dokumentácie mal byť výsledok v hexadecimálnej sústave `0xA00`, pre Windows XP 64-bitovej verzie `0x502`, čo sa potvrdilo aj pri analýze.

Program kontroluje, či bol spustený s parametrom /u, ak áno, prechádza do časti programu, ktorá sa už neriadi stavmi. Toto správanie je neskôr popísané bližšie.

Na zaistenie behu iba jednej inštancie malvéru používa mutex, spomínaný v podsekcii 2.1.2, s názvom „`m{Volume Serial Number}`“. Ak mutex už existuje, program končí.

Hlavný RSA-1024 verejný kľúč je uložený v globálnej premennej, zašifrovaný AES-256 kľúčom z inej premennej. Pomocou spomínaného AES-256 kľúča sa dešifruje *hlavný* RSA-1024 verejný kľúč a HTML dokument s odkazom pre obeť (*ransom note*) a *identifikátor verzie*.

V tomto momente sa vytvára alebo otvára *stavový súbor* s cestou `%APPDATA%\{Volume Serial Number}` a číta sa z neho zapísaný stav. Ak je súbor prázdny alebo je stav neplatný,

vynuluje sa a do súboru sa zapíše stav 0. Podľa prečítaného stavu pokračuje stavom, ktorý po ňom nasleduje.

Stav 1

Stav zabezpečuje hľadanie napádaných súborov. Pomocou `CreateStreamOnHGlobal` sa vytvára *stream* na zapisovanie ciest súborov na šifrovanie. Následne sa volá funkcia, ktorá prechádza cez všetky disky. Funkcia získa zoznam dostupných diskov cez `GetLogicalDrives`, pre každý z nich zistí koreňovú cestu a jeho druh – či sa jedná o pevný, prenosný alebo sieťový disk. Pre každý disk tieto údaje odovzdáva funkcii, ktorá zostaví zoznam súborov. Ak sa jedná o pevný alebo prenosný disk, pomocou `FindFirstFileW` a `FindNextFileW` prechádza cez obsah disku, pri priečinkoch, ktoré neignoruje sa volá rekurzívne a pri súboroch po skontrolovaní, že patrí k danej kategórii, zapíše cestu k nemu do *streamu*.

Následne získava sieťové súbory na šifrovanie zo sieťových diskov, ktoré vymenuje vďaka `WNetEnumResourceW`. Pre každý sieťový disk volá funkciu, ktorá podobne ako pri lokálnych diskoch prechádza jeho obsah a zapíše cesty súborov do *streamu*.

Stream s cestami súborov ukladá do časti 1 *stavového súboru* a nakoniec doň zapisuje stav 1.

Stav 2

Stav generuje *záznam údajov o obeti*. Najskôr sa zo *stavového súboru* načíta *stream*, z ktorého sa čítajú cesty súborov. Pre každý súbor sa určí jeho kategória a počítadlo tejto kategórie v podobe globálnej premennej sa inkrementuje.

Pripravia sa potrebné kľúče – *hlavný* RSA-1024 verejný kľúč sa importuje z reťazca dešifrovaného počas stavu 0; a *lokálny* pár RSA-1024 kľúčov sa vygeneruje.

Počty súborov sa použijú pri vytváraní *záznamu o obeti* a *identifikátora obete* v podobe ako boli predstavené v predošlej podsekcii. Vygeneruje sa AES-256 kľúč, ktorým sa zašifruje *záznam údajov o obeti*, ten zapíše na koniec zašifrovaný *hlavným* RSA-1024 verejným kľúčom.

V priečinku `APPDATA` vytvorí súbor s názvom *identifikátora obete* a príponou KEY, ktorý následne prekopíruje do koreňových priečinkov pevných diskov, na pracovnú plochu a do priečinka `TEMPLATES`[1].

Verejný *lokálny* RSA-1024 kľúč sa zapíše do časti 2, *identifikátor obete* do časti 3 a nakoniec stav 2 do časti 0 *stavového súboru*.

Stav 3

Stav 3 obsluhuje šifrovanie súborov. Zo *stavového súboru* sa načíta *stream* s cestami súborov a *lokálny* RSA-1024 verejný kľúč. Program postupne podľa poradia prechádza cez kategórie súborov a každý súbor patriaci danej kategórii sa pokúsi zašifrovať. V prípade neúspechu šifrovanie opakuje 5-krát, vždy so 16 milisekundovým uspaním medzi pokusmi.

Proces šifrovania nezačne hneď, najprv si overí, či súbor nebol už predtým zašifrovaný – prečíta posledné 4 byty súboru, ktoré porovná s kontrolným súčtom CRC-32 predošlých 128 bytov. Pri súbore zašifrovanom touto vzorkou by sa mali porovnávané hodnoty rovnať.

[1] `TEMPLATES` – predvolene `%APPDATA%\Microsoft\Windows\Templates`

Ďalej potrebuje určiť veľkosť šifrovaných dát. Pri súboroch väčších ako 5 MiB sa šifruje iba prvých 5 MiB, pri menších sa zašifruje celý súbor okrem posledných 32 bytov.

Samotné šifrovanie prebieha nasledovne – pomocou `CreateFileMappingW` sa vytvorí nepomenovaný objekt mapovania súboru so spomínanou veľkosťou, funkcia `MapViewOfFile` sprítupní jeho obsah v pamäti. Takýmto spôsobom dokáže obsah súboru prepísať na mieste. V tomto momente vygeneruje AES-256 kľúč, ktorý exportuje ako text, následne ho zašifruje *lokálnym* RSA-1024 kľúčom. Pôvodným AES-256 kľúčom zašifruje obsah súboru a na samotný koniec súboru pomocou `WriteFile` zapíše zašifrovaný AES-256 kľúč vo veľkosti 128 bytov a jeho kontrolný súčet vo veľkosti 4 byty.

Prípona šifrovaných súborov sa nemení, no nastaví sa im atribút „iba pre čítanie“. Po prejdení všetkých kategórií zapisuje do *stavového súboru* stav 3.

Stav 4

V tomto stave sa generuje odkaz pre obeť (*ransom note*). *Identifikátor obete* je potrebné prečítať zo *stavového súboru*, zašifrovaný *záznam údajov o obeti* zas z príslušného súboru s príponou KEY. V stave 0 dešifrovaný obsah dokumentu je zapísaný do HTML súboru s nahradenými reťazcami *identifikátora obete* za `{key}` a zašifrovaný *záznam údajov o obeti* (kódovaný v Base64) za `{data}`.

Takýto dokument je umiestnený v priečinku `APPDATA` s názvom `identifikátora obete`, odkiaľ sa kopíruje na koreňové priečinky všetkých pevných diskov, na pracovnú plochu a do priečinkov `TEMPLATES`[2], `STARTUP`[3]. Umiestnenie v priečinku `STARTUP` spôsobí, že odkaz sa otvorí pri každom prihlásení daného používateľa.

Následne sa vytvára zoznam šifrovaných súborov. Na to musí program importovať *hlavný* RSA-1024 verejný kľúč z globálnej premennej a načítať *stream* s cestami súborov zo *stavového súboru*. Zoznam ciest súborov pretransformuje na cesty oddelené znakmi `"\r\n"` – CRLF, čo je oddeľovač riadkov v operačnom systéme Windows.

Zoznam v takomto formáte zašifruje vygenerovaným AES-256 kľúčom a zapíše do súboru s názvom *identifikátora obete*, príponou LST v priečinku `APPDATA`. AES-256 kľúč zapíše na koniec súboru zašifrovaný *hlavným* RSA-1024 verejným kľúčom, podobne ako v prípade súboru so `záznamom o obeti`. Výsledný súbor prekopíruje ešte na koreňové adresáre pevných diskov a do priečinka `TEMPLATES`.

Nakoniec nastaví všetkým súborom so *záznamom o obeti* vytváraným v stave 2 vlastnosť „iba na čítanie“ a uloží stav 4 do *stavového súboru*.

Stav 5

Program zisťuje, či beží na operačnom systéme Windows verzie Vista (resp. Server 2008) alebo novšej. Používa na to verziu vo formáte určenom v stave 0, ktorú porovnáva s hexadecimálnou hodnotou `0x600`, čo zopovedá verzii `6.0` [14]. Ak beží na vyššej alebo rovnej verzii, skontroluje, či má administrátorské oprávnenia. Podľa toho sa rozhoduje, či sa spustí ako administrátor s parametrom `/u`, alebo vymaže *Volume Shadow Copies*[4].

[2] `TEMPLATES` – predvolene `%APPDATA%\Microsoft\Windows\Templates`

[3] `STARTUP` – predvolene `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup`

[4] Volume Shadow Copy Service – systém zálohovania vo Windows pre súborové systémy NTFS a ReFS

Mazanie *Volume Shadow Copies* nastáva, aby sa nebolo možné po zašifrovaní vrátiť k pôvodným súborom obnovením takýchto záloh. Mazanie sa snaží utajiť spustením `cmd.exe`[5] prostredníctvom `wmic.exe`[6]. Maskovanie týmto nástrojom je často používané malvérom rôznych druhov. Príkaz, ktorý spolu s parametrami spúšťa cez `ShellExecuteExW` vyzerá nasledovne:

```
wmic.exe process call create
"cmd.exe /c
 vssadmin.exe delete shadows /all /quiet &
 bcdedit.exe /set {default} recoveryenabled no &
 bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures"
```

Vykonávané príkazy najskôr vymažú *Volume Shadow Copies*, vypnú automatické spustenie *Windows Error Recovery*[7] a nakoniec nastavia, aby boli pri spúšťaní Windows ignorované chyby, ktoré by viedli k spusteniu vo *Windows Error Recovery* režime. Poškodenie dôležitých súborov môže vyvolať spustenie v tomto režime, ktorý ponúkne používateľovi spustenie v núdzovom režime, preinštalovanie systému, či všeobecne zamedziť bežné spustenie.

V prípade chýbajúcich oprávnení sa pomocou `ShellExecuteExW` s parametrom `„runas"` ako `lpVerb` spúšťa svoj pôvodný súbor s administrátorskými oprávneniami a parametrom `/u`. Pri predvolených nastaveniach sa obeti zobrazí dialógové okno pýtajúce si oprávnenia a bez ich udelenia spustenie zlyhá. Pokus o spustenie sa opakuje dokopy 10-krát, pričom keď sa spustenie nepodarí, program pokračuje rovnako ako keby bolo úspešné, len nebeží súčasne jeho druhá inštancia.

Nezávisle od oprávnení sa program vždy pokúša odstrániť hodnotu `IsShortcut` z kľúča `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\lnkfile` v *registri*[8]. Toto okrem iného zapríčiní hlavne to, že súbory s príponou `lnk` – odkazy nebudú mať v ikone šípku, ako býva zvykom vo Windows. Pri predvolenej konfigurácii ale bez administrátorských oprávnení táto operácia neuspeje.

Po dokončení týchto úkonov program ukladá stav 5 do *stavového súboru*.

Stav 6

V stave 6 sa zobrazuje odkaz pre obeť. Z časti 3 *stavového súboru* číta *identifikátor obete*, následne sa súbor na pracovnej ploche s takýmto názvom a príponou HTML pokúša otvoriť predvoleným prehliadačom. Jedná sa o *ransom note* vygenerovaný v stave 4, na samotné otvorenie znova využíva `ShellExecuteExW`. Stav končí zapísaním stavu 6 do *stavového súboru*.

Stav 7

V poslednom stave sa šíri malvér prostredníctvom odkazov. Najskôr sa však samotnému spustiteľnému súboru vymaže *Alternate Data Stream*[9] `Zone.Identifier`, ktorým sú vo Windows označené súbory stiahnuté z internetu. Otváranie súborov s takýmto označením je nutné odsúhlasiť používateľom, čomu sa chcú útočníci vyvarovať.

[5] Command Prompt – predvolený interpret príkazov vo Windows
[6] WMIC (Windows Management Instrumentation Command-Line Utility) – nástroj na správu zariadení
[7] Windows Error Recovery – režim obnovy pri poškodení detegovanom pri spúšťaní Windows
[8] Windows Registry – systémová databáza Windows, určená najmä pre konfiguračné dáta
[9] Alternate Data Stream – vedľajšie dáta priradené k súboru, typicky obsahujúce metadáta

Malvér vymenováva pevné, prenosné a sieťové disky. Na pracovnej ploche a v koreňovom adresári všetkých týchto diskov vytvára odkazy, vykonávajúce šírenie. Odkazy pomenuje po vnorených priečinkoch, nastaví im ikonu priečinka a priečinky skryje. Do umiestnenia, kde sa tvoria odkazy prekopíruje svoj spustiteľný súbor ako skrytý pod názvom, ktorý vychádza z kontrolného súčtu CRC-32 vypočítaného z *Volume Serial Number*. Každý odkaz pri otvorení vykonáva príkaz v nasledujúcom formáte:

```
cmd.exe /c explorer.exe "{priečinok}" &
type "{názov}" > "%tmp%\{názov}" &
start "{názov}" "%tmp%\{názov}"
```

kde `{priečinok}` zastupuje názov vnoreného priečinka, `{názov}` nový názov spustiteľného súboru. Postupne sa v novom okne otvorí tento priečinok, spustiteľný súbor sa skopíruje do `%tmp%` a spustí sa.

Po dokončení šírenia program ukončí svoju činnosť.

Priebeh spustenia s parametrom `/u`

– Pri spustení s parametrom `/u` program už nepracuje podľa stavov. Taktiež nepoužíva *mutex* na zamedzenie súčasného behu. Predpokladá, že má administrátorské oprávnenia, keďže `/u` používa pri snahe nadobudnúť tieto oprávnenia v stave 5.

Rovnakým spôsobom ako v stave 5 sa pokúša vymazať *Volume Shadow Copies* a modifikovať kľúč `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\lnkfile` v *registri*. Pri očakávaných oprávneniach by mali obe operácie byť úspešné.

Program sa pokúša otvoriť *stavový súbor* na čítanie. V prípade, že súbor existuje, program vytvára svoj *stream* podobne ako v stave 1, no tentoraz nevyhľadáva a nezapisuje cesty k súborom na sieťových diskoch, ale iba na lokálnych. Takisto sa generujú počty súborov v jednotlivých kategóriách, no už sa ďalej nevyužívajú.

Nasleduje proces šifrovania súborov, z časti 2 *stavového súboru* preto program prečíta a importuje *lokálny* RSA-1024 verejný kľúč. Rovnako ako v stave 3 postupne prechádza cez kategórie súborov a každý súbor patriaci danej kategórii sa 5-krát pokúsi zašifrovať. Keďže týmto spôsobom sa malvér spúšťa počas stavu 5 bežného priebehu – kedy by súbory mali byť už zašifrované – tentoraz prakticky zašifruje iba súbory, u ktorých tento proces predtým zlyhal.

Po dokončení operácií alebo v prípade, že *stavový súbor* neexistoval (prípadne sa ho nepodarilo otvoriť) vykoná rovnaké kroky ako pri stave 7 – vymaže `Zone.Identifier`, šíri sa prostredníctvom odkazov na lokálnych aj sieťových diskoch a končí.

Z analýzy vyplýva, že pri spustení s parametrom `/u` Spora vykonáva niekoľko zbytočných krokov – vytvára *stream* s cestami súborov miesto ich okamžitého šifrovania a počíta štatistiky kategórií súborov, aj keď ich ďalej nevyužíva.