VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA A DETEKCE VYSOCE OBFUSKOVANÝCH MALWARE HROZEB

ANALYZING AND DETECTING HIGHLY OBFUSCATED MALWARE THREATS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DÁVID DUCHOŇ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Dr. Ing. DUŠAN KOLÁŘ**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

163358

Ústav: Ústav informačních systémů (UIFS)
Student: **Duchoň Dávid**
Program: Informační technologie
Název: **Analýza a detekce vysoce obfuskovaných malware hrozeb**
Kategorie: Bezpečnost
Akademický rok: 2024/25

Zadání:

1. Seznamte se s problematikou malware hrozeb. Následně se seznamte s metodami analýzy kódu jako jsou reverzní inženýrství či sandboxing. Dále se seznamte s metodami detekce takových hrozeb a studujte jejich výhody i nedostatky.
2. Po domluvě s konzultantem ze společnosti Gen (dříve Avast) detailně analyzujte vybrané skupiny malwaru typu cryptic (vysoce obfuskované hrozby) a zdokumentujte jejich vývoj v čase a používané techniky.
3. Zkoumejte rozšířenost těchto hrozeb, způsoby jejich šíření a nejnovější trendy. Snažte se nalézt i nové, doposud nepopsané hrozby tohoto typu či nástroje, které útočníci k tomuto účelu používají.
4. Díky získaným poznatkům navrhněte metody pro efektivní obranu proti těmto hrozbám. Zaměřte se na detekční pravidla v jazyce YARA pro alespoň deset různých rodin této hrozby. V případě nutnosti dále navrhněte potřebná rozšíření tohoto jazyka pro zmíněné účely.
5. Po konzultaci s vedoucím realizujte navržené metody z bodu 4 v praxi a otestujte je na sadě alespoň tisíce prevalentních vzorků moderního malwaru. Klaďte důraz na využití výstupů práce v praxi.
6. Svoji práci zhodnoťte a uveďte výhled na další možný vývoj.

Literatura:

- Yehoshua, Nir a Kosayev, Uriel: Antivirus Bypass Techniques: Learn practical techniques and tactics to combat, bypass, and evade antivirus software, ISBN 10: 1801079749 / ISBN 13: 9781801079747, Packt Publishing, Limited, 2021
- Sikorski, Michael a Honig, Andrew. Practical Malware Analysis: a Hands-On Guide to Dissecting Malicious Software. San Francisco: No Starch Press, 2012.
- Szor, Peter. The Art of Computer Virus Research and Defense, Addison-Wesley Professional (2005), ISBN 978-0321304544
- Eilam, Eldad, a Chikofsky, Elliot J. Reversing: secrets of reverse engineering. Indianapolis, IN: Wiley, 2005.

Při obhajobě semestrální části projektu je požadováno:
Splnění prvních tří bodů a rozpracování bodu čtvrtého.

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

Vedoucí práce: **Kolář Dušan, doc. Dr. Ing.**
Konzultant: Křoustek Jakub, Ing. Ph.D.
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 10.11.2024

Abstrakt

Táto práca zkúma aké metódy a techniky autori hrozieb používajú. Rieši sa tu ako detekovať takéto hrozby a na aké situácie si treba dať pozor. Pri takýchto hrozbách treba zohľadniť detekčné pravidlo podľa ktorého treba danú rodinu vysoko obfuskovanej malwarovej hrozby detekovať. Najefektívnejšim spôsobom je využitie reverzného inžinierstva s použitím nástrojov na statickú analýzu a dynamickú analýzu. Súčasťou práce je ukážka toho ako taká analýza môže vypadať. Výstupom tejto práce je úspešné detekovanie takýchto hrozieb využitím vytvorených pravidiel v jazyku YARA a následné otestovanie týchto pravidiel na získaných vzorkoch z reálneho sveta.

Abstract

This thesis explores the methods and techniques used by malware authors. It discusses how to detect these threats and which situations require special attention. For these threats, we need to customize detection rule to effectively detect specific highly obfuscated malware threat family. The most effective approach is the use of reverse engineering, utilizing tools for both static and dynamic analysis. A part of this thesis showcases what such analysis can look like. The outcome of this thesis is the successful detection of these threats using custom rules written in a language called YARA and the testing of these rules on real-world samples.

Kľúčové slová

YARA, cryptic, crypter, statická analýza, dynamická analýza, detekcia, malware hrozba

Keywords

YARA, cryptic, crypter, static analysis, dynamic analysis, detection, malware threat

Citácia

DUCHOŇ, Dávid. *Analýza a detekce vysoce obfuskovaných malware hrozeb*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Dr. Ing. Dušan Kolář

Analýza a detekce vysoce obfuskovaných malware hrozeb

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením doc. Dr. Ing. Dušana Koláře. Uviedol som všetky literárne pramene, publikácie a dalšie zdroje z ktorých som čerpal.

.....
Dávid Duchoň
11. mája 2025

Poďakovanie

Ďakujem svojemu vedúcemu bakalárskej práce doc. Dr. Ing. Dušanovi Koláři za pripomienky a ochotu pomáhať pri písaní práce. Ďalej by som chcel poďakovať aj Ing. Jakubovi Křoustkovi, Ph.D. a Milošovi Schrötterovi za odbornú pomoc, konzultácie a podporu.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Malware je jednou z najväčších hrozieb v kybernetickom svete. Príchodom internetu sa začala prejavovať dôležitosť malwaru kvôli motivácii autorov týchto hrozieb spôsobiť škody alebo niečo získať vypustením týchto hrozieb do dnešného sveta. Preto je potrebné nejakým spôsobom týmto hrozbám zabrániť šíreniu, následne tieto hrozby detekovať a predchádzať im. Kvôli tomuto vznikol smer, ktorý umožnil vznik počítačových programov nazvaných antivírus, aby tieto hrozby vedeli odhaliť a zabrániť ich šíreniu. Ale s vyvíjajúcou dobou sa autori týchto hrozieb začali zdokonaľovať tak, aby antivírusy tieto hrozby neboli schopné odhaliť. Túto nastupujú vysoko obfuskované hrozby, ktoré sa snažia zabezpečiť, aby sa zložitosť potrebná pre detekciu týchto hrozieb zväčšila a tým efektivita odhaľovania týchto hrozieb znížila.

Táto práca je pre kybernetickú bezpečnosť dôležitá kvôli tomu, lebo poskytuje nadhľad nad všeobecnými metódami, ako je možné pristúpiť k takej analýze týchto vysoko obfuskovaných hrozieb a následnej detekcii týchto hrozieb. Je nevyhnutné proti takýmto hrozbám bojovať, aby bol zminimalizovaný potenciálny dopad na bezpečnosť jednotlivých počítačových systémov.

V dnešnej dobe sa už proti týmto hrozbám robia určité opatrenia, ale netreba zabudnúť na fakt, že to je nikdy nekončiaci boj s týmito hrozbami.

Toto je perspektívny smer v oblasti boja proti malwarovým hrozbám z dôvodu rozšíreného použitia tohto špecifického typu hrozieb. Nachádzajú sa tu zaujímavé techniky a spôsoby obfuskácie. Pri analýze týchto hrozieb si človek, ktorý robí analýzu, dokáže rozšíriť obzory z hľadiska znalostí o hrozbách a získať nadhľad nad tým, aké idey autorov hrozieb sa aktuálne začínajú prejavovať, a tým sa dobre pripraviť na prichádzajúce hrozby.

V kapitole 2 sa rozoberá termín malware hrozba a vysvetľujú sa najznámejšie typy hrozieb, ktoré sú na svete známe.

Kapitola 3 rozoberá spôsob, ako je možné a všeobecne odporúčané pristúpiť k analýze takýchto hrozieb. Je tu vysvetlený termín reverzné inžinierstvo a popísaný sandboxing. Vytvára sa tu nadhľad nad statickou analýzou a dynamickou analýzou a poskytuje prostriedky, akými sa dá analýza realizovať.

Detekcia pri malware hrozbách je nevyhnutnou súčasťou obrany. Preto kapitola 4 vysvetľuje dva rôzne spôsoby detekcie. Jedná sa o statickú detekciu a dynamickú detekciu. V rámci týchto dvoch spôsobov je hlbšie popísaný spôsob, ako je možné dané typy detekcií realizovať úspešne.

Vysoko obfuskované hrozby sú jedným z tých typov malwaru, ktoré sú náročné detekovať, preto kapitola 5 zadefinuje, čo je to vysoko obfuskovaná hrozba, ako sa šíri, z akých základných častí sa skladá a aké obchodné modely sa používajú.

Aby bolo možné vytvoriť dobrú detekciu, je potrebné vysoko obfuskovanú hrozbu detailne zanalyzovať. Preto kapitola 6 ukazuje analýzu vybraných rodín a popisuje, ako jednotlivé rodiny fungujú, čo nám poskytne dobrý základ pre detekciu týchto hrozieb.

Na detekciu vysoko obfuskovaných hrozieb bol vybraný vyjadrovací prostriedok YARA na klasifikáciu jednotlivých rodín tohto typu malwaru. V kapitole 7 je ukázaná tvorba YARA pravidiel zanalyzovaných rodín a bližšie popísané techniky, akými je možné popísať tento typ malware hrozby.

Experimentálne výsledky sú ukázané v kapitole 8 kde je popísaný dopad vytvorených YARA pravidiel na klientov používajúcich bezpečnostné produkty firmy Gen.

V rámci kapitoly 9 je vyhodnotenie výsledkov dosiahnutých v tejto práci a sú navrhnuté vylepšenia do budúcnosti, ako zlepšiť pokrytie pri detekcii týchto hrozieb.

Kapitola 2

Malware hrozba

V tejto kapitole sa vysvetlí termín malware hrozba a rôzne známe typy malware hrozieb, ktoré sú v dnešnej dobe zadefinované.

Malware hrozba je škodlivý software. Je to v podstate kód alebo časť payloadu alebo súboru, ktorého cieľom je spôsobiť škody na napadnutých systémoch rôznymi spôsobmi. [23]

Najznámejšie spôsoby, ktorými dokáže malware spôsobiť škody, sú [23]

- Získanie plného prístupu ku koncovému systému
- Ukradnutie citlivých informácií ako sú heslá a podobné informácie
- Šifrovanie súborov a požadovanie výkupného za dešifrovanie týchto súborov
- Pokazenie používateľskej skúsenosti pri používaní špecifických programov
- Vykonávanie sledovania úživateľov
- Nútenie špecifických reklám danému užívateľovi

Na lepšie porozumenie malwaru je dobré si zadefinovať aké typy malwaru sú známe a ktoré majú dobre zadefinovaný popis funkcionality.

Keď analytik, to znamená človek, ktorý dokončí analýzu malwaru povie o danom malwaru, že sa jedná o určitý typ malwaru, myslí tým, že malware sa správa špecificky. Pre takúto klasifikáciu malwaru musíme vedieť pojmy, ktoré nazývame typy malwaru.

Vírus

Typ malwaru pri ktorom sa malware replikuje v systéme. Príkladom je to, keď sa snaží svojím chovaním napríklad spôsobiť postupné prepísanie všetkých súborov na disku nejakým obsahom.

Červ

Malware, ktorého účelom je rozšíriť sa po sieti a nakaziť ďalšie systémy pripojené na sieť, aby tento malware mohol v budúcnosti vykonať špecifické škodlivé akcie.

Rootkit

Typ malwaru, ktorý je nájdený na nižších úrovniach v operačnom systéme, ktoré majú vyššiu úroveň privilegovanosti vykonávať špecifické akcie v systéme.

Downloader

Funkciou tohto malwaru je stiahnúť škodlivý obsah. Následne tento stiahnutý obsah použiť na poškodenie používateľa.

Ransomware

Tento malware zašifruje určitý obsah systému a následne pre odšifrovanie tohto obsahu systému vyžaduje výkupné, pre to, aby užívateľ mohol pristúpiť k danému obsahu.

Botnet

Tento malware je špecifický tým, že je schopný zapríčiniť to, že používateľ, teda napadnutý systém, bude súčasťou veľkej siete napadnutých systémov. Tieto napadnuté systémy dostávajú rovnaké príkazy súčasne od servera tohto útočníka a môžu byť súčasťou budúceho potenciálneho útoku.

Backdoor

Malware, ktorý zanecháva otvorené zadné vrátka. Tieto zadné vrátka ponechávajú útočníkovi schopnosť vstúpiť do systému.

PUP(Potentionally Unwanted Program)

Toto je typ malwaru, ktorého špecifickým účelom je predstaviť nepožadovaný obsah používateľovi systému. Môže sa jednať o reklamy.

Dropper

Pri tomto malware sa určitá časť z jeho obsahu vloží do úložného priestoru napadnutého systému.

Scareware

Účelom tohto malwaru je nastrašiť užívateľov napadnutého systému, aby vykonali akcie, ktoré môžu byť pre systém škodlivé. Príkladom takejto akcie môže byť nainštalovanie falošného softvéru, ktorý nerobí to čo by mal.

Trojan

Trojan je malware, ktorý sa hrá pred užívateľom, že robí niečo legitímne. Môže to byť neškodná aplikácia v operačnom systéme ako napríklad falošný antivírus ale obsahuje škodlivú funkcionalitu.

Spyware

Účel tohto softvéru je špehovať na užívateľov a snažiť sa ukradnúť ich citlivé údaje, ktoré môžu byť následne speňažené.

Infostealer

Účelom tohto malwaru je ukradnúť citlivé informácie spojené s užívateľom na systéme. Môže sa jednať o širokú škálu informácií ako sú heslá, lokácie alebo osobné dokumenty. [18]

PWS(Password Stealer)

Jedná sa o podtyp Infostealeru. Účelom je ukradnúť heslá vyskytujúce sa v prehliadačoch alebo aplikáciach.[18]

Coinminer

Malware, ktorý využíva výpočtový výkon obete aby ťažil kryptomeny.[10]

RAT(Remote Access Trojan)

Malware je špecifický tým, že poskytuje neautorizovaný prístup k napadnutému systému na diaľku.[4]

Cryptic

Tento typ malwaru používa široké spektrum obfuskačných techník, ktoré umožňujú hrozbu spustiť bez detekcie antimalwarového softwaru. Na hrozbu, ktorá je v rámci tohto typu malwaru spúšťaná, je použitá kompresia alebo šifrovanie pre zabránenie detekcie.[1]

Malware vo svete

Motivácia vytvárať tieto škodlivé programy vychádza zo samotného postoja autora takéhoto programu nejakej entite spôsobiť určité škody alebo najvýznamnejšie v tejto dobe je získať finančný prínos pre autora. Niekedy autor nemusí mať ani odôvodnenie na spáchanie škôd. Môžu byť aj nebezpečné prípady, ktoré sa stanú pre niekoho osudnými. [3]

Malware hrozby sú po svete rozšírené. Súčasné štatistiky o potenciálnych útokoch, ktoré sa vo svete uskutočňujú, sú alarmujúce[1]. Často záleží aj od obdobia, počas ktorého sa tieto pokusy nakaziť systémy dejú. Niektoré útoky môžu súvisieť so šírením určitých politických ideológií. Dôvodov je potenciálne viac, ale dôležitým faktom je to, že je nutné týmto hrozbám dávať dostatočný odpor, aby nespôsobili škody.

[1] https://www.av-test.org/en/statistics/malware/

Kapitola 3

Metódy analýzy malwarových hrozieb

Samotná analýza malwarových hrozieb je nevyhnutnou predispozíciou na úspešnú detekciu týchto hrozieb. Preto táto kapitola vysvetľuje dva známe spôsoby analýzy kódu: reverzné inžinierstvo a sandboxing.

Keďže najviac prevalentná distribúcia malwaru je na platforme Windows, je predpokladaný fakt, že reverzné inžinierstvo a sandboxing sú vykonávané pre malware určený na túto platformu.

Reverzné inžinierstvo

Reverzné inžinierstvo je možné zadefinovať ako činnosť, kde jednotlivec alebo skupina získava znalosť o fungovaní určitého pozorovaného objektu vytvoreného niekým alebo niečím. Jedná sa o činnosť blízku vedeckému výskumu, pri ktorej sa výskumník snaží prísť na fungovanie určitého materiálu alebo konceptu, ktorý prezkúmava. [6]

V rámci tejto práce tomu môžeme rozumieť ako získavaniu poznatkov o fungovaní malwaru a o prezkúmaní jeho zákutí a záhad, ktoré môže ukrývať. Všeobecne sa jedná o pochopenie kódu, ktorý tento malware obsahuje.

Keďže malware je kód, tak je nutné tomuto kódu porozumieť. Malware kód môže byť napísaný v dvoch formách. Jedna forma predstavuje kód zo skriptovacieho jazyka, to znamená kód, ktorý sa dá ľahko čítať. Príkladom takéhoto kódu môžu byť Powershell skripty, VisualBasic skripty alebo trendom je tiež prítomnosť Python skriptov. Ďalším kódom je binárny kód. V tomto prípade je binárny kód reprezentovaný strojovým kódom, čiže kódom, ktorému procesor daného systému rozumie.

V tejto práci si rozdelíme spôsoby reverzného inžinierstva na statickú analýzu a dynamickú analýzu.

Statická analýza

Statická analýza predstavuje analýzu kódu bez spúšťania daného programu, to znamená, že sa snažíme nájsť hlavne na základe statických vlastností informácie o fungovaní malwaru. [6] Jedná sa o jeden z prvých krokov analytika, keď sa chystá detailne analyzovať daný vzorok malwaru. Je podstatné vybrať vhodný prostriedok na analýzu kódu a to podľa kritéria, ktoré určuje, aký kód chce analytik analyzovať. V rámci tejto práce je podstatný

strojový kód. Preto sú brané do úvahy hlavne nástroje, ktoré slúžia na statickú analýzu binárneho kódu. Keďže sa zameriavame na binárny kód, je nutné vysvetliť spustiteľný kód.

PE formát

Keďže najväčší počet hrozieb sa nachádza na platforme Windows, je pre nás táto platforma najzaujímavejšia. Pre binárny kód všeobecne platí, že keď chceme, aby sa spustil, musí operačný systém vedieť, ako to spustiť. Na binárny kód na platforme Windows sa používa PE(Portable Executable) formát. Tento formát je pri analýze potrebné poznať. Hlavnou podstatou je, že program, ktorý vytvára proces a teda spúšťa daný binárny kód, musí vedieť, kde má vôbec byť prvá inštrukcia. Adresu tejto inštrukcie definuje PE hlavička. Tejto adrese sa hovorí vstupný bod. [13]

.NET

V rámci platformy Windows je jeden z najrozšírenejších ekosystémov práve .NET. .NET predstavuje sadu knižníc pre vývojárov, ktorí môžu relatívne ľahko vyvíjať programy práve v tejto technológii. Táto technológia je význačná tým, že všetky programy v nej vytvorené sú v PE formáte uložené využitím špecifického formátu a sú v rámci binárnych dát jednotlivé kódy vyjadrené v systéme zvanom CLI (Common Language Infrastracture)[1]. Znalosť tohto formátu je pri analýze malwarových hrozieb pre platformu Windows tiež potrebná. [12]

Disassembler

Disassembler je program, ktorý je schopný konvertovať strojový kód do jazyka symbolických inštrukcií. [6] Keďže existuje aj .NET tak v tomto prípade strojový kód predstavuje binárnu podobu inštrukcií medzikódu, ktorý následne prekonvertuje na symbolickú podobu do medzikódu. Jeden z najznámejších nástrojov, ktoré budeme pre analýzu vzorkov používať, je nástroj IDA[2] v kombinácii s nástrojmi ako je Ghidra[3]. Pre .NET hrozby budeme používať ILSpy[4].

Tieto nástroje majú výhodu v tom, že sú dlhodobo vyvíjané a tým už sú schopné vďaka ich dlhodobému vývoju poskytnúť dostatočnú úroveň potrebnú pre analytika na vykonanie detailnej statickej analýzy.

Analýza v takýchto nástrojoch je schopná rýchlo napredovať a robiť pokroky kvôli možnej vizualizácii jazyka symbolických inštrukcií a naviazaniu jednotlivých súvisiacich celkov takéhoto kódu nazvaných blokmi. Keďže samotný jazyk symbolických inštrukcií pri komplexných programoch môže spraviť tento druh analýzy časovo náročným procesom, tak vznikol nástroj, ktorý sa volá Dekompilátor.

Dekompilátor

Dekompilátor je nástroj, ktorý je schopný zobrať na vstupe strojový kód špecifikovaného programu a tento strojový kód premeniť na kód v kompilovanom jazyku. [6] Štandardným kódom je jazyk C. Výhodou tohto nástroja je schopnosť zminimalizovať čas potrebný na porozumenie funkcionality kódu analyzovaného malwaru.

[1] https://ecma-international.org/publications-and-standards/standards/ecma-335/
[2] https://hex-rays.com/ida-pro
[3] https://ghidra-sre.org/
[4] https://github.com/icsharpcode/ILSpy

Prepracované disassemblery ako IDA[5] a Ghidra[6] majú tento nástroj dostupný. Jednou nevýhodou je fakt, že dekompilátor môže vynechať nejaké inštrukcie, ktoré sa tomuto nástroju prejavujú tak, že by na kód a teda jeho význam nemali vplyv. Môže sa to prejaviť tak, že následná analýza takého kódu môže obsahovať časti, ktoré nie sú dokonalou ekvivalentnou reprezentáciou strojového kódu. [6] V takýchto prípadoch je potrebné, aby analytik manuálne pomohol alebo naviedol tento nástroj na správne vygenerovanie kódu.

Statická analýza má jednu veľkú nevýhodu a to zložitosť a priamu úmeru od veľkosti a rozsahu daného kódu. Jedna zvyčajná nevýhoda sa začína prejavovať v momente, keď kód je obfuskovaný, čo sa nám prejaví v rámci analýzy vysoko obfuskovaných hrozieb. Pre takúto analýzu je rozumné daný kód poznať. Výhoda takéhoto prístupu je skutočnosť, že tým, že kód nie je spustený, nemôže sa spustiť nelegitímne správanie. Vo všeobecnosti to znamená, že program nemôže bežať, čo následne implikuje fakt, že analytik používajúci túto metódu nemôže spôsobiť infikovanie jeho stroja, na ktorom vykonáva statickú analýzu. Najväčšia výhoda tohto prístupu je, že poskytuje komplexné porozumenie toho, ako je program štrukturovaný, o aké potenciálne miesta sa môže analytik uchytiť pri pokuse detekovať danú hrozbu.

Dynamická analýza

Úlohou dynamickej analýzy je zistiť správanie programu resp. analyzovanej hrozby. Celá činnosť sa deje cez spúšťanie danej hrozby v kontrolovanom prostredí. [16][18] Analytik pri použití tohto spôsobu môže použiť rôzne druhy tohto prostredia. Najvýznamnejšími nástrojmi používanými analytikmi sú virtuálny stroj a sandbox.

Virtuálny stroj je plne virtualizované prostredie, schopné virtualizovať celý hardvérový systém. Súčasťou takéhoto systému bývajú CPU, operačná pamäť, úložisko a sieťové rozhrania. Účelom virtuálneho stroja je poskytnúť virtualizované prostredie, ktoré sa správa ako fyzický počítač. Dané prostredie analytik používa, keď chce spraviť detailnú analýzu programu.

V rámci virtuálneho stroja analytik na získanie informácií o hrozbe používa rôzne nástroje schopné poskytnúť informácie o behu programu. Keď sa jedná o hrozbu určenú na platformu Windows, analytik má možnosť použiť sadu prostriedkov ako sú Process Explorer[7] alebo Process Monitor[8].

Ďalším nástrojom, ktorý je nevyhnutný pre analýzu strojového kódu, je Debugger.

Debugger

Debugger je nástroj používaný developermi, analytikmi na testovanie, analýzu a opravu programov. Umožňuje používateľom monitorovať, kontrolovať a manipulovať chod programu. [16] Analytici najčastejšie používajú v tejto dobe x64Dbg[9], ktorý bude súčasťou vykonávania analýzy hrozieb. V rámci tejto práce tento program bude používaný často vďaka jeho schopnosti kontrolovať chod programu. Pre .NET sa jedná o nástroj dnSpy[10].

Tento spôsob je používaný na odhalenie chovania hrozby, čo nie je často možné pri použití statickej analýzy pri modernom malwari ľahko realizovať.

[5] https://hex-rays.com/ida-pro
[6] https://ghidra-sre.org/
[7] https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer
[8] https://learn.microsoft.com/en-us/sysinternals/downloads/procmon
[9] https://x64dbg.com/
[10] https://github.com/dnSpyEx/dnSpy

Sandboxing

Sandboxing je bezpečnostná technika, ktorá je súčasťou dynamickej analýzy použitá na izoláciu a spúšťanie programov v kontrolovanom prostredí. [23] Sandbox je oddelené prostredie s obmedzeným prístupom k systémovým zdrojom. Následkom tohto faktu je, že akýkoľvek škodlivý kód neovplyvní stroj, na ktorom daný Sandbox pôsobí.

Pri analýze vzorkov pomocou techniky sandboxingu sú sandboxy prispôsobené ich účelu. Keď sa hrozba v tomto sandboxe spustí, väčšinou sa vykonajú špecializované monitorovacie procedúry, pozberajú sa relevantné štatistiky, ktoré sú schopné odhaliť činnosti, ktoré môžu pomôcť k odhaleniu alebo detekovaniu potenciálnej hrozby, o ktorej nie je známe, či je to hrozba alebo nie.

Keďže sa zameriavame na hrozby určené na platformu Windows, je vhodným spôsobom použiť Sandbox schopný monitorovať a zaznamenávať vykonané operácie nad týmto systémom. Konkrétne to môže byť využitie knižníc a ich procedúr vo Windows API.

Jedna kľúčová výhoda pri využití sandboxingu je získanie nadhľadu nad celkovým fungovaním špecifickej testovanej hrozby. Výstup analýzy zo sandboxu je následne možné použiť na detailnejšie prejdenie a až následne na základe pozorovaného chovania, ktoré sa prejavilo pri spustení danej hrozby v danom sandboxe, je možné pokračovať rozhodnutím, či sa jedná o hrozbu. Antivírusy sú schopné tento výsledok využiť na vyhodnotenie daného kódu ako hrozby a tým je aplikovaná detekcia tejto hrozby. Veľkou nevýhodou tejto techniky je fakt, že hrozby môžu byť tak upravené ich autormi, že sú schopné výsledok tejto analýzy ovplyvniť, a to detekovaním, že daná spustená hrozba je spúšťaná v konkrétnom sandboxe, na ktorý je hrozba pripravená a pokúsi sa schovať škodlivé správanie.

Jedným populárnym Sandboxom je tzv. Cuckoo Sandbox.[11] Poskytuje nadhľad nad systémom Windows samotným. Jednou výhodou tohto sandboxu je schopnosť automatizovane spustiť a ukončiť spúšťaný program a tým aj v prípade napadnutia systému tento systém obnoviť. To znamená, že sandboxing výhoda je zjavná pri automatizovanom testovaní hrozieb a následnom zhodnotení, či výsledok analýzy danej hrozby preukazuje prítomnosť škodlivého správania typického pre malware.

[11] https://cuckoo.readthedocs.io/en/latest

Kapitola 4

Metódy detekcie malware hrozieb

V tejto kapitole si preberieme možné metódy detekcie malware hrozieb, ktoré sa používajú dnes a vyhodnotíme si ich výhody a nevýhody, ktoré sa pri aplikovaní týchto detekcií môžu vyskytnúť.

Metódy, ktoré sú používané hlavne pri antivírusoch, sú Statická detekcia, Dynamická detekcia a Heuristická detekcia. Tieto detekcie sú založené na tom, aký engine daný antivírus používa, čiže môže sa jednať o Statický engine, Dynamický engine a Heuristický engine. [23] Aby sa uskutočnila daná detekcia, je nutné mať vytvorené detekčné pravidlo alebo algoritmus, ktoré je schopné rozhodnúť o existencii škodlivého správania pri vzorku.

4.1 Statická detekcia

Statická detekcia je založená na tom, že sa hľadá nejaká statická signatúra, ktorá je pre danú hrozbu špecifická. Veľká časť tejto detekcie zahŕňa hlavne špecifické reťazce alebo určité bytové vzory pri určitých miestach.

Celý proces statickej detekcie začína tak, že sa najprv určí typ analyzovaného súboru. Následne treba získať informácie o požadovanej štruktúre, ktorú by mohla daná hrozba obsahovať. Jedným ďalším dôležitým údajom, ktorý sa dá na detekciu použiť, sú metadata alebo inak povedané dáta o dátach.

Metadata

Keďže sa zameriavame hlavne na PE spustiteľné súbory, tak metadata, ktoré sú pre tieto súbory typické, môžu byť mená príslušných sekcií. [13]

Každá sekcia obsahuje jej meno, čo môže nabádať na fakt, že tieto mená sú autormi týchto hrozieb manuálne vytvorené. Pri takomto spôsobe to je relatívne ľahké odhaliť a detekovať. Keďže stačí mať prístup k funkčnému parseru schopného tento formát rozpoznať.

Ďalšia vlastnosť, ktorá sa dá využiť, je využitie pri PE súboroch prítomnosť importovacích a exportovacích tabuliek v PE. [13] Tieto tabuľky dávajú dôležitú informáciu loaderu spúšťajúcemu daný program. Konkrétne sú to informácie týkajúce sa požadovaných knižníc, ktoré sa používajú v rámci programu. Táto informácia sa rozširuje na využité procedúry, ktoré program v nejakom bode zavolá. Niektoré procedúry môžu byť tak špecifické, že v bežných spustiteľných súboroch sa používajú zriedkavo. Čo môže viesť k efektívnemu záchytnému bodu, ktorý môže statická signatúra alebo algoritmus využiť.

Využitie značiek je jeden z možných prostriedkov pre dostatočnú statickú detekciu. Značky môžu byť časové, také značky, ktoré hovoria o dátume vytvorenia daného súboru.

Rich Header[1] je v PE formáte tiež uznávaný na značkovanie programov. Tieto značky sú špecifické hlavne pre kompilátory, ktoré vytvorili daný PE súbor. Takýto spôsob dokáže aj pomôcť pri rozlíšení legitímneho softvéru, v rámci tohto prípadu to netreba brať ako smerodajný údaj, keďže štruktúra tohto formátu je dobre známa aktérmi roznášajúcimi malware. [13]

Jedným z posledných typických príkladov takéhoto prístupu je využitie informačnej entropie. Každá sekcia môže mať určitý rozsah tejto hodnoty, čo môže efektívne rozlíšiť potenciálnu rodinu, ktorú je treba odlíšiť od ostatných hrozieb alebo normálnych neškodných programov. Pri tomto prípade sa jedná o relatívne zložitú činnosť, kde je potrebné určiť najlepšie taký rozsah, aby sa neprekrýval s ostatnými programami, ktoré sú neškodné. Najčastejšie sa používa jedna hraničná hodnota, ktorá môže určiť hornú mez alebo dolnú mez. [18]

Certifikát, ktorý overuje identitu programu v PE formáte, je možné pri jeho pridaní do daného PE súboru. Malware často používa certifikát, ktorý je ukradnutý, v tomto prípade nie je možné brať tento certifikát ako záruku, že program je autentický. Expirovaný certifikát je v malware tiež používaný, v tomto prípade je možné o programe samotnom rozhodnúť, že nie je bezpečné ho spustiť. Posledný typ certifikátu, ktorý je malwareom zneužívaný, je samopodpísaný. V takomto prípade je možné rozhodnúť, že je nebezpečne spúšťať taký program. V rámci detekcie môže byť špecifická hodnota certifikátu tiež použitá ako jeden z ďalších oporných bodov. [13]

Reťazce

Tento spôsob je jeden z tých efektívnejších, obzvlášť, keď sa jedná o malware, ktorý sa snaží obeti, ktorú tento malware postihol, zdieľať nejakú určitú informáciu. Často to môžu byť reťazce, ktoré súvisia s reťazcami, ktoré oznamujú užívateľovi, že je jeho systém napadnutý alebo je nutné zaplatiť určité výkupné. Takéto reťazce sú významné hlavne pri malware typu Ransomware. Ďalší typ malwaru, ktorý môže mať špecifické reťazce uložené, je RAT. Tento typ má význačne uložené reťazce špecifických adries URL alebo môže sa taktiež jednať o IP adresy. Reťazce, ktoré popisujú samotnú štruktúru PE formátu, sú tiež v určitých prípadoch významné, ako bolo spomenuté vyššie pre metadátach. [18]

Bajtové vzory

Bajtové vzory reprezentujú špecifické sekvencie bajtov, ktoré pri určitej hrozbe sú významné hlavne pri funkcionalite danej hrozby. Tieto vzory môžu byť napríklad binárne kódy, ktoré reprezentujú časť určitej procedúry podstatnej pri plnení funkcionality malwaru.

Pri bajtových vzoroch sa môže jednať aj o formu takzvaného paddingu pri dátach, ktoré môžu byť určitým spôsobom označené. Význam takýchto dát sa vyskytne vtedy, keď sa jedná o konfigúraciu malwaru samotného [7]. V rámci tohto prípadu môže mať ten malware vytvorený vlastný parser, ktorý očakáva špecifickú štruktúru dát, ktorej obsah je variabilný iba v špecifických oblastiach.

Z tohto nám vyplýva, že treba detekčné pravidlo napísať tak, aby bolo možné danú malware hrozbu detekovať za predpokladu, že môže nadobudnúť viacero rôznych foriem. Konkrétne to predstavujú reťazce alebo bajtové vzory špecifické pre danú hrozbu.

[1] https://www.giac.org/paper/grem/6321/leveraging-pe-rich-header-static-alware-etection-linking/169729

Nevýhodou tejto detekcie je náchylnosť na skutočnosť, keď sa hrozba jej autormi modifikuje tak, že napísané detekčné pravidlo odchytávacie danú hrozbu je nutné upraviť. Preto nám z toho plynie, že statické signatúry sú náchylné na čas. Preto treba venovať veľkú pozornosť aktualizovaniu takýchto signatúr.

Výhodou je možnosť jednoducho detekovať určitú množinu hrozieb relatívne za krátku dobu. To znamená, že je to najrýchlejší spôsob detekcie. Ďalšia výhoda je, keď existuje daná signatúra v databáze, tak je možné hrozbu zastaviť ešte skôr, ako sa spustí.

4.2 Dynamická detekcia

Ďalšiou metódou, ktorá sa môže použiť na detekciu malwarových hrozieb, je dynamická detekcia. Princípom tejto detekcie je detailnú analýzu správania hrozby na systémovej úrovni použiť na detekčné účely.

Systémové služby

Pri hrozbách je vhodné sa zamerať na využívanie služieb poskytovaných operačným systémom, hlavným takýmto prostriedkom sú funkcie a procedúry, ktoré sú poskytované v rámci API operačných systémov. [8] V kontexte Windowsu to je známy Windows API[2]. Túto činnosť treba vykonávať tak, aby sme sa zameriavali na určitú sekvenciu využitých procedúr a funkcií. Samostatné jedno volanie funkcie alebo procedúry nemôže byť brané ako dostatočne kvalitné, z dôvodu, že rôzne programy môžu mať rovnaké správanie.

Práca so súbormi

Významným indikátorom taktiež býva vytváranie, modifikovanie alebo vymazávanie súborov v súborovom systéme. Pre toto sú typické zmeny v súborovom systéme s vopred definovanými menami pre dané súbory.

Windows Register

Windows je špecifický tým, že má globálne úložisko určené pre ukladanie stavových a konfiguračných informácií známe ako Register[3]. Malware je známy využitím tohto úložiska. V rámci detekcie je vhodné zaznamenávať, ku ktorým vetvám v rámci registru malware pristupuje a ktoré vetvy modifikuje. [19]

Sieťová komunikácia

Hrozby sa taktiež snažia používať sieťovú komunikáciu, takže akýkoľvek náznak špecifickej komunikácie v sieti, ktorý sa prejavuje pri tejto komunikácii, je dobrým smerom, ako by bolo možné dané detekčné pravidlo vytvoriť. [5][18]

Na dynamickú detekciu je potrebné použiť prostriedok schopný dodať informácie súvisiace s behom danej hrozby. Používaným prostriedkom je virtualizované prostredie. Jeho výstup analýzy by mal poskytnúť dostatočný podklad na uskutočnenie detekcie. Známym takým prostredím je Cuckoo Sandbox, ktoré uchováva záznamy správania malwaru a aktivít, ktoré so systémom robí.

[2] https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list
[3] https://learn.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry

Výhodou tejto detekcie je fakt, že môže odhaliť hrozby aj v prípadoch, keď je ich kód zašifrovaný alebo vysoko obfuskovaný za predpokladu známeho správania danej hrozby.

Nevýhodou môže byť fakt, že niektoré hrozby menia prvky, ktoré sú hlavným indikátorom použitým pri dynamickej detekcii. To znamená napríklad, že sa mení názov známych súborov, ktoré sa v súborovom systéme zmenia alebo sa môže kompletne zmeniť špecifická sieťová komunikácia, ktorá bola v predošlom detekčnom pravidle zvýraznená ako charakteristický prvok fungovania danej hrozby. Pri práci s registrom to predstavuje zmenu názvov pristupovaných vetví.

Kapitola 5

Vysoko obfuskované hrozby

V tejto kapitole je vysvetlené, čo je to vysoko obfuskovaná hrozba, aké techniky sa používajú pri takejto hrozbe a ako je Cryptic vo svete populárny. Je vykonaná analýza vybranej skupiny malwaru typu Cryptic, kde sú ukázané výsledky analýzy a vysvetlené, ako dané analyzované hrozby fungujú a ako sú tieto analyzované vzorky prevalentné.

5.1 Cryptic

Vysoko obfuskovaná hrozba alebo Cryptic je určitý typ malwaru, ktorý bol vytvorený preto, aby vďaka určitým špecifickým vlastnostiam dokázal uniknúť detekcii či už antivírusom alebo analytikom. Používa obfuskačné techniky, ktoré sa používajú na to, aby spravili náročnú analýzu kódu alebo aby sa vyhli antivírusom. Synonymom pre Cryptic je pomenovanie Crypter. [2]

Princíp Crypticu

Crypticy sa väčšinou skladajú zo základných častí. Tieto časti sú dešifrovacia časť, načítavacia časť a následne skok do originálneho vstupného bodu. [23][18] Keďže sa v tejto práci zameriavame na Crypticy, ktoré sú obsiahnuté v PE súboroch, budeme popisovať jednotlivé časti Crypticu s týmto predpokladom.

Dešifrovacia časť je zodpovedná za dešifrovanie hlavného programu, ktorý bol zašifrovaný v rámci tohto daného Crypticu. Používajú sa tu rôzne šifrovacie algoritmy. V tejto časti je najviac významná obfuskačná časť. Používajú sa tu techniky, ktoré sa snažia zbrzdiť analytika a taktiež techniky, ktoré sa snažia zabrániť dynamickej analýze využitím virtualizovaného prostredia. Súčastou tejto časti je takzvaná dešifrovacia rutina, ktorá je zodpovedná za dešifrovanie hlavného programu.

V načítavacej časti je najvýznamnejším prvkom načítavanie dôležitých závislosti hlavného programu. V prípade binárneho kódu sa môže jednať o načítavanie potrebných knižníc a používaných procedúr implementovaných v týchto knižniciach. Úlohou tejto časti je zabezpečiť, aby podklady pre daný program boli pripravené. Časť načítavania môže tiež obsahovať určité techniky na zbrzdenie analýzy analytikom alebo detekciu virtualizovaného prostredia, poprípade detekciu antivírusového programu.

Skok do originálneho vstupného bodu je finálna časť pre pôsobenie Crypticu. V tejto časti sa originálny program spustí. Spôsobov, ako to docieliť, je viacero. Táto časť sa odvíja hlavne od toho, akým spôsobom je hlavný program dešifrovaný. Najčastejšie sa používa vytvore-

nie nového vlákna, procesu alebo vytvorenie miesta pre hlavný program v rámci adresného priestoru procesu, v ktorom daný Cryptic beží v operačnom systéme.

Techniky

Techniky, ktoré tento typ malwaru používa, sú rôzne.

Najčastejšie techniky sú techniky súvisiace s komplikovaním statickej analýzy. Keď disassembler alebo dekompilátor sa tento kód snaží prejsť, tak niektoré heuristiky, ktoré určujú, či sa k danému kódu dá dostať určitým spôsobom, následne neuspejú. Výstupom môžu byť niektoré časti kódu označené tak, že dekompilátor a disassembler nepredpokladajú, že sa spustí časť tohto kódu.

Ďalšími potenciálnymi technikami sú techniky, ktoré sa snažia spraviť dynamickú analýzu komplexnou. Často sa to deje tým, že sa snaží zistiť prítomnosť debuggeru. Debugger sa na platforme Windows dá detekovať viacerými spôsobmi. Jedná sa hlavne o využitie Windows API procedúr, ktoré sú schopné zistiť a vrátiť tieto stavové informácie o debuggeroch. [16]

Ďalšou technikou pri dynamickej analýze je pridanie zbytočných inštrukcií. Deje sa to tak, že analytik je potom nútený odfiltrovať tieto zbytočné inštrukcie, čo je v kontexte komplexného kódu zložité. [16][23]

5.2 Cryptic vo svete

Cryptic je vo svete relatívne populárny. [15] Dôvodom je motivácia autorov hrozieb svoj kód určitým spôsobom ochrániť. Preto je samotný Cryptic dôležitý pre týchto autorov.

Na svete existujú rôzne nástroje na vytvorenie týchto Crypticov. Autori tieto nástroje môžu použiť, ale niektorí si dokonca vytvárajú vlastné Crypticy, aby vytvorili pre svoju hrozbu plnohodnotnú obranu proti detekcii. Nevýhodou použitia pri týchto Crypticoch je to, že keď sa detekuje jedna rodina Crypticu, následne to autorov núti vytvoriť Cryptic nový. Preto ako dôsledok tejto skutočnosti je fakt, že vzniká relatívne veľké množstvo rodín tohto typu malwaru.

V súčasnosti tieto Crypticy sú používané v podstate na širokú škálu typov hrozieb. Hypoteticky, je možné Cryptic použiť na ľubovoľný typ malwareu, kvôli všeobecnej funkcionalite. To znamená, že proces dešifrovania, načítavania a skoku do originálneho vstupného bodu je všeobecný a neexistuje len jeden spôsob, ako to spraviť.

Obchodné modely

Hrozba je hlavne rozšírená v underground marketoch, kde sa ponúka často formou Crypter-as-a-Service. [11] To znamená, že tam je model poskytovania Crypticu cez možné pravidelné platby pre stále aktualizácie a podporu.

Ďalší spôsob, akým sa tieto Cryptery šíria, je použitie hackovacích fór. Na fórach môžu byť zdieľané za platby, alebo niektoré môžu byť propagované zdarma. Známym miestom, kde je tiež možné získať Cryptic, je Dark Web.

Vlastný alebo nevlastný Cryptic

Autori Crypticov môžu za určitú čiastku ponúknuť takzvaný nevlastný Cryptic. To vo všeobecnosti znamená, že tento Cryptic je určený pre širokú škálu programov. Tieto Crypticy majú pre autorov určitých malware hrozieb tú nevýhodu, že tým, ako je Cryptic nevlastný,

neposkytuje ich programom obranu učenú priamo pre ich malware. V praxi to znamená fakt, že daný Cryptic pre šifrovaný malware nie je najlepšie prispôsobený a tým nemusí poskytnúť dostatočnú ochranu pre daný malware.

Druhou stranou Crypticov sú vlastné Crypticy. Pri týchto Crypticoch autori pri používaní Crypter-as-a-Service obchodného modelu sú schopní vďaka vyšším platobným čiastkám prispôsobiť Cryptic podľa typu programu, ktorý im ich klienti, čiže autori ostatných hrozieb, ponúknu.

Výskyt Crypticov

Crypticy je možné nájsť tiež pri phishingových kampaniach. Často sú posielané formou e-mailových príloh, alebo škodlivých linkov. [20]

Hľadanie Crypticov

Nájdenie nepopísaného Crypticu je relatívne náročný proces. Tým, že sa v Crypticoch najviac používa šifrovanie alebo zabalenie, by bolo vhodné hľadať vzorky, ktoré obsahujú vysokú entropiu v sekciách PE. Nie je pevne stanovená hodnota tejto entropie pri jednotlivých sekciách v rámci PE súboru. Väčšina Crypticov obsahuje tieto hodnoty pri zašifrovaných sekciách v rozmedzí 6 až 8. Jednou vecou, na čo si treba dať pozor, je na veľkosť týchto sekcií. Môže sa jednať o typ malwaru, ktorý len malú časť sekcie používa na šifrované dáta, preto treba dôkladne správanie týchto hrozieb overiť, aby sa overilo, či to je naozaj Cryptic.

Kapitola 6

Analýza Crypticov

V tejto kapitole sa ukáže podmnožina zo všetkých zanalyzovaných rodín Crypticov. Zdokumentuje sa tu aké techniky sú pri týchto rodinách používané a aké sú tieto rodiny Crypticov prevalentné v čase.

6.1 Cryptic Havefun

Táto rodina je relatívne počtom vzorkom prevalentná. V rámci troch mesiacov bolo nájdených na počítačoch napadnutých užívateľov 81. Celkovo vzorkov obsahujúcich túto rodinu majú zhruba 1500 nájdených unikátnych vzorkov s unikátnym hashom celého vzorku.

Typ malwaru, ktorý sa šíri je typ malwaru PWS. Čiže autori malwaru, ktorý sa snaží kradnúť užívateľské údaje.

Na obrázku 6.1 je vidieť vstupný bod tohto Crypticu. Prebieha tu volanie funkcie `CreateThread`, ktorá vytvorí vlákno pre spustený proces, ktorým je Cryptic. Vlákno má nastavený vstupný bod `StartAddress`, ktorý sa začne spúšťať. Následne prebieha čakanie pomocou procedúry `WaitForSingleObject`, ktorá umožní čakať na ukončenie vytvoreného vlákna.

Vstupný bod `StartAddress`, spôsobí načítanie dynamickej knižnice `shell32.dll`, čo je knižnica používaná na funkčnosť shellu.

V rámci danej procedúry sú dve volania funkcie `fwrite`, ktoré zapričínia zápis binárnych dát do súborov. V tomto bode 6.2 prebieha zápis binárnych dát do špecifického miesta pre tieto dešifrované dáta.

```
void main()
{
    handleThread = CreateThread(StartAddress);
    WaitForSingleObject(handleThread);
}
```

Obr. 6.1: Havefun – Pseudokód vstupného bodu

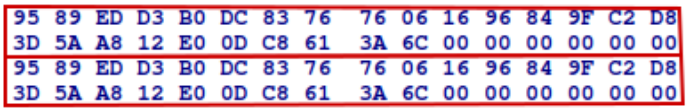
```
fileName = &createdFileName;
if (randomVariable >= 0x10) {
    // This executes everytime
    fileName = createdFileName;
}
fileDescriptor = fopen(fileName,"wb");
fwrite(decipheredBlock, sizeOfBlock, fileDescriptor);
fclose(fileDescriptor);
```

Obr. 6.2: Havefun – Pseudokód zápisu do súboru

Tieto dva dešifrované súbory vzniknú postupným dešifrovaním obsahu dát v PE súbore pomocou dešifrovacieho algoritmu 6.3, ktorého kľúče sa nachádzajú na špecifickom mieste v adresnom priestore procesu. Algoritmus je najviac podobný RC4 implementácii. [9] Veľkost týchto kľúčov je 32 bajtov 6.4.

```
indexHelper = 0;
for (int i = 0; i < lengthBlock; i++) {
    indexToKey = (indexToKey + 1) % 256;
    keyByte = key[indexToKey];
    indexToKey2 = (keyByte + indexHelper) % 256;
    key[indexToKey] = key[indexToKey2];
    key[indexToKey2] = keyByte;
    indexToKey3 = (keyByte + key[indexToKey]) % 256;
    indexHelper = indexToKey2;
    decryptedBlock[i] = decryptedBlock[i] ^ key[newValue];
}
```

Obr. 6.3: Havefun – Pseudokód dešifrovacieho algoritmu



Obr. 6.4: Havefun – Kľúče

Z tohto obrázku je viditeľný fakt, že RC4 kľúče na odšifrovanie payloadov sú rovnaké, toto nie je samozrejmosť, keďže samotné načítanie kľučov nie je vôbec podmienené tým či sú rovnaké. [9]

Dešifrované spustiteľné súbory, ktoré sa vytvoria, sa následne spúšťajú cez procedúru `ShellExecuteA`, ktorá sa zavolá obfuskovaným spôsobom tak, že sa využije ukazovateľ na ňu, ako je znázornené na 6.5. Podstatou tejto funkcionality je, že spustiteľné súbory sú spúšťané zo súborového systému.

Na 6.5 je ukázaná forma obfuskácie vkladania zbytočného kódu, čo je bežná forma použitá a rozmiestnená po celom kóde daného Crypticu. Princíp tejto obfuskácie je v tom, že sa kontrolujú podmienky, ktoré budú vždy pravdivé, čiže nastáva tu pokus pri statickej a dynamickej analýze analytikom o potenciálne zdržanie skúšaním a pozeraním ďalších vetví.

```
copyFileName{N} = &fileName{N}
if (random_variable >= 0x10) {
    // This will execute everytime
    copyFileName{N} = fileName{N};
}
pointerToShellExecuteAProcedure("open",copyFileName1);
```

Obr. 6.5: Havefun – Pseudokód volania dešifrovaného obsahu

6.2 Cryptic Dyg

Táto rodina je zaujímavá tým, že za posledné 4 mesiace roka 2024 bolo chytených 435 vzorkov na počítačoch užívateľov antivírusu Avast. Prevalentným typom malwaru je PWS.

Vstupný bod tejto varianty je pomerne jednoduchý. Obsahuje volanie jednej procedúry, v ktorej celý proces začína.

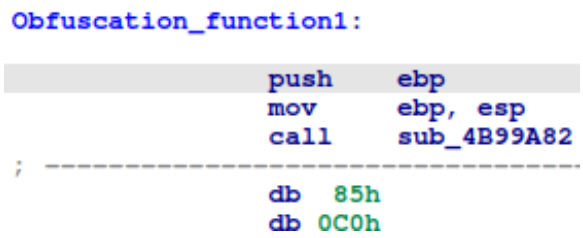
Táto varianta sa snaží detekovať či je spustená cez súbor, ktorý obsahuje známe hashe ako SHA256 alebo MD5. Ukážka použitej detekcie je na 6.6. Pokusom tejto techniky je zastaviť beh analyzovanej hrozby v sandboxe alebo pri analytikovi, kvôli skutočnosti, že analytici často pomenúvajú vzorky podľa ich získaných hešov od použitia hešovacej funkcie, ktorá získa túto hodnotu na základe celkového obsahu vzorku.

```
bool detectHash() {
    GetModuleFileNameW(&fileName,maxFileNameLength);
    lengthFileName = lstrlenW(fileName);

    if (!contains_hex_characters(fileName) ||
        lengthFileName != 16 && lengthFileName != 32
        && lengthFileName != 40 && lengthFileName != 64) {
            return false;
    }
    return true;
}
```

Obr. 6.6: Dyg – Pseudokód detekovania hashu

Tento Cryptic sa snaží sťažiť analýzu malwaru tým, že schová pre analytika v rámci statickej analýzy inštrukcie, ktoré statický analyzátor do úvahy neberie, lebo v kontexte vykonanej automatickej počiatočnej analýzy neprejde všetky možné cesty, ktorými môže malware prejsť. Na obrázku 6.7 je znázornená obfuskácia. Bajty znázornené pseudoinštrukciou `db` nie sú statickým analyzátorom zanalyzované. V skutočnosti počas dynamickej analýzy sa dá zistiť, že sa tam v určitom bode procesor dostane.

Do alokovaného bloku, ktorý je alokovaný `GlobalAlloc` procedúrou je prenášaná časť šifrovaných dát cez procedúru 6.8. Táto procedúra prenáša dáta s variabilným ukazovateľom na aktuálny bajt. Procedúra je v rámci behu Crypticu spustená nad viacerými blokmi s odlišnými konštantami ovplyvňujúce aktuálny ukazovateľ na prenášaný bajt.



Obr. 6.7: Dyg – Obfuskácia pre statický analyzátor

```
index = 0;
if (lengthOriginalBlock > 0) {
    moveConstant = 9871;
    moveConstant2 = 413355184;
    do {
        newBlock[moveConstant - 9871] = originalBlock[index];
        moveConstant2 = moveConstant2 + 912095;
        index += (moveConstant2 >> (moveConstant1 & 3)) % 5 + 1;
        ++moveConstant;
    } while (index < length);
}
```

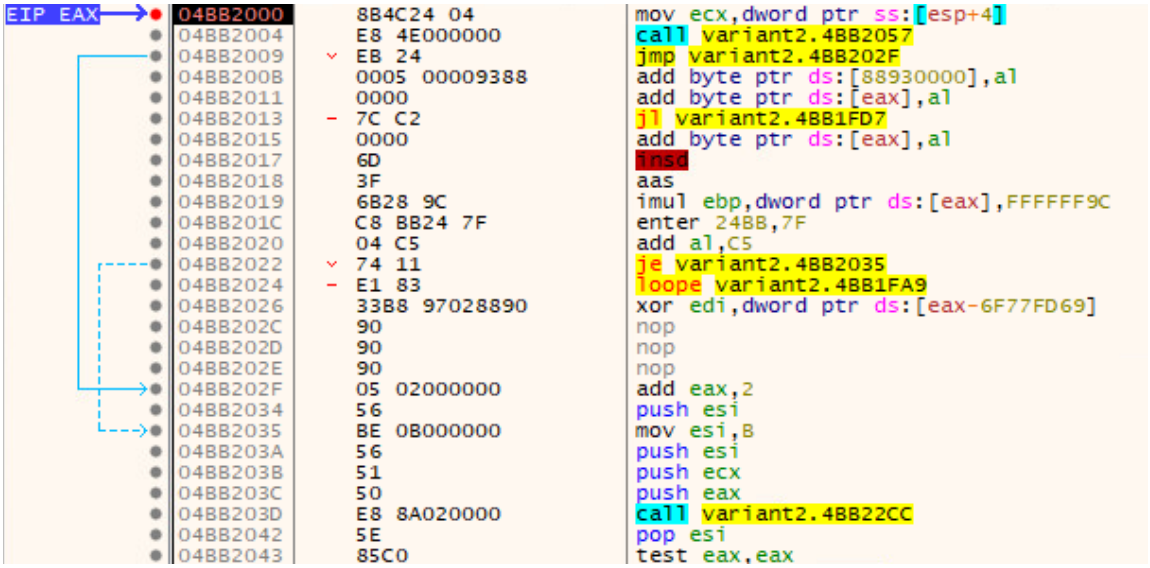
Obr. 6.8: Dyg – Pseudokód pre prenosovú funkciu

Najprv je použitá tvorba kľúča o veľkosti 16 bajtov. Následne hodnota tohto kľúča je vygenerovaná cez uloženú hodnotu v malware. Postupne prebieha XOR operácia cez celý blok 6.9. Následne prebieha druhé dešifrovanie dát cez rovnaký algoritmus. Tento algoritmus sa nedá jednoducho identifikovať, pretože je relatívne jednoduchý a používa dešifrovanie založené na jednobajtovej XOR operácii.

```
length = blockLength;
indexToKey = 0;
indexToBlock = 0;
newBlock = HeapAlloc(blockLength);
keyLength = 16;

memmove(&key, originalKey, keyLength); // move original key into key
do {
    length--;

    if (indexToKey == 0) {
        key = xor(key,definedChunk); // xor every byte with each other
    }

    newBlock[indexToBlock] = originalBlock[indexToBlock] ^ key[indexToKey];
    indexToBlock++;
    indexToKey = (indexToKey + 1) % keyLength;
} while (length);
```

Obr. 6.9: Dyg – Pseudokód pre prvé dešifrovanie

Nakoniec prebieha skok do poslednej časti, kde sa načítajú potrebné závislosti. Na obrázku je vidieť ako táto časť vypadá 6.10.



Obr. 6.10: Dyg – Začiatok dešifrovanej časti

V dešifrovanej časti nastáva ešte dodatočné dešifrovanie a následná alokácia ďalších blokov cez `VirtualAlloc` a `VirtualProtect`, do ktorých je vložená ďalšia odšifrovaná časť spolu so skopírovanými sekciami už z odšifrovaných častí. Je to z toho dôvodu, že tento Cryptic sa snaží vyhnúť detekcii tým, že sa spolieha na fungovanie len v pamäti procesu, čo je jedna z najefektívnejších techník ako sa dá vyhnúť detekcii pri antivírusoch. [23]

6.3 Cryptic Marmimo

Za dobu, čo sme tento Cryptic zaznamenali bolo nájdených unikátnych 6000 vzorkov, s tým, že tam počítame aj vzorky dodávané na služby poskytujúce detekciu vzorkov pre užívateľov, kde je otestovaný daný vzorok na sade antivírusov. Príkladom takejto služby je VirusTotal[1]. Prevalentným malwarom je typ Downloader.

Vstupný bod 6.11 tohoto Crypticu obsahuje zaregistrovanie triedy resp. v tomto prípade konkrétne okna. Toto okno má požadovanú procedúru, ktorá spracováva správy posielané danému oknu.

```
int main() {
    // Set message handling
    newWindow.lpfnWndProc = messageHandlingProcedure;
    newWindow.lpszClassName = "Marmimo";
    RegisterClassExW(&newWindow);
    if (!isCreatedWindow())
        return 0;

    while (GetMessageW(&messageObject)) {
        // Process messages
    }
}
```

Obr. 6.11: Marmimo – Pseudokód pre vstupný bod

Samotná procedúra, ktorá je zodpovedná za spracovanie správ zaslaných danému zaregistrovanému oknu nie je ničím špeciálnym význačna. Jedina vetva, ktorá spracováva správu s identifikačnou hodnotou správy 0x496 obsahuje obfuskovanú procedúru, ktorá zavolá dešifrovaciu časť 6.12.

Obfuskovaná procedúra umožní skok mainpuláciou zásobniku procesoru skok do kontext v ktorom sa začne dešifrovať blok dát.

```
messageHandlingProcedure(MessageId) {
    if (MessageId == 0x496) {
        obfuscatedProcedure();
    }
    // another checks for other message identifications
}
```

Obr. 6.12: Marmimo – Volanie obfuskovanej procedúry

Dešifrovanie obsahu je vykonávané algoritmom 6.13. V rámci tohto algoritmu prebieha dešifrovanie jednotlivých blokov dát. Jedná sa v tomto prípade znova o jednoduchú XOR smyčku.

[1] https://www.virustotal.com/gui/home/upload

```
i = 0;
do {
    finished = lengthBlock;
    decryptedBlock[i] = decryptedBlock[i] ^ keyBlock[i];
    i++;
    lengthBlock--;
} while (finished != 1);
```

Obr. 6.13: Marmimo – Dešifrovací algoritmus

Ihneď po dešifrovaní obsahu je následovaný skok na miesto, do ktorého bol vložený daný dešifrovaný kód. Cryptic tu vykonáva činnosť načítavania potrebných knižníc pomocou volania `LoadLibrary` procedúry a následne načítavanie daných procedúr poskytovanými týmito knižnicami. Na obrázku 6.14 je binárny kód, ktorý danú činnosť vykonáva, kde je viditeľné volanie procedúry `GetProcAddress`, ktorá získava ukazateľ na procedúru `swprintf`.

```
00401A2B   8BF3          mov esi,ebx
00401A2D   AC            lodsb
00401A2E   84C0          test al,al
00401A30   75 FB         jne fd2540f5fde75ab6dd56f2103ecf7b0eba
00401A32   AC            lodsb
00401A33   FEC8          dec al
00401A35   74 0F         je fd2540f5fde75ab6dd56f2103ecf7b0ebaa
00401A37   FEC8          dec al
00401A39   74 14         je fd2540f5fde75ab6dd56f2103ecf7b0ebaa
00401A3B   4E            dec esi
00401A3C   56            push esi
00401A3D   FF75 F0       push dword ptr ss:[ebp-10]
00401A40   FF55 F8       call dword ptr ss:[ebp-8]
00401A43   50            push eax
00401A44   EB E7         jmp fd2540f5fde75ab6dd56f2103ecf7b0eba
00401A46   56            push esi
00401A47   FF55 FC       call dword ptr ss:[ebp-4]
00401A4A   8945 F0       mov dword ptr ss:[ebp-10],eax
00401A4D   EB DE         jmp fd2540f5fde75ab6dd56f2103ecf7b0eba
00401A4F   8B45 F4       mov eax,dword ptr ss:[ebp-C]
```

Obr. 6.14: Marmimo – Strojový kód načítavajúci procedúry z knižníc

Cryptic následne vytvorí súbor, ktorý je spustiteľný s názvom `murzuja.exe` a vytvorí ho na ceste `AppData/Temp/Local`. Je následovaná procedúra `CreateProcess`, ktorá vytvorí proces pre daný súbor, v ktorom sa nachádza finálny dešifrovaný obsah.

Tento Cryptic použil techniky ako obfuskáciu kódu pre zmanipulovanie statického analyzátoru, snažil sa analytikovi spomaliť čas prostredníctvom manipulácie zásobníku a následných skokov medzi nezvyčajnými miestami. Nakoniec sa snažil načítavať jednotlivé knižnice a procedúry dynamicky v pamäti procesu, aby sa vyhol potenciálnej detekcii antivírusom.

6.4 Cryptic Bango

Táto rodina Crypticu za týždeň začiatkom roka mala 2025 8 vzorkov, ktoré pochádzajú od verejne dostupných zdrojov pre dodanie malwaru podobné službe VirusTotal. Malware, ktorý sa pomocou tohto prevažne šíri, je Trojan.

Vstupný bod obsahuje vysoký počet zbytočného kódu. V rámci tejto varianty Crypticu je to najvýznamnejšia použitá technika. Na obrázku 6.15 je ukázaný spôsob ako tento Cryptic sa snaží volať Win API procedúry a volá rovnaké funkcie v cykle. Je to z dôvodu,

aby ovplyvnil potenciálnu dynamickú analýzu a zťažoval efektivitu tejto analýzy. Táto technika sa volá API Hammering.

```
counter = 0x100;

do {
    // Call useless procedure from Win API
    GetCommandLineA(); // Win API procedure which does nothing in context
} while (counter != 0);
```

Obr. 6.15: Bango – API Hammering

Cryptic vykonáva dešifrovanie dát pomocou daného algoritmu 6.16. Dešiforvací algoritmus sa najviac podobá na obfuskovanú RC4 implementáciu. [9] Tieto dáta, ktoré sa dešifrujú, sú dešifrované v pamäti procesu.

```
for (index = 0; index < lengthBlock; index++) {
    indexToKey = (indexToKey + 1) % 256;
    pointer = &data[4 * indexToKey];
    value = pointer[0];
    indexToKey2 = (indexToKey2 + indexToKey) % 256;
    pointer2 = &data[4*indexToKey2];
    pointer[0] = pointer2[0];
    pointer2[0] = value;
    decryptedBlock[index] ^= data[4*((value + pointer[0]) % 256)]
}
```

Obr. 6.16: Bango – Dešifrovacia rutina

Načítavanie knižníc prebieha až v momente, keď sa prejde do alokovaného a dešifrovaného kusu kódu. Je tu významný počet volaní pomocou procedúry `GetProcAddress` 6.17.

```
handleLibrary = LoadLibrary(libraryName);

index = 0
for (procedureName in getNeededProcedures(handleLibrary)) {
    addressProcedures[index] = GetProcAddress(procedureName);
    index++;
}
```

Obr. 6.17: Bango – Pseudokód načítavania procedúr

6.5 Cryptic Delegate

Tento cryptic je .NET cryptic. V rámci vzorkov ktoré prenáša bolo zistené, že sa jedná hlavne o PWS typ malwaru.

V prvej fáze je jednoduché dešifrovanie využitím RC2 implementácie v .NET knižniciach ako je ukázané na obrázku 6.18. Na obrázku stojí za povšimnutie fakt, že hodnoty parametrov pre RC2 implementáciu sa získavajú z hešovacej tabuľky, ktorej získané hodnoty sú zakódované natvrdo v kóde. V 6.18 je `array` kľúč, `array2` inicializačný vektor parameter a `array3` dešifrovaný payload. [14]

```
private static byte[] FirstDecryption(byte[] payload)
{
    byte[] array = Convert.FromBase64String(HashTable.GetStoredValue(6210));
    byte[] array2 = Convert.FromBase64String(HashTable.GetStoredValue(3341));
    byte[] array3;
    using (RC2CryptoServiceProvider rc2CryptoServiceProvider = new RC2CryptoServiceProvider())
    {
        rc2CryptoServiceProvider.Key = array;
        rc2CryptoServiceProvider.IV = array2;
        array3 = rc2CryptoServiceProvider.CreateDecryptor().TransformFinalBlock(payload, 0, payload.Length);
    }
    return array3;
}
```

Obr. 6.18: Delegate – Prvá časť dešifrovania

Pomocou volania `Assembly.Load` .NET knižniciach je načítaný payload s názvom Zcslbibt.dll. Tento payload 6.19 má zaujímavú štruktúru týkajúcu sa veľkého počtu využitia tried, ktorých bázová trieda je `MulticastDelegate`, ktorých názov je náhodný. Tieto triedy patria do koreňového menného priestoru. Táto skutočnosť má za následok fakt, že sa z týchto tried stávajú efektívne metódové ukazovatele, ktoré nepriamo volajú určitú procedúru alebo funkciu priamo poskytovanú Windows API alebo metódu z payloadu. Týmto sa dosiahne komplexná zložitosť analýzy z dôvodu, že počet týchto ukazovateľov je zhruba 80.

Obr. 6.19: Delegate – Štruktúra payloadu

Každý ukázateľ, má statický konštruktor, čo vykoná operácie ako je vidno na obrázku 6.20. Najprv zavolá `PrepareProcessIfFirstDelegateCalled`, čo zaistí skutočnosť,

aby sa mohla spraviť inicializačná sekvencia nastavenia stavových premien. Následuje volanie `SetPointer`, umožňujúce nastaviť metódu, na ktorú ukazuje daný ukazateľ podľa `TypeHandle`, ktoré získa zo samotného typu. Táto hodnota `TypeHandle` slúži na určenie kľúča, ktorý bude použitý na získanie hodnoty z hash mapy. Hodnota daného kľúča je skutočná metóda, na ktorú bude ukazovať daný ukazateľ. Ukázateľe v koreňovom mennom

```
static SpecialDelegate()
{
    MainClass.PrepareProcessIfFirstDelegateCalled();
    MainClass.SetPointer(typeof(SpecialDelegate).TypeHandle);
}
```

Obr. 6.20: Delegate – Konštruktor ukázateľa

priestore majú meno jednej metódy rovnaké. Táto metóda má v rôznych videných vzorkoch rôzne meno, ale platí, že meno tejto metódy 6.21 je pre každý ukázateľ rovnaké. Parametrom bývajú najprv smerom zľava-doprava argumenty od prvého argumentu do metódy, na ktorú odkazuje ukázateľ.

```
public static string O7KOXmCw7E(ref byte param1, string param2, SpecialDelegate methodPointer)
{
    return methodPointer(ref param1, param2);
}
```

Obr. 6.21: Delegate – Volanie ukázateľa

Jedna metóda sa v rámci procesu modifikuje a následne poslúži ako skok do natívnych procedúr a funkcií patriacich Windows API. Táto modifikovaná metóda je zodpovedná za volania jednotlivých ukazovateľov.

Postupným volaním vytvorí takzvaný suspendovaný proces, ktorého rodičom je proces `explorer.exe`, ktorého obsah je program spustený v aktuálnom momente, to znamená, že si vytvorí vlastnú kópiu.

V rámci vytvorenej kópie odšifruje dáta cez obfuskovaný kód skokmi využitím AES šifry a následne prinúti skok do vstupného bodu, ktorým je dešifrovaný PE súbor. [17] Po skončení payloadu tento proces zruší a ukončí svoju činnosť.

Celá hlavná metóda, ktorá je spúšťaná, je obfuskovaná pomocou switch case konštrukcie 6.22, kde každý elementárny krok je čo najviac rozmiestnený pod iným caseom. Na obrázku možno vidieť, že natvrdo dané hodnoty do casu sú porovnávané s premennou `num2`.

```
case 29:
    num3 = 176 - 58;
    num2 = 73;
    if (MainClass.ReturnNull() == null)
    {
        num2 = 539;
        continue;
    }
    continue;
case 30:
    array7[num8 + 2] = (byte)((num9 & 16711680U) >> 16);
    num2 = 88;
    continue;
case 31:
    num3 = 175 - 58;
    num2 = 612;
    if (!MainClass.AlwaysTrue())
    {
        num2 = 575;
        continue;
    }
    continue;
case 32:
    num3 = 203 - 67;
    num2 = 48;
    if (!MainClass.AlwaysTrue())
    {
        num2 = 43;
        continue;
    }
    continue;
case 33:
    array6 = MainClass.GetBytes(MainClass.ToInt32(num10));
    num2 = 606;
    continue;
```

Obr. 6.22: Delegate – Obfuskácia hlavnej metódy

Získané poznatky z analýzy

Pri ukážke častí analýzy vyššie popísaných rodín bolo zistené, ako efektívne analyzovať tento typ hrozby. Pre analýzu je potrebné vykonávať a kombinovať statickú a dynamickú analýzu. Samotná statická analýza je nedostatočná z dôvodu, že kód je vo významnej miere obfuskovaný a obsahuje nadmerné množstvo častí, ktoré sa snažia zťažiť analýzu analytikom. Dynamická analýza pomôže túto časť zminimalizovať vďaka schopnosti identifikovať a pomôcť pri dešifrovaných častiach, ktoré taktiež môžu obsahovať dôležité aspekty ako sú načítavacia časť potrebných závislostí pre daný payload. [18]

Kapitola 7

Metódy pre obranu proti Crypticom

Za účelom efektívnej ochrany pred hrozbami typu Cryptic je potrebné tieto hrozby identifikovať a klasifikovať. Na tento účel bol vybraný jazyk YARA, ktorý sa ako nástroj v oblasti detekcie malwaru etabloval ako robustný nástroj, umožňujúci definovať pravidlá na základe charakteristických vzorov správania.

7.1 YARA

Pre aplikovanie týchto postupov tu máme jazyk YARA. YARA je silný a všeobecne využiteľný nástroj používaný primárne na identifikovanie a klasifikovanie rodín malwaru pomocou napísaných pravidiel v tomto jazyku. [22]

Tento nástroj umožňuje kyberbezpečnostným profesionálom vytvárať popisy malwarových rodín, ktorých špecifickým znakom je použitie textových alebo binárnych vzorov. [22]

Štruktúra pravidla YARA

Klasické YARA pravidlo má typicky 3 sekcie

1. meta sekcia
2. strings sekcia
3. condition sekcia

Sekcia `meta` slúži na popis metadát. Tieto metadata môžu byť autor, popis, dátum, typ malwareu, heš vzorky na základe ktorého bolo toto pravidlo vytvorené. [21]

Sekcia `strings` definuje reťazce, hexadecimálne vzory alebo regulárne výrazy, ktoré sú hľadané pre daný cieľ. [21]

Sekcia `condition` je zodpovedná za rozhodnutie toho, kedy je dané pravidlo vyhovujúce danému vzorku. [21]

Na obrázku 7.1 je ukážka pravidla, ktorého forma je použitá na tvorbu klasifikačných pravidiel Crypticov. V `meta` sekcii je použitý popis `author`, ktorý hovorí o autorovi daného pravidla. Popis `severity` značí, o aký typ klasifikácie sa jedná. Hodnoty môžu byť `clean` alebo `malware`. Popis `type` určuje, aký typ malwaru klasifikujeme. Pre určenie rodiny malwaru je použitý popis `strain`.

```
rule malware_rule
{
    meta:
        author = "author of rule"
        description = "Short description of malware rule"
        severity = "malware"
        type = "malware_type"
        strain = "family_strain"
    strings:
        $byte_pattern = {AB CD EF}
    condition:
                $byte_pattern
}
```

Obr. 7.1: Ukážka YARA pravidla

7.2 Klasifikácia Crypticu pomocou YARA pravidiel

Keď sa snažíme klasifikovať Cryptic, je potrebné si rozobrať, ako by sa to dalo pomocou YARA pravidiel spraviť.

Každý Cryptic môže mať špecifickú vlastnosť, ktorá sa môže využiť pre klasifikáciu. Konkrétne sa jedná o pravidlá zamerané na statické vlastnosti alebo dynamické vlastnosti Crypticu.

V rámci statického pravidla je možné použiť špecifické statické dáta, ktoré Cryptic môže zanechávať. Môžu to byť špecifické stringy súvisiace s dešifrovacím procesom alebo môžu to byť kľúče, ktoré môžu byť na určitých bytoch rovnaké alebo majú špeciálny vzťah, ktorý je možné využiť. Pri ukázanej analýze týchto Crypticov, je vhodné premýšľať nad statickým pravidlom pomocou hľadania špecifických dešifrovacích rutín. Dešifrovacie rutiny môžu byť relatívne efektívny spôsob klasifikácie statického pravidla. Je to z toho dôvodu, že aj keď Cryptic môže naberať mnoho statických foriem týchto dešifrovacích rutín, tak sémantika týchto rutín môže byť identická pri odlišných variantách. Môže sa jednať o statické kľúčové hodnoty, ktoré sa používajú kvôli špecifickým vlastnostiam, s ktorými autor mohol pri tvorbe danej dešifrovacej rutiny počítať a tým ovplyvniť celkové fungovanie daného dešifrovania. Samotné šifrované dáta, ktoré Cryptic zašifroval, môžu mať špecifické vzory. Našou metódou ako by bolo vhodné postupovať, je pri analýze úspešne detekovať dešifrovaciu rutinu a túto rutinu následne použiť ako záchytný bod, okolo ktorého sa celá statická klasifikácia bude vytvárať.

Hypotézou je, že dešifrovacia rutina dokáže poskytnúť dostatočný podklad pre efektívne odhalenie takýchto Crypticov, ktoré nie sú *Metamorfické*. [23].

Pre dynamickú obranu je potrebné si dať pozor na správanie týchto hrozieb v infikovanom systéme. Je pri tomto postupe náročné jednoznačne identifikovať, či správanie v rámci daného systému odpovedá nelegitímnemu správaniu. Táto forma obrany vyžaduje veľkú pozornosť pri určovaní detekcie. Pre Windows je najtypickejším spôsobom, ako by to bolo možné spraviť, použitie poznatkov o pomenovaných objektoch na tejto platforme. Pri akomkoľvek vytváraní objektov udalostí, zámkov s výlučným prístupom, tried a kontextových okien je možné, že špecifický pomenovaný objekt môže pri odlišných vzorkách, ale pri tej is-

tej rodine Crypticov, byť rovnaký. Týmto spôsobom je možné najefektívnejšie detekovať daný Cryptic.

Pre napísanie YARA pravidiel je potrebné rozhodnúť, aký typ pravidla je vhodné použiť. Pri Cryptic rodinách Havefun, Dyg, Marmimo a Bango bolo vidieť, že sa pomerne jednoducho dali identifikovať dešifrovacie rutiny, ktoré neboli nijakou špeciálnou formou obfuskované, preto sme rozhodli, že pravidlá týchto 4 variánt by mali dostatočne stačiť ako statické pravidlo zachytávajúce binárnu sekvenciu inštrukcií popisujúcu špecifickú časť týchto dešifrovacích rutín.

Pre rodinu Crypticu Bango je ukážkovo viditeľný spôsob, ako je možné klasifikovať Cryptic pomocou známej dešifrovacej rutiny. YARA pravidlo 7.2 ukazuje, ako sa detekujú 4 varianty dešifrovacej rutiny použitej pri tejto rodine. Každá dešifrovacia rutina predstavuje premennú v sekcii `strings` s názvom `h00`, `h01`, `h02` a `h03`. Jednotlivé dešifrovacie rutiny sú popísané v strojovom kóde, preto bolo nutné použiť bajtové vzory. Toto klasifikačné pravidlo bude platiť vtedy, keď sa aspoň 1 dešifrovacia rutina v rámci PE súboru objaví.

```
rule bango_cryptic_known_sequences
{
    ...
    strings:
        $h00 = { 76 63 8D 43 01 8B CE 99 F7 F9 8B FE 8B DA 8B 84 ( 1D | 5D
            | 9D | DD ) ?? ?? ?? ?? 8D 8C ( 1D | 5D | 9D | DD ) ?? ?? ?? ??
            03 45 FC 99 F7 FF 8A 01 89 55 FC 8D BC ( 15 | 55 | 95 | D5 ) ??
            ?? ?? ?? 8B 94 ( 15 | 55 | 95 | D5 ) ?? ?? ?? ?? 89 11 0F B6 D0
            8B 45 18 89 17 8B 7D 14 03 C7 89 45 14 8B 01 03 C2 8B CE 99 F7
            F9 8B 45 14 8A 8C ( 15 | 55 | 95 | D5 ) ?? ?? ?? ?? 30 08 47 3B
            7D 10 89 7D 14 72 9D 8B 45 18 }
        $h01 = ...
        $h02 = ...
        $h03 = ...
    condition:
        ...
        any of ($h0*)
}
```

Obr. 7.2: Detekcia dešifrovacej rutiny

Ukážkou YARA pravidla, ktoré používa reťazce, je pravidlo 7.3. Konkrétne sa jedná o pravidlo popisujúce triedy nachádzajúce sa v .NET PE súbore v medzikóde. Existencia všetkých tried popísaných v pravidle je nevyhnutnosťou.

```
rule dark_tortilla_cryptic_known_sequences
{
    ...
    strings:
        $s00 = "Class1_PreStart"
        $s01 = "Class2_Computer"
        ...
        $s08 = "Class9_FakeMessage"
    condition:
        ...
        all of ($s0*)
}
```

Obr. 7.3: Detekcia reťazcov

Resource Cryptic YARA pravidlo 7.4 je ukážka ako sa dá zachytiť pravidlo podľa formátu PE tohoto Crypticu. V rámci tohoto pravidla je viditeľné volanie `pe.imports` funkcie z modulu `pe`, čo nám dokáže popísať vlastnosť ktorá popisuje, čo Cryptic k základnej funkcionalite používa. Ďalšia popísaná vlastnosť, ktorá nám pomôže klasifikovať Cryptic je využitie hodnoty entropie zašifrovaného bloku v rámci jedného PE súboru ako je vidieť volanie `math.entropy` funkcie z modulu `math`. Informácia, ktorá je pri tomto Crypticu dôležitá je využitie poľa `pe.resources` aby bolo možné zistiť existenciu zdroja v PE súbore, ktorý je špecifický svojím typom a identifikačným číslom

```
rule resource_cryptic_known_sequences
{
    ...
    condition:
        ...
        for any resource_index in (0 .. pe.number_of_sections - 1) :
        (
            ...
            math.entropy(pe.sections[resource_index].raw_data_offset,
                pe.sections[resource_index].raw_data_size) >= 7.2
        ) and
        ...
        pe.imports("kernel32.dll", "FindResourceA") and
        ...
        for any payload in pe.resources :
        (
            payload.type == 10 and
            payload.id == 2
        )

}
```

Obr. 7.4: Detekcia štruktúry PE súboru

Delegate Cryptic pravidlo 7.5 je zaujímavé popisom vzťahu kódu samotného ako je možné v rámci .NET hrozieb popísať prostredníctvom CLI[1]. V pravidle je využitie skutočnosti existencie tried, ktoré dedia od triedy `MulticastDelegate` s využitím poľa `classes`. Pravidlo je komplexnejšie v tom, ako musí využiť pozretie existencie tried, ktoré sa nachádzajú aspoň 2-krát v .NET súbore, ktorých bázová trieda je `MulticastDelegate` a ich menný priestor je kmeňový.

```
rule delegate_cryptic_known_sequences
{
    ...
    condition:
    ...
    for 2 class in dotnet.classes :
    (
        class.namespace == "" and
        for any type in class.base_types :
        (
            type contains "MulticastDelegate"
        )
    )
    ...
}
```

Obr. 7.5: Detekcia vlastností kódu

V rámci ostatných YARA pravidiel sú tieto popísané techniky využité pri ostatných pravidlách. Pravidlá v prílohach využívajú iba statické vlastnosti, kvôli dôvodom spomenutým vyššie.

[1] https://ecma-international.org/publications-and-standards/standards/ecma-335/

Kapitola 8

Experimentálne výsledky

V rámci tejto bakalárskej práce bolo vytvorených celkovo 17 YARA pravidiel, ktoré boli zamerané na detekciu 14 rôznych rodín malwaru typu Cryptic. Tieto pravidlá boli navrhnuté a implementované na základe analýzy štruktúr, správania a spoločných znakov medzi vzorkami jednotlivých malwarových rodín.

Nasledujúce časti kapitoly poskytujú detailný pohľad na výsledky detekcií, ktoré boli získané pri aplikácii YARA pravidiel na reálne malware vzorky. Výsledky sú prezentované formou početnosti detekcií jednotlivých vzoriek a rozdelenia typov malwaru podľa ich klasifikácie. Okrem toho sú uvedené aj štatistiky o množstve vytvorených detekčných signatúr, ktoré boli následne využité v bezpečnostných riešeniach firmy Gen, pričom všetky vzorky boli čerpané z interných zdrojov firmy.

Každá analyzovaná rodina Cryptic bola podrobená detailnej analýze a súčasťou tejto kapitoly je popis výsledkov pre každú z nich.

Dyg Cryptic

Pre rodinu Dyg bolo pomocou YARA pravidla v prílohe C detekovaných 1114 vzoriek. Na základe analýzy typov malwaru sa ukázalo, že najčastejším typom, ktorý bol týmto typom malwaru šírený, je malware typu PWS (Password Stealer). Ďalším, menej početne zastúpeným typom bol Trojan. Výsledky poukazujú na špecifickú orientáciu tejto rodiny Crypticu na krádež prihlasovacích údajov.

Havefun Cryptic

Varianta A

V tejto variante bolo pomocou YARA pravidla v prílohe A identifikovaných 1879 vzoriek, čo naznačuje pomerne vysokú prevalenciu tejto verzie Crypticu. Typové rozdelenie detekovaného malwaru je rôznorodé. Najvyšší podiel predstavuje PWS, nasledovaný Trojanom, Infostealerom a RAT.

Varianta B

Na rozdiel od varianty A, táto varianta vďaka YARA pravidlu v prílohe B vykazuje úzke zameranie na jeden typ malwaru – PWS. Z celkového počtu 285 vzoriek bolo všetkých 285 klasifikovaných ako tento typ, čo svedčí o jej špecializácii.

Asabf Cryptic

Rodina Asabf predstavuje Cryptic napísaný v jazyku Batch, pričom jeho hlavným cieľom je šírenie malwaru typu Dropper. Z 24 detekovaných vzoriek využitím YARA pravidla v prílohe J ich až 23 patrilo do tejto kategórie. Zvyšný prípad bol klasifikovaný ako Trojan. Táto dominancia naznačuje úzke zameranie na sťahovacích hrozieb.

SimdaPacked Cryptic

Rodina SimdaPacked bola použitá na šírenie jedinej známej rodiny malwaru typu Backdoor – konkrétne Simda. Počet detekcií pre túto kombináciu výrazne prevyšuje 260000, čo naznačuje vysokú mieru prevalencie a aktivity tejto rodiny v zaznamenaných hitoch. YARA pravidlá sú dve varianty, kde jedna varianta v prílohe P sa snažila pokryť videné časti dešifrovacích rutín a druhá varianta videnú ikonu v prílohe Q.

Dxpack Cryptic

Dxpack je špecifická rodina, ktorá zahŕňa len jednu malwarovú rodinu s názvom Dinwod. Tento Cryptic slúži výlučne na prenos malwaru typu Trojan. Celkový počet detekcií YARA pravidlom v prílohe N pre túto kombináciu dosiahol výrazných 7 029 619, čo z nej robí jednu z najrozšírenejších v celom súbore analyzovaných Crypticov.

Blopprod Cryptic

Varianta A

Varianta A tejto rodiny vykazuje dominanciu dvoch typov malwaru – Trojan a RAT. Trojan bol najčastejšie detekovaným typom. Celkovo bolo zachytených 1281 vzoriek pomocou YARA pravidla v prílohe F, pričom Trojan tvoril väčšinu prípadov.

Varianta B

Na rozdiel od varianty A, druhá varianta tejto rodiny vďaka YARA pravidlu v prílohe G preukázala dominantné zastúpenie malwaru typu Dropper, ktorý tvoril 191 z celkových 192 detekovaných vzoriek. Jeden prípad bol klasifikovaný ako RAT.

Marmimo Cryptic

Rodina Marmimo sa špecializuje na distribúciu malwaru typu Downloader. V tomto prípade bolo identifikovaných 7545 prípadov využitím YARA pravidla v prílohe D, čo naznačuje jej efektívnosť pri šírení tohto typu škodlivého softvéru.

Matic Cryptic

Rodina Matic je reprezentantom .NET Crypticov. Celkovo bolo detekovaných 513 prípadov využitím YARA pravidla v prílohe H, pričom až 500 z nich bolo klasifikovaných ako PWS. To poukazuje na veľmi špecifické zameranie tejto rodiny.

Delegate Cryptic

V tejto rodine Crypticov bolo celkovo detekovaných 2302 vzoriek pomocou YARA pravidla v prílohe M. Najčastejšie typy zahŕňajú PWS, RAT a Infostealer. PWS mal pritom najvýznamnejšie zastúpenie s počtom 566 detekovaných prípadov.

Rug Cryptic

Rodina Rug vykazuje veľmi úzke zameranie na šírenie Infostealerov, ktoré tvorili 81 z 136 zachytených vzoriek YARA pravidlom v prílohe K. Zvyšné prípady boli klasifikované ako PWS.

Gru Cryptic

Rodina Gru zahŕňa širšie spektrum typov malwaru. Celkovo bolo detekovaných 1388 vzoriek využitím YARA pravidla v prílohe O, z ktorých najväčší podiel tvoril RAT (1303 prípadov). Okrem toho boli detekované aj prípady Rootkitu (66) a Coinminer (19).

DarkTortilla Cryptic

DarkTortilla je ďalší .NET Cryptic, ktorého primárnym cieľom je distribúcia RAT. Z celkových 130 detekovaných prípadov pomocou YARA pravidla v prílohe I až 112 predstavoval typ malwaru RAT.

Resource Cryptic

Táto rodina zahŕňa prevažne typy malwaru PWS (48 prípadov) a Infostealer (16 prípadov), pričom celkový počet detekcií dosiahol 64 pomocou YARA pravidla v prílohe L.

Zhrnutie rodín

Na tabuľke 8.1 je možné vyvodiť fakt, že väčšina rodín Crypticu, ktoré v rámci tejto práce boli klasifikované a detekované cez YARA pravidlá, šírila aspoň 2 rôzne rodiny malwaru z určených typov malwaru.

Najúspešnejší je podľa počtu hitov v tabuľke 8.1 práve rodina Dxpack, po nej následuje rodina SimdaPacked. Najväčší počet typov malwaru ktoré šírila Cryptic rodina je Havefun varianta A. S počtom rodín, ktoré boli nájdené sa dá vydedukovať, že 5 Cryptic rodín šírilo 4 rodiny malwaru.

YARA Pravidlo	Počet hitov	Počet typov	Počet rodín
Asabf	24	2	2
Blopprod/A	1 281	2	2
Blopprod/B	192	2	3
DarkTortilla	130	2	4
Delegate	2 302	3	3
Dyg	1 114	2	3
Dxpack	7 029 619	1	1
Gru	1 388	3	4
Havefun/A	1 879	4	4
Havefun/B	285	1	1
Marmimo	7 545	1	1
Matic	513	2	4
Resource	64	2	4
Rug	136	2	2
SimdaPacked/A	172 382	1	1
SimdaPacked/B	98 009	1	1

Tabuľka 8.1: Štatistika YARA pravidiel pri detekcii hrozieb typu `Cryptic`

8.1 Detekčné signatúry

Detekčné signatúry pri týchto YARA pravidlách boli v určitom počte. Každé YARA pravidlo a jeho hity vytvorili podnet na vytvorenie a klasifikovanie detekčných signatúr, ktoré majú dosah na používateľov bezpečnostných riešení firmy Gen. Táto tabuľka 8.2 ukazuje počet detekčných signatúr, ktoré boli spozorované so vzorkami, ktoré hitovali YARA pravidlá od začiatku roku 2025 a vďaka ktorým boli zablokované útoky na klientov bezpečnostných riešení firmy Gen.

Z tabuľky 8.2 vyplýva, že najviac signatúr má pravidlo Havefun, konkrétne sa jedná o variantu A. Ďalšie relatívne významné YARA pravidlo, ktoré bolo pri detekciách vzorkov videné so signatúrami, je Blopprod, varianta A. Výnimkou je pravidlo Havefun, varianta B, ktoré ešte do obdobia písania tejto práce nebolo videné s vytvorenými signatúrami pri detekovaných vzorkoch.

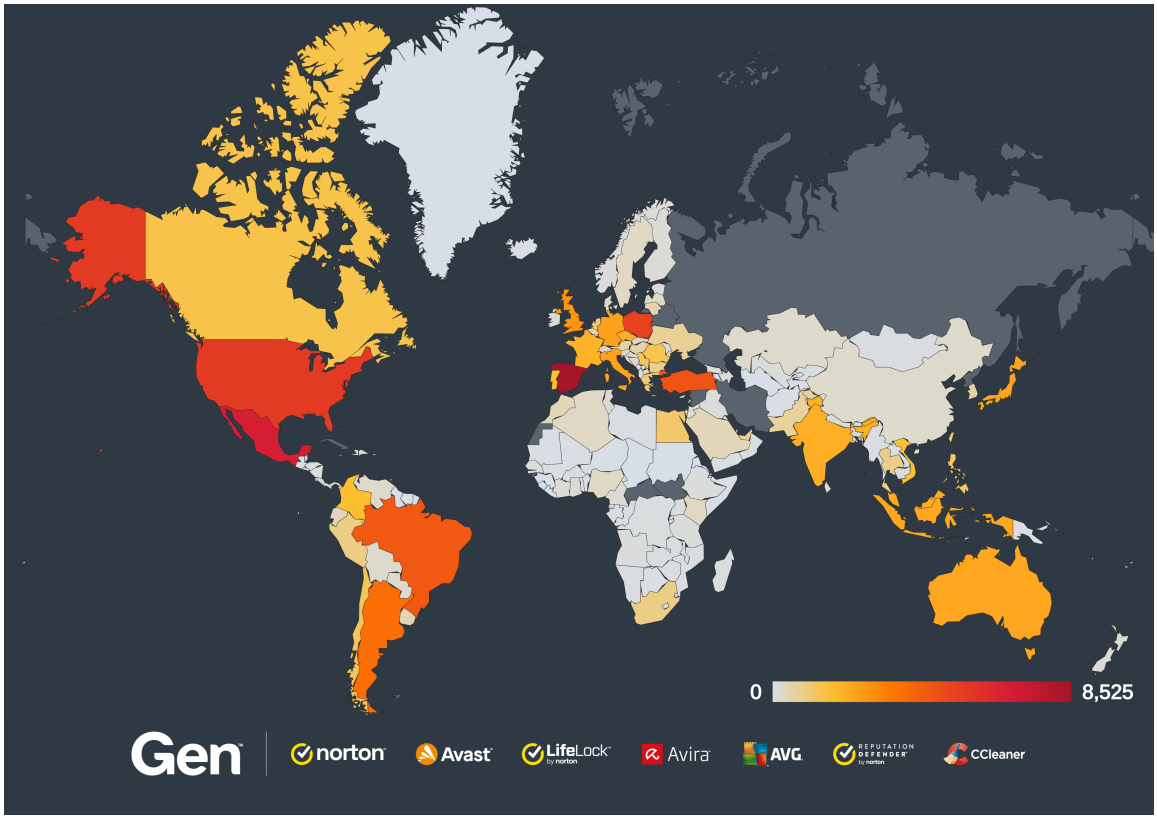
Na tabuľke 8.2 je vidieť najväčší počet hitov pre Havefun variantu A. Hneď nasleduje DarkTortilla s 9798 hitmi. Najmenej zase malo Havefun varianta B pravidlo. Keďže ešte nebola spozorovaná žiadna detekčná signatúra pri vzorkoch, ktoré hitovalo YARA pravidlo. Za predpokladu, že by sa táto varianta znova vrátila vo veľkom počte, tak by sa dal predpokladať výskyt vzniku detekčných signatúr vďaka hitom z YARA pravidla pre detekciu varianty.

YARA Pravidlo	Naviazané signatúry	Zablokované útoky na klientovi
Asabf	12	14
Blopprod/A	95	1 367
Blopprod/B	19	219
DarkTortilla	16	9 798
Delegate	9	602
Dyg	33	1 223
Dxpack	4	159
Gru	34	4 905
Havefun/A	151	49 577
Havefun/B	0	0
Marmimo	8	153
Matic	42	513
Resource	9	1 281
Rug	5	216
SimdaPacked/A	2	17
SimdaPacked/B	6	19

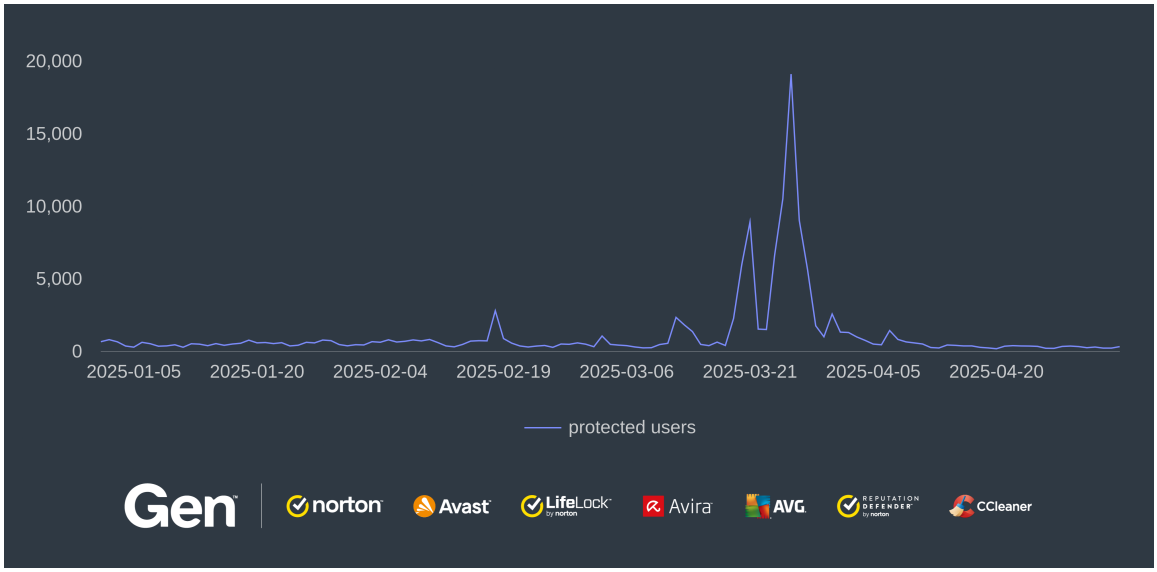
Tabuľka 8.2: Štatistika zablokovaných útokov podľa YARA pravidiel na klientovi

Na obrázku 8.1 je viditeľná tepelná mapa znázorňujúca počet zablokovaných útokov na klientov podľa jednotlivých štátov, kde boli na zablokovanie použité naviazané detekčné signatúry na vytvorené YARA pravidlá v prílohách, ktoré sú obsiahnuté v tabuľke 8.2. V rámci jednotlivých štátov je viditeľné, že Španielsko, Spojené štáty Americké a Poľsko boli najviac ochránené týmito hrozbami vďaka naviazaným signatúram, ktoré zablokovali útoky.

Na obrázku 8.2 je viditeľný graf znázorňujúci na časovej osi počet zablokovaných útokov v danom čase, ktoré boli zablokované vďaka detekčným signatúram, ktoré sú obsiahnuté v tabuľke 8.2. Obrázok 8.2 naznačuje, že najviac zablokovaných útokov na klientov bezpečnostných riešení firmy Gen je medzi obdobím konca marca a začiatku apríla.



Obr. 8.1: Tepelná mapa zablokovaných útokov



Obr. 8.2: Zablokované útoky vďaka signatúram v čase

8.2 Zhodnotenie

Výsledky experimentálnej časti jednoznačne ukazujú, že rodiny malwaru typu Cryptic slúžia ako nosiče pre rôzne škodlivé typy malwaru. Najčastejšie prenášanými typmi malwaru sú PWS a RAT, ktoré dominujú vo viacerých analyzovaných rodinách.

YARA pravidlá boli vytvorené so zámerom maximalizovať pokrytie známych škodlivých vzoriek. Väčšina pravidiel je zameraná na detekciu statických vzorov, ako sú bajtové sekvencie, metadata a reťazce, ktoré sa opakovane vyskytujú v týchto rodinách malwaru.

Zároveň však treba poznamenať, že vytváranie behaviorálnych pravidiel by si vyžadovalo ďalší výskum, pretože viaceré Cryptic rodiny nevykazujú jednoznačné správanie, ktoré by bolo možné spoľahlivo detekovať bez zvýšeného rizika výskytu falošných hitov.

Z pohľadu zlepšenia detekčných schopností sa javí ako najefektívnejšia stratégia zamerať sa na ďalšie monitorovanie správania Crypticov a postupnú modifikáciu statických vzorov, ktoré súvisia s ich dešifrovaním a načítavaním škodlivého payloadu. Možnosť, ako by sa dalo pokrytie zlepšiť, je aj pokus o vytvorenie viacerých druhov pravidiel pre jednotlivé rodiny Crypticov, aby sme pokryli každý možný aspekt špecifický pre danú rodinu.

Tabuľka 8.2, obrázok 8.1 a obrázok 8.2 ukazujú, že YARA pravidlá umožnili zlepšiť pokrytie vysoko obfuskovaných hrozieb na strane klientov firmy Gen pri prevalentných hrozbách v súčasnosti.

Kapitola 9

Záver

V tejto práci sme si vysvetlili problematiku malwarových hrozieb. Ukázali sme si metódy pre analýzu kódu, ako je reverzné inžinierstvo, kde sme si popísali statickú analýzu a dynamickú analýzu. V rámci toho popisu sme si ukázali nástroje, ktoré sú v dnešnej dobe používané pri takýchto analýzach kódu. Vysvetlili sme si, ako je sandboxing možné pre dynamickú analýzu využiť a prečo je aj sandboxing dôležitý. Popísali sme si dve metódy pre detekciu malwarových hrozieb.

Cryptic bol vysvetlený ako typ malwaru, ktorého hlavnou úlohou je vyhnúť sa detekciám antivírusov a aby pri dynamickej analýze zťažil prácu analytikom. Zistili sme, že Cryptic sa najefektívnejšie bráni kvôli šifrovaniu kódu hlavného programu, ktorý sa následne spustí. Ukázali sme si takú možnú analýzu Crypticu, kde sme zistili, aké sú obfuskačné techniky používané.

V rámci celej práce bolo zanalyzovaných 14 Cryptic rodín. Počas analýzy sme zistili a identifikovali podstatné body, ktoré sú špecifické pre tento typ malwaru. Identifikovali sme načítavaciu časť, dešifrovaciu časť a skok do pôvodného vstupného bodu.

Ako možný prostriedok na detekciu a klasifikáciu rodín Crypticov sme si zvolili známy nástroj YARA. V rámci používania tohto nástroja sme si rozobrali rôzne metódy, ako sme realizovali detekciu a klasifikáciu malwaru tohto typu. Zvolili sme si statické pravidlá kvôli najjednoduchšiemu spôsobu, ako plnohodnotne detekovať Cryptic.

Celkovo sme vytvorili 17 YARA pravidiel. Zistili sme, že najpočetnejšia rodina v počte detekcií pomocou YARA pravidiel je Dxpack Cryptic. V rámci vzorkov to bolo viac ako 7 000 000 hitov. Ďalšou významnou rodinou v počte detekovaných vzorkov je rodina Simda-Packed, ktorá je zodpovedná za šírenie backdooru Simda celkovo v počte viac ako 250 000 vzorkov. V rámci uvedených rodín v kapitole 8 bolo vidieť, že väčšina rodín šírila viacero typov malwaru. Najviac bolo vidieť, že sa šíri PWS a RAT.

Neskôr bolo ukázané, že detekčné signatúry, ktorých tvorba bola ovplyvnená hitovaním vzorkov vďaka vytvoreným YARA pravidlám, mali významný dopad na počet zablokovaných útokov vo svete. Celkovo sa nám podarilo od začiatku roka 2025 zabrániť vyše 50 000 útokom na klientov firmy Gen.

Z čoho najväčší podiel tvorila rodina Havefun varianta A o vyše 49 000 zablokovaných útokoch. Druhý najväčší podiel mala rodina DarkTortilla s počtom 9 700 zablokovaných útokov.

Vďaka vykonanej štatistike je viditeľný výsledok zlepšenia detekčného pokrytia pre malware typu Cryptic analyzovaných rodín.

Aby sme sa pokúsili zlepšiť pokrytie detekcií pri týchto rodinách. Mali by sme pozerať, ako sa tieto rodiny vyvíjajú a poprípade iterovať nad existujúcimi pravidlami a vylepšiť

a viac sa pokúsiť zovšeobecniť dané pravidlá. Možným ďalším spôsobom, ako by sa dal dosiahnúť výsledok, je pokúsiť sa nájsť potenciálne záchytné body, ktoré by pomohli určiť behaviorálny aspekt týchto rodín.

Literatúra

[1] *Crypter* online. 2017. Dostupné z: https://www.malwarebytes.com/glossary/crypter. [cit. 2025-07-01].

[2] Arntz, P. *Explained: Packer, Crypter, and Protector* online. 2017. Dostupné z: https://www.malwarebytes.com/blog/news/malware/2017/03/explained-packer-crypter-and-protector. [cit. 2025-07-01].

[3] Avast. *NHS Hospital Ransomware* online. 2021. Dostupné z: https://www.avast.com/business/resources/what-is-hospital-ransomware#pc. [cit. 2025-07-01].

[4] Avast. *RATs, Rootkits, and Ransomware, Oh My.* 2025. Dostupné z: https://blog.avast.com/rats-rootkits-and-ransomware-oh-my.

[5] Chaz, L.; Platon, K.; Davide, B.; Juan, C. a Manos, A. A Lustrum of Malware Network Communication: Evolution and Insights. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, s. 788–804. ISBN 978-1-5090-5533-3.

[6] Eilam Eldad, C. E. J. *Reversing: secrets of reverse engineering.* IN. Indianapolis, 2005. ISBN 0-7645-7481-7.

[7] Elastic Security Labs. *Dissecting REMCOS RAT: An in-depth analysis of a widespread 2024 malware, Part One.* Marec 2024. Dostupné z: https://www.elastic.co/security-labs/dissecting-remcos-rat-part-one.

[8] Fellicious, C.; D'Antoni, L.; Bianchi, A. a Balzarotti, D. *A Large-Scale Dataset of Windows API Calls for Malware Detection.* 2024. Dostupné z: https://arxiv.org/abs/2502.12863.

[9] Fontaine, C. RC4. In: *Encyclopedia of Cryptography, Security and Privacy.* Cham: Springer Nature Switzerland, 2025, s. 2070–2072. ISBN 978-3-030-71522-9. Dostupné z: https://doi.org/10.1007/978-3-030-71522-9_365.

[10] Fraunhofer FKIE. *Win.coinminer.* 2025. Dostupné z: https://malpedia.caad.fkie.fraunhofer.de/details/win.coinminer. Accessed: 2025-05-11.

[11] Intel471. *A Briefing on Malware Crypting Services* online. 2024. Dostupné z: https://intel471.com/blog/a-briefing-on-malware-crypting-services. [cit. 2025-07-01].

[12] Microsoft. *Introduction to .NET.* 2025. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction.

[13] Microsoft. *PE Format* online. 2025. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format. [cit. 2025-07-01].

[14] Rivest, R. L. *A Description of the RC2(r) Encryption Algorithm* RFC 2268. RFC Editor, marec 1998. Dostupné z: https://doi.org/10.17487/RFC2268.

[15] Sekoia TDR, L. T. *The Architects of Evasion: a Crypters Threat Landscape* online. 2024. Dostupné z: https://blog.sekoia.io/the-architects-of-evasion-a-crypters-threat-landscape/. [cit. 2025-07-01].

[16] Sikorski, A. *Practical Malware Analysis: a Hands-On Guide to Dissecting Malicious Software*. Professional. No Starch Press„ 2012. ISBN 978-1593272906.

[17] Standards, N. I. of; (NIST), T.; Dworkin, M. J.; Barker, E.; Nechvatal, J. et al. *Advanced Encryption Standard (AES)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2001-11-26 00:11:00 2001. Dostupné z: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=901427.

[18] Szor, P. *The Art of Computer Virus Research and Defense*. Professional. Addison-Wesley, 2005. ISBN 978-0321304544.

[19] Teoderick Contreras, S. T. R. T. *From Registry with Love: Malware Registry Abuses*. 2023. Dostupné z: https://www.splunk.com/en_us/blog/security/from-registry-with-love-malware-registry-abuses.html.

[20] Trend Micro. *Crypter* online. 2025. Dostupné z: https://www.trendmicro.com/vinfo/us/security/definition/crypter. [cit. 2025-07-01].

[21] VirusTotal. *Writing YARA rules* online. 2025. Dostupné z: https://yara.readthedocs.io/en/latest/writingrules.html. [cit. 2025-07-01].

[22] Wikipedia. *YARA* online. 2025. Dostupné z: https://en.wikipedia.org/wiki/YARA. [cit. 2025-07-01].

[23] Yehoshua, U. *Antivirus Bypass Techniques: Learn practical techniques and tactics to combat, bypass, and evade antivirus software*. Limited. Packt Publishing, 2021. ISBN 13: 9781801079747.

Prílohy

Príloha A

Cryptic Havefun varianta A

```
import "pe"

rule havefun_cryptic_a_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "havefun"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = { 66 C1 EF FD F7 E1 0F B7 DB 66 03 FB C1 DA 4B C1 D8 35 C1 CA 1B 66
            C1 DA 5E 4B 66 03 CE 0F B6 FF 2B F8 66 C1 EB 7B 66 BF 83 01 C1 E8 5C
            66 41 81 F6 71 01 00 00 66 C1 D1 FC 33 F7
            }
    condition:
        pe.is_pe and
        $h00
}
```

Príloha B

Cryptic Havefun varianta B

```
import "pe"

rule havefun_cryptic_b_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "havefun"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = { 33 F6 BA ?? ?? ?? ?? E8 ?? ?? ?? ?? 8B D8 8B 0B 8B 49 04 8B 4C 19
            30 8B 79 04 8B CF 89 ( 7C 24 ?? | BC 24 ?? ?? ?? ?? ) 8B 17 FF 52 04
            8D ( 44 24 ?? | 84 24 ?? ?? ?? ?? ) 50 E8 ?? ?? ?? ?? 83 C4 04 8B C8
            8B 10 6A 0A 8B 42 20 FF D0 88 44 24 ?? 85 FF 74 13 8B 07 8B CF FF 50
            08 85 C0 74 08 8B 10 8B C8 6A 01 FF 12 }
    condition:
        pe.is_pe and
        $h00
}
```

Príloha C

Cryptic Dyg

```
import "pe"

rule dyg_cryptic_known_sequences {
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "dyg"
    strings:
        $h00 = { 4A 85 C9 75 14 0F 10 44 24 34 0F 28 0D ?? ?? ?? ?? 0F 57 C8 0F 11
            4C 24 34 8B 5C 24 14 8A 44 0C 34 32 03 88 45 00 45 FF 44 24 14 41 83
            E1 0F 85 D2 75 CD 8B 5C 24 20 }
        $h01 = { 4D 85 C9 75 0C 0F 57 0D ?? ?? ?? ?? 0F 11 4C 24 34 8B 5C 24 14 8A
            44 0C 34 32 03 88 02 42 FF 44 24 14 41 83 E1 0F 85 ED 75 D6 }
    condition:
        pe.is_pe and
        any of them
}
```

Príloha D

Cryptic Marmimo

```
import "pe"


rule marmimo_cryptic_known_sequences {
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "marmimo"
    strings:
        $h00 = { 51 8B 07 4E 8B C8 AD 4A 4E 32 E1 52 8A C4 4E AA 58 85 C0 75 07 FF
            75 10 8B 55 14 5E 59 E2 E1 41 C3 }
        $h01 = { 55 8B EC 83 EC 14 8B 45 0C 85 C0 74 09 56 57 E8 ?? ?? ?? ?? 5F 5E
            8B E5 5D C2 10 00 }
    condition:
        pe.is_pe and
        $h00 and $h01
}
```

Príloha E

Cryptic Bango

```
import "pe"


rule bango_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "bango"
    strings:
        $h00 = { 76 63 8D 43 01 8B CE 99 F7 F9 8B FE 8B DA 8B 84 ( 1D | 5D | 9D |
            DD ) ?? ?? ?? ?? 8D 8C ( 1D | 5D | 9D | DD ) ?? ?? ?? ?? 03 45 FC 99
            F7 FF 8A 01 89 55 FC 8D BC ( 15 | 55 | 95 | D5 ) ?? ?? ?? ?? 8B 94 (
            15 | 55 | 95 | D5 ) ?? ?? ?? ?? 89 11 0F B6 D0 8B 45 18 89 17 8B 7D 14
            03 C7 89 45 14 8B 01 03 C2 8B CE 99 F7 F9 8B 45 14 8A 8C ( 15 | 55 |
            95 | D5 ) ?? ?? ?? ?? 30 08 47 3B 7D 10 89 7D 14 72 9D 8B 45 18 }
        $h01 = { 76 63 8D 43 01 8B CE 99 F7 F9 8B FE 8B DA 8B 44 ( 1D | 5D | 9D |
            DD ) ?? 8D 4C ( 1D | 5D | 9D | DD ) ?? 03 45 FC 99 F7 FF 8A 01 89 55
            FC 8D 7C ( 15 | 55 | 95 | D5 ) ?? 8B 54 ( 15 | 55 | 95 | D5 ) ?? 89 11
            0F B6 D0 8B 45 18 89 17 8B 7D 14 03 C7 89 45 14 8B 01 03 C2 8B CE 99
            F7 F9 8B 45 14 8A 4C ( 15 | 55 | 95 | D5 ) ?? 30 08 47 3B 7D 10 89 7D
            14 72 9D 8B 45 18 }
        $h02 = { 76 63 8D 43 01 8B CE 99 F7 F9 8B FE 8B DA 8B 85 ?? ?? ?? ?? 8D 8D
            ?? ?? ?? ?? 03 45 FC 99 F7 FF 8A 01 89 55 FC 8D BD ?? ?? ?? ?? 8B 95
            ?? ?? ?? ?? 89 11 0F B6 D0 8B 45 18 89 17 8B 7D 14 03 C7 89 45 14 8B
            01 03 C2 8B CE 99 F7 F9 8B 45 14 8A 8D ?? ?? ?? ?? 30 08 47 3B 7D 10
            89 7D 14 72 9D 8B 45 18 }
        $h03 = { 76 63 8D 43 01 8B CE 99 F7 F9 8B FE 8B DA 8B 45 ?? 8D 4D ?? 03 45
            FC 99 F7 FF 8A 01 89 55 FC 8D 7D ?? 8B 55 ?? 89 11 0F B6 D0 8B 45 18
            89 17 8B 7D 14 03 C7 89 45 14 8B 01 03 C2 8B CE 99 F7 F9 8B 45 14 8A
            4D ?? 30 08 47 3B 7D 10 89 7D 14 72 9D 8B 45 18 }
    condition:
        pe.is_pe and
        any of ($h0*)
}
```

Príloha F

Cryptic Blopprod varianta A

```
import "pe"

rule blopprod_cryptic_a_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "BlopprodCryptic"
    strings:
        $h00 = { 8B 45 90 03 45 DC 8A 08 88 8D 18 FE FF FF 8B 55 98 8A 44 15 D8 88
            85 14 FE FF FF 8A 85 14 FE FF FF 50 8A 85 18 FE FF FF 66 5B 32 C3 88
            85 34 FE FF FF 8D 8D 14 FE FF FF 89 8D 28 FE FF FF 8D 95 18 FE FF FF
            89 95 1C FE FF FF 8D 85 38 FE FF FF 89 85 20 FE FF FF 8B 8D 20 FE FF
            FF 8A 95 34 FE FF FF 88 11 8B 45 F8 8B 88 ?? ?? ?? ?? 89 8D 10 FE FF
            FF 8B 95 10 FE FF FF 03 55 DC 8A 85 38 FE FF FF 88 02 E9 4C FF FF FF }
    condition:
          pe.is_pe and
        $h00
}
```

Príloha G

Cryptic Blopprod varianta B

```
import "pe"

rule blopprod_cryptic_b_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "BlopprodCryptic"
    strings:
        $h00 = { C6 85 14 FE FF FF 00 8B 45 DC 89 85 20 FE FF FF 6A 01 8B 4D 90 03
            8D 20 FE FF FF 51 8D 95 14 FE FF FF 52 E8 FB 37 00 00 83 C4 0C 8B 45
            98 8A 4C 05 D8 88 8D 10 FE FF FF 8A 85 10 FE FF FF 50 8A 85 14 FE FF
            FF 66 5B 32 C3 88 85 34 FE FF FF 8D 95 10 FE FF FF 89 95 28 FE FF FF
            8D 85 14 FE FF FF 89 85 18 FE FF FF 8D 8D 38 FE FF FF 89 8D 1C FE FF
            FF 8B 95 1C FE FF FF 8A 85 34 FE FF FF 88 02 8B 4D F8 8B 91 ?? ?? ??
            ?? 89 95 0C FE FF FF 8B 85 0C FE FF FF 03 45 DC 8A 8D 38 FE FF FF 88
            08 E9 2F FF FF FF }
    condition:
        pe.is_pe and
        $h00
}
```

Príloha H

Cryptic Matic

```
import "dotnet"

rule matic_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known sequences"
        strain = "MaticCryptic"
        type = "cryptic"
        severity = "malware"
    strings:
        $s00 = "CypherMatic" ascii fullword
        $s01 = "SYYM" ascii fullword
    condition:
        dotnet.is_dotnet and
        all of them
}
```

Príloha I

Cryptic DarkTortilla

```
import "dotnet"

rule dark_tortilla_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known sequences"
        reliability = "report"
        severity = "malware"
        type = "cryptic"
        strain = "DarkTortilla"
    strings:
        $s00 = "Class1_PreStart"
        $s01 = "Class2_Computer"
        $s02 = "Class3_Zone"
        $s03 = "Class4_Reader"
        $s04 = "Class5_Decrypter"
        $s05 = "Class6_GetOptions"
        $s06 = "Class7_GlobalOptions"
        $s07 = "Class8_AntiVMs"
        $s08 = "Class9_FakeMessage"
    condition:
        dotnet.is_dotnet and
        all of ($s0*)
}
```

Príloha J

Cryptic Asabf

```
rule asabf_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "AsabfCryptic"
        type = "cryptic"
        severity = "malware"
    strings:
        $s00 = "messageFileName =
            ASaBf.getNamedItem(\"messageFileName\").nodeValue" wide
        $s01 = ".save( wanMX )" wide
        $s02 = "Set ASaBf = JoGki.attributes" wide
        $s03 = "ExpandEnvironmentStrings(\"%TEMP%\")" wide
        $s04 = ".SelectNodes(\"//*/provider\")" wide
    condition:
        filesize < 1000KB and
        all of them
}
```

Príloha K

Cryptic Rug

```
import "pe"

rule rug_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "RugCryptic"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = { B2 EB EC 6D 53 7A D1 3D 9D 06 A3 69 62 A8 1B B9 9E 33 37 60 0A D4
            35 09 10 3C 53 EC D6 B0 EA A7 FD D6 33 D6 BD E8 A8 2E 6C 29 D4 87 BE
            C4 AE 20 AA 8F 7E D9 AF E1 DD 18 60 }

    condition:
        pe.is_pe and
        all of them
}
```

Príloha L

Cryptic Resource

```
import "math"
import "pe"

rule resource_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known sequences"
        severity = "malware"
        type = "cryptic"
        strain = "ResourceCryptic"
    condition:
        pe.is_pe and
        pe.number_of_sections == 5 and
        for any resource_index in (0 .. pe.number_of_sections - 1) :
        (
            pe.sections[resource_index].name == ".rsrc" and
            math.entropy(pe.sections[resource_index].raw_data_offset,
                pe.sections[resource_index].raw_data_size) >= 7.2
        ) and
        pe.number_of_resources >= 1 and
        pe.imports("kernel32.dll", "FindResourceA") and
        pe.imports("kernel32.dll", "LoadResource") and
        pe.imports("kernel32.dll", "SizeOfResource") and
        pe.imports("kernel32.dll", "GetProcAddress") and
        pe.imports("kernel32.dll", "Sleep") and
        pe.imports("kernel32.dll", "QueryPerformanceCounter") and
        pe.imports("kernel32.dll", "QueryPerformanceFrequency") and
        for 5 icon_index in (0 .. pe.number_of_resources - 1) :
        (
            pe.resources[icon_index].type == pe.RESOURCE_TYPE_ICON
        ) and
        for any payload in pe.resources :
        (
            payload.type == 10 and
            payload.id == 2
        )

}
```

Príloha M

Cryptic Delegate

```
import "dotnet"

rule delegate_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known sequences"
        severity = "malware"
        type = "cryptic"
        strain = "DelegateCryptic"
    condition:
        dotnet.is_dotnet and
        dotnet.number_of_resources >= 5 and
        // 2 resources have 5040 B and 256 B respectively
        for any resource in dotnet.resources :
        (
            resource.length == 5040
        ) and
        for any resource in dotnet.resources :
        (
            resource.length == 256
        ) and
        for all class in dotnet.classes :
        (
            (
                class.namespace != "" or
                not
                (
                    for any type in class.base_types :
                    (
                        type contains "MulticastDelegate"
                    )
                )
            ) or
            (
                for any target_method in class.methods :
                (
                    target_method.name != "Invoke" and
                    target_method.name != ".cctor" and
```

```
                for all other_class in dotnet.classes :
                (
                    (
                        other_class.namespace != class.namespace or
                        not
                        (
                            for any type in other_class.base_types :
                            (
                                type contains "MulticastDelegate"
                            )
                        )
                    ) or
                    (
                        for any other_method in other_class.methods :
                        (
                            other_method.name == target_method.name
                        )
                    )
                )
            )
        ) and
        for 2 class in dotnet.classes :
        (
            class.namespace == "" and
            for any type in class.base_types :
            (
                type contains "MulticastDelegate"
            )
        ) and
        for any class in dotnet.classes :
        (
            class.namespace endswith ".Properties" and
            class.name == "Resources"
        )
}
```

Príloha N

Cryptic Dxpack

```
import "pe"

rule dxpack_cryptic_known_sequences {
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        severity = "malware"
        type = "cryptic"
        strain = "DxpackCryptic"
    strings:
        $h00 = { 55 8B EC 8B 45 0C 53 56 8B 75 08 03 C6 89 45 08 8B 45 14 33 DB 89
            18 8A 16 8B 45 10 57 8B CE 80 FA 11 76 1D 0F B6 FA 83 EF 11 8D 4E 01
            83 FF 04 0F 82 AA 00 00 00 8A 11 88 10 40 41 4F 75 F7 EB 5F 0F B6 31
            41 83 FE 10 0F 83 B2 00 00 00 85 F6 75 16 EB 07 81 C6 FF 00 00 00 41
            80 39 00 74 F4 0F B6 11 8D 74 16 0F 41 8B 11 89 10 6A 04 5A 03 C2 03
            CA 4E 74 2A 3B F2 72 1D 8B 39 89 38 2B F2 03 C2 03 CA 3B F2 73 F2 85
            F6 74 14 8A 11 88 10 40 41 4E 75 F7 EB 09 8A 11 88 10 40 41 4E 75 F7
            0F B6 31 41 83 FE 10 73 57 0F B6 11 C1 EA 02 C1 E6 06 8D B4 32 01 07
            00 00 8B F8 2B FE 8A 17 88 10 8A 57 01 88 50 01 8A 57 02 41 88 50 02
            83 C0 03 }
    condition:
        pe.is_pe
        and
        any of them
}
```

Príloha O

Cryptic Gru

```
import "pe"

rule gru_cryptic_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "GruCryptic"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = { 66 00 75 00 6E 00 63 00 74 00 69 00 6F 00 6E 00 20 00 4C 00 6F 00
            63 00 61 00 6C 00 3A 00 47 00 65 00 74 00 2D 00 44 00 65 00 6C 00 65
            00 67 00 61 00 74 00 65 00 7B 00 50 00 61 00 72 00 61 00 6D 00 28 00
            5B 00 4F 00 75 00 74 00 70 00 75 00 74 00 54 00 79 00 70 00 65 00 28
            00 5B 00 54 00 79 00 70 00 65 00 5D 00 29 00 5D 00 5B 00 50 00 61 00
            72 00 61 00 6D 00 65 00 74 00 65 00 72 00 28 00 50 00 6F 00 73 00 69
            00 74 00 69 00 6F 00 6E 00 3D 00 30 00 29 00 5D 00 5B 00 54 00 79 00
            70 00 65 00 5B 00 5D 00 5D 00 24 00 50 00 61 00 72 00 61 00 6D 00 65
            00 74 00 65 00 72 00 54 00 79 00 70 00 65 00 73 00 2C 00 5B }

    condition:
        pe.is_pe and
        all of them
}
```

Príloha P

Cryptic SimdaPacked varianta A

```
import "pe"

rule simda_packed_a_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "SimdaPacked"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = { AC 34 56 8D 1D 08 00 00 00 8D 0D 4C 00 00 00 C7 45 D8 12 00 00 00
            83 C9 13 34 65 BB 7D 00 00 00 81 C2 01 00 00 00 81 C1 02 00 00 00 03
            DF 4A 81 C2 02 00 00 00 03 D8 02 C4 8D 8B 0D 00 00 00 83 C9 2F 89 7D
            F0 89 7D F4 81 E9 02 00 00 00 2B CE B9 68 00 00 00 43 2A C4 81 C1 01
            00 00 00 8D 99 08 00 00 00 F7 D3 81 EB 02 00 00 00 B9 95 00 00 00 66
            21 DA 8B D9 43 32 C4 F7 D9 81 C1 01 00 00 00 8B D0 81 C2 02 00 00 00
            81 EB 02 00 00 00 03 CE 8D 0D A4 00 00 00 4B 43 49 04 52 2B DF 81 C2
            02 00 00 00 03 D6 2B 1D 2D A7 40 00 81 C3 02 00 00 00 43 2A C4 8D 0D
            09 00 00 00 8B D0 89 4D E0 81 EA 01 00 00 00 03 D3 03 D1 21 F2 49 2C
            AB 81 EB 02 00 00 00 81 C3 02 00 00 00 8D 8B F0 00 00 00 87 D1 03 CE
            03 CB 02 C4 8B D6 8D 8A 06 00 00 00 8B D0 33 D9 0F C9 B9 25 00 00 00
            33 DD 66 BA DC 52 4A 34 28 89 5D FC BA 58 00 00 00 81 C2 02 00 00 00
            81 E9 01 00 00 00 81 EA 02 00 00 00 33 DB 34 8A 0F AF CA 81 E2 E1 00
            00 00 BA AE 00 00 00 89 5D D0 8D 9A 03 00 00 00 BB 08 00 00 00 03 CB
            49 34 8B 89 7D F8 8B D6 66 BB 70 D2 C7 45 E0 05 00 00 00 89 45 DC 8D
            8B 0B 00 00 00 33 15 1D A3 40 00 8B C8 32 C4 81 FF 0E 63 2B 69 75 0C }
        $h01 = { AC C0 C0 02 83 CB 67 81 C3 01 00 00 00 49 87 C9 8B D3 BB 68 00 00
            00 03 CA 2C E8 81 EA 02 00 00 00 4A 81 EA 01 00 00 00 8D 0D A8 00 00
            00 2B 1D 84 24 41 00 2B 15 53 28 41 00 2B D6 83 CA 73 02 C4 42 8D 1D
            6A 00 00 00 41 03 DF 03 DF 8B DF 81 EA 02 00 00 00 3B 1D E9 25 41 00
            89 4D FC 03 CF 2B 0D CB 27 41 00 02 C4 BA 88 00 00 00 B9 AB 00 00 00
            C7 45 F0 2D 00 00 00 BB F5 00 00 00 2B D9 8B DF C0 C8 02 C7 45 F8 35
            00 00 00 8B CA 03 CF 81 C1 01 00 00 00 81 EB 02 00 00 00 81 C2 02 00
            00 00 2B D6 83 C9 03 04 52 81 EB 02 00 00 00 66 21 EA 83 C9 56 83 C9
            6F 8D 15 00 00 00 00 8D 1D 06 00 00 00 BA 7D 00 00 00 89 55 F4 4B 04
            A5 33 DD 8D 90 6E 00 00 00 81 EB 02 00 00 00 8B DE B9 A7 00 00 00 33
            1D E2 2C 41 00 8B D6 C0 C0 03 8D 0D DC 00 00 00 83 FE A1 72 09 }
```

```
$h02 = { AC 32 C4 83 CB 13 81 C2 01 00 00 00 66 21 C1 BA 34 00 00 00 8D 91
    B9 00 00 00 4B 43 34 77 81 EA 01 00 00 00 3B 15 7E 19 49 00 89 45 F0
    89 55 C8 03 D1 66 09 CB C0 C8 03 03 DA 0F 9D C2 81 C1 01 00 00 00 8B
    D9 2B D1 89 4D D8 4B 89 75 C0 0F 9C C3 8D 1D 5C 00 00 00 2B C8 03 C8
    34 55 89 75 B8 81 C1 01 00 00 00 03 CE BB B3 00 00 00 81 C1 01 00 00
    00 8B CF 81 C3 02 00 00 00 4A 04 98 66 21 F3 8D 15 0B 00 00 00 81 EA
    02 00 00 00 B9 84 00 00 00 2B DF 8B CE 1B D9 42 C0 C0 01 81 C3 02 00
    00 00 03 C8 81 C3 02 00 00 00 81 EA 01 00 00 00 41 2C 4D 81 C2 02 00
    00 00 8D 99 0E 00 00 00 81 C3 02 00 00 00 03 CA 8B CA 49 04 6B C7 45
    F0 78 00 00 00 87 CB 81 C1 01 00 00 00 89 45 D4 66 B9 55 7F 2A C4 81
    C2 02 00 00 00 81 EA 02 00 00 00 2B CF 21 C2 89 7D D8 03 D0 4B 02 C4
    03 D1 2B DA 03 D0 81 C2 02 00 00 00 81 E9 02 00 00 00 C7 45 F4 72 00
    00 00 03 CB 2C 4B 8D 8A 3F 00 00 00 8D 15 0E 00 00 00 2B DF 81 EA 02
    00 00 00 8B D3 81 C3 02 00 00 00 C0 C0 03 85 D8 74 03 }
$h03 = { AC 34 F2 BA 8D 00 00 00 81 C2 02 00 00 00 B9 47 00 00 00 2B DA 8B
    D9 42 34 8F C7 45 B4 4B 00 00 00 2B 0D 87 D2 43 00 8D 1D C6 00 00 00
    BA DC 00 00 00 83 CA 32 42 04 EB 89 45 9C 66 09 C3 83 CA 0A 89 4D FC
    8B CA 87 D9 3B 1D 75 F3 43 00 33 D8 34 E6 83 C9 1F 8D 97 BD 00 00 00
    8B C8 BA 88 00 00 00 B9 B0 00 00 00 81 C3 01 00 00 00 03 DF 83 CB 26
    C0 C0 01 83 F8 FA 7B 07 89 5D B4 83 C9 5A 43 0F CA 83 CB 0B 8D 88 45
    00 00 00 8D 15 90 00 00 00 87 CA 2A C4 8D 0D 89 00 00 00 81 E1 6C 00
    00 00 03 DA 33 0D 10 D5 43 00 4A 02 C4 C7 45 98 53 00 00 00 8D 0D 27
    00 00 00 8D 9F 84 00 00 00 0F CA 81 EA 02 00 00 00 66 21 DA B9 04 00
    00 00 C0 C8 01 B9 0B 00 00 00 81 C3 02 00 00 00 33 15 EA DF 43 00 81
    C3 02 00 00 00 81 C1 02 00 00 00 BB 2B 00 00 00 83 CB 0A 2A C4 81 C1
    01 00 00 00 66 81 FF B1 44 78 04 }
$h04 = { 8B 35 34 D5 43 00 89 35 CC D9 43 00 C7 05 0D D6 43 00 13 04 00 00
    81 25 0D D6 43 00 FF 00 00 00 8B 15 0D D6 43 00 89 15 B0 D7 43 00 BB
    36 0A 00 00 89 5D FC 2B 1D F8 C4 43 00 53 8F 05 78 DA 43 00 6A 45 6A
    00 6A 00 6A 66 68 76 C1 43 00 FF 15 74 72 40 00 A3 F5 DA 43 00 8D 35
    06 C2 43 00 8B FE B9 0E 00 00 00 AC C0 C0 11 2C 24 C0 C0 08 AA 83 E9
    01 0F 85 ED FF FF FF 8D 3D C0 C0 43 00 C6 07 36 C6 47 01 7E C6 47 02
    66 C6 47 03 3D C6 47 04 3B C6 47 05 01 C6 47 06 34 C6 47 07 22 C6 47
    08 5D 8D 15 3F C1 43 00 C6 02 4C C6 42 01 6E C6 42 02 75 C6 42 03 68
    C6 42 04 64 C6 42 05 1B C6 42 06 22 C6 42 07 1D 58 5E 5F C9 C2 08 00 }
$h05 = { AC C0 C8 03 8D 15 6E 00 00 00 2B DA 81 C2 01 00 00 00 03 CE 49 8D
    1D 0F 00 00 00 BB 42 00 00 00 89 75 F8 8B C8 03 D1 02 C4 89 75 B8 03
    D1 8B DA 89 75 EC 8D 15 7D 00 00 00 8D 92 0B 00 00 00 4B 02 C4 3B 15
    51 DC 40 00 81 EA 01 00 00 00 8D 0D 37 00 00 00 0B 0D EA D7 40 00 2B
    D1 8B D0 03 C8 32 C4 C7 45 B4 65 00 00 00 81 C3 02 00 00 00 81 EB 02
    00 00 00 2B D6 2C F7 03 D1 87 D1 2B DF 83 CB 7B 03 D9 81 C1 01 00 00
    00 8B DA 03 D0 2C 86 81 C3 02 00 00 00 83 CA 70 83 FE 81 72 06 89 45
    A0 8B CE 42 2B D6 02 C4 66 81 F9 C4 32 7F 08 2B D0 8D 9A C5 00 00 00
    2B D9 03 D1 49 99 C0 C8 03 }
$h06 = { AC 2A C4 8D 15 0E 00 00 00 C7 45 AC 30 00 00 00 8D 99 02 00 00 00
    8D 0D 05 00 00 00 81 C1 01 00 00 00 2A C4 8B D9 81 C3 01 00 00 00 33
    D1 2B 15 58 3A 41 00 BB 0F 00 00 00 2B DF 03 D7 8B DE C7 45 E4 75 00
    00 00 4B 02 C4 0B 15 A9 3C 41 00 C7 45 FC 2C 00 00 00 0F B3 CA 89 45
    E0 8B D1 C0 C8 03 81 EA 02 00 00 00 0F CA 8B D7 42 8D 1D 02 00 00 00
    3B 0D B4 37 41 00 43 4B C0 C8 01 66 81 F9 95 57 7B 02 87 CA 8D 96 14
    00 00 00 BA 8E 00 00 00 4B 04 D0 0F C9 81 C3 01 00 00 00 81 C3 02 00
    00 00 81 E9 02 00 00 00 4A 02 C4 85 DE 75 03 89 75 E4 }
$h07 = { AC 33 D5 8D 15 00 00 00 00 85 C2 74 01 49 8D 15 00 00 00 00 BB 00
    00 00 00 2B D2 F7 D3 85 C2 74 04 3C F5 2B C9 33 D8 C1 EA 02 8D 0D 00
```

```
    00 00 00 87 DA 8D 0D 00 00 00 00 81 E3 00 00 00 00 33 DE 8D 1D 00 00
    00 00 8D 1D 00 00 00 00 8D 1D 01 00 00 00 8D 0D 01 00 00 00 03 C9 02
    C4 85 C3 74 04 57 5A 85 F7 8D 15 00 00 00 00 33 CF BB 00 00 00 00 C1
    E1 0C B9 00 00 00 00 52 5B 2B C8 8D 15 01 00 00 00 2B D1 BA 00 00 00
    00 B9 01 00 00 00 1B D2 21 FB BA 00 00 00 00 41 2A C4 BB 00 00 00 00
    03 C8 BB 01 00 00 00 21 C3 87 DA 87 D3 C1 EB 0E 49 8D 15 01 00 00 00
    8D 1D 01 00 00 00 C1 EA 0E 8D 15 01 00 00 00 B9 00 00 00 00 3B FD 79
    04 4B 85 C7 42 BB 00 00 00 00 21 F1 85 EB 4B 04 7F 2B DB 8D 15 01 00
    00 00 8D 0D 01 00 00 00 BA 00 00 00 00 21 CB 3B D9 79 04 49 21 CA 4B
    8D 15 01 00 00 00 99 C1 E2 09 0F BE D3 8D 1D 00 00 00 00 8D 1D 01 00
    00 00 8D 15 01 00 00 00 21 C2 21 E9 BA 00 00 00 00 BB 01 00 00 00 33
    D7 3A EB 79 06 }
$h08 = { AC 85 F0 74 02 3C CB C1 E9 09 21 D3 3C E9 79 02 8B D3 03 C9 81 E1
    01 00 00 00 BA 00 00 00 00 C1 E3 01 BA 00 00 00 00 C1 EB 12 21 F9 81
    E1 01 00 00 00 43 2C B4 BB 00 00 00 00 03 CF C1 E3 08 2B CA 8D 0D 01
    00 00 00 BB 00 00 00 00 C1 EB 10 3A D6 72 03 87 DA 43 8D 0D 00 00 00
    00 03 D2 BB 01 00 00 00 21 F2 C1 EB 1B 8D 15 00 00 00 00 8D 15 01 00
    00 00 3B CC 7F 04 3B D0 85 CA BB 00 00 00 00 81 E1 00 00 00 00 21 C2
    FE C0 3A E8 75 04 C1 E9 11 4B 81 E2 01 00 00 00 21 C3 BA 01 00 00 00
    B9 01 00 00 00 C1 E1 0F 3B F2 70 05 }
$h09 = { AC C0 C0 02 81 C2 02 00 00 00 2B D6 8D 9B 4C 00 00 00 0F AB D3 33
    0D 26 48 41 00 03 D1 03 CF 89 4D E8 34 9D 81 EB 01 00 00 00 8D 89 0B
    00 00 00 81 E9 01 00 00 00 83 F8 FE 79 06 BA CA 00 00 00 4A 66 BB 04
    23 C0 C8 01 0F C9 BB 22 00 00 00 81 EB 02 00 00 00 F7 D2 BB CD 00 00
    00 8D 8A 70 00 00 00 03 DE 42 B9 B3 00 00 00 34 45 2B CA 2B D6 8D 1D
    06 00 00 00 2B D8 89 55 EC 03 CA 81 E3 71 00 00 00 81 EB 01 00 00 00
    83 C9 7D 4B 32 C4 C7 45 DC 4E 00 00 00 03 D7 81 C1 02 00 00 00 87 C9
    2B DA 03 D9 C0 C8 02 81 EB 01 00 00 00 2B D1 03 DF 81 C1 02 00 00 00
    8D 15 95 00 00 00 0F AB D9 C0 C8 01 81 C2 01 00 00 00 B9 F5 00 00 00
    81 C1 02 00 00 00 33 D3 03 D1 32 C4 BB DB 00 00 00 03 D6 BB 00 00 00
    00 BB 13 00 00 00 66 BB D5 2A 43 8D 15 02 00 00 00 49 BB C7 00 00 00
    42 43 02 C4 81 E9 02 00 00 00 89 5D F4 03 D9 8D 9A 09 00 00 00 B9 17
    00 00 00 8D 1D E7 00 00 00 03 D8 03 D6 4B 32 C4 8B D3 83 CA 0E 66 81
    FB 64 6D 79 08 8D 88 0D 00 00 00 03 CA BB 7C 00 00 00 8B C8 42 2C 57
    2B 1D 8D 48 41 00 8D 1D 6B 00 00 00 66 BB 76 E6 B9 7C 00 00 00 83 CA
    46 8D 9B 01 00 00 00 0F 99 C2 83 CB 23 32 C4 66 BB 1A F9 81 E9 02 00
    00 00 03 CB 81 C3 02 00 00 00 33 DA 85 C3 75 03 }
$h10 = { A3 2D 4A 42 00 BF 97 05 00 00 C1 CF 14 03 3D 63 27 42 00 89 3D 30
    47 42 00 C7 05 3E 49 42 00 3F 02 00 00 83 2D 3E 49 42 00 43 8B 0D 3E
    49 42 00 89 0D 77 43 42 00 BB 02 09 00 00 81 E3 FF 0F 00 00 C1 EB 07
    89 5D D0 C7 05 7D 46 42 00 47 0E 00 00 81 25 7D 46 42 00 FF 0F 00 00
    8B 35 7D 46 42 00 89 35 6F 3F 42 00 C7 05 93 48 42 00 9A 09 00 00 FF
    35 93 48 42 00 58 89 45 BC BB 76 07 00 00 C1 E3 07 81 FB 83 05 00 00
    74 06 2B 1D 4E 30 42 00 53 8F 05 32 4A 42 00 8D 35 AF 11 42 00 8B FE
    B9 03 00 00 00 AC C0 C0 15 C0 C8 17 04 39 C0 C8 04 04 40 AA 83 E9 01
    0F 85 E8 FF FF FF }
$h11 = { AC 32 C4 0F CB 8B D6 8D 15 02 00 00 00 03 DA 8D 1D F1 00 00 00 8D
    90 09 00 00 00 43 34 F4 BB 48 00 00 00 8D 0D 07 00 00 00 03 D7 81 E9
    02 00 00 00 81 EA 01 00 00 00 8D 9A 75 00 00 00 89 45 F4 4B 4B 32 C4
    81 C3 02 00 00 00 81 E9 02 00 00 00 8B CE 81 C1 02 00 00 00 03 DF 03
    D1 81 EB 01 00 00 00 33 D5 32 C4 8D 97 05 00 00 00 2B D9 3B 0D 5B 54
    42 00 33 0D 1B 6D 42 00 42 02 C4 8D 92 04 00 00 00 1B D3 8D 90 00 00
    00 00 2B 0D 4C 6C 42 00 BA 86 00 00 00 03 DA 2C B3 81 E9 02 00 00 00
    85 FA 75 01 4B 81 C2 02 00 00 00 81 E9 02 00 00 00 2A C4 42 8D 1D 6D
```

```
    00 00 00 8B CF 81 E9 02 00 00 00 2B D9 03 D3 8D 9E 73 00 00 00 81 EB
    01 00 00 00 33 D7 32 C4 33 C8 8B CB 0F CB 81 C1 01 00 00 00 03 D6 8B
    D8 F7 D9 81 C2 02 00 00 00 4B C0 C4 02 BB A9 0B 00 00 2B DE 03 1D 32
    4D 42 00 81 EB AE 0F 00 00 C1 EB 07 C1 CB 07 8B D3 03 D2 D1 E2 2B D1
    C1 EA 03 01 15 DA 45 42 00 8B D7 42 D1 CA 81 C2 A5 02 00 00 03 15 9E
    4C 42 00 4A 81 C2 55 0D 00 00 C1 CA 08 81 EA 79 07 00 00 81 FA D8 0E
    00 00 78 03 }
$h12 = { AC C0 C0 02 81 FE A1 F1 AC 62 7E 0B 8B CF C7 45 A0 71 00 00 00 8B
    D9 21 CB 0F 90 C1 4B 02 C4 2B D3 81 E9 02 00 00 00 0B 15 62 A3 41 00
    BB 9F 00 00 00 89 5D C8 34 CE 8D 0D 0A 00 00 00 8B DA 8D 8A E7 00 00
    00 81 C1 02 00 00 00 81 EB 01 00 00 00 34 22 33 0D F9 92 41 00 BB 54
    00 00 00 81 EA 01 00 00 00 B9 AD 00 00 00 2B DA 1B CE B9 A0 00 00 00
    B9 C6 00 00 00 04 C0 8D 1D D5 00 00 00 33 0D 60 9A 41 00 81 E9 01 00
    00 00 03 CE 4A 2A C4 8B D1 81 C3 01 00 00 00 33 0D A9 8D 41 00 8D 9E
    0A 00 00 00 83 CA 66 34 60 83 C9 13 81 EB 01 00 00 00 8D 15 07 00 00
    00 81 EB 01 00 00 00 8D 0D F5 00 00 00 32 C4 C7 45 C8 6D 00 00 00 BB
    4B 00 00 00 03 D6 8D 9A 0C 00 00 00 2B DF 8D 0D CB 00 00 00 2B DA 03
    CA 32 C4 2B 0D 44 A6 41 00 03 D7 C7 45 C8 76 00 00 00 89 45 B4 8B DA
    89 7D E4 03 CF 03 D8 49 32 C4 83 C9 73 89 75 A8 8D 15 3A 00 00 00 B9
    88 00 00 00 81 EA 02 00 00 00 8D 1D 0D 00 00 00 83 C9 5F 41 2A C4 8B
    CA 81 C2 01 00 00 00 33 CE 8D 8B 08 00 00 00 8B CF 03 D6 03 D9 4B 0B
    1D 16 A2 41 00 8B CB 4A 02 C4 2B DA 8D 1D 9B 00 00 00 8D 8B 0F 00 00
    00 03 C8 83 C9 49 03 D0 42 02 C4 2B D3 81 F9 79 5F B3 CF 7C 05 }
$h13 = { AC C0 C8 02 81 EA 02 00 00 00 C7 45 D8 20 00 00 00 83 CB 27 03 CB
    C7 45 B0 23 00 00 00 83 CB 2E 42 34 94 33 0D E6 A7 43 00 2B 0D FD 18
    44 00 89 7D BC 0B 0D 12 7B 43 00 81 EA 02 00 00 00 4A 2A C4 C7 45 C0
    27 00 00 00 81 C3 01 00 00 00 03 D0 4A 8D 0D 06 00 00 00 81 C2 01 00
    00 00 89 55 E8 32 C4 8D 0D 4A 00 00 00 8D 15 4E 00 00 00 C7 45 F0 1F
    00 00 00 8D 1D 0A 00 00 00 03 D3 8D 15 04 00 00 00 42 32 C4 81 E9 01
    00 00 00 C7 45 E0 1E 00 00 00 81 C2 02 00 00 00 83 FF B8 7F 09 03 C8
    81 EA 02 00 00 00 42 03 DF 43 32 C4 81 C3 02 00 00 00 8B CE 85 CA 75
    04 33 D0 87 DA 81 EB 01 00 00 00 03 D6 2A C4 8B DA 0B 1D 8B DF 44 00
    BA 23 00 00 00 81 C1 01 00 00 00 B9 45 00 00 00 03 CF 32 C4 81 FE 84
    DE 6A E7 76 05 }
$h14 = { AC C0 C8 02 89 5D E0 81 EA 01 00 00 00 0F CB 8D 0D 57 00 00 00 8D
    88 0C 00 00 00 8D 1D 0F 00 00 00 4B 33 DF 4B 04 9C 2B D7 03 CB 83 CB
    5D B9 AE 00 00 00 85 CE 74 01 4A 81 C1 01 00 00 00 42 C0 C8 03 81 C1
    02 00 00 00 8D 0D 02 00 00 00 4B 81 EB 01 00 00 00 87 DA 4B C0 C8 02
    81 EB 02 00 00 00 BA 6C 00 00 00 BA 01 00 00 00 89 7D F0 2B CA 4A 2C
    9B 03 D9 83 CB 54 8D 0D 0F 00 00 00 2B 15 40 70 41 00 0F BE C9 49 C0
    C0 03 2B 1D 94 73 41 00 2B D7 89 75 E4 66 B9 09 C1 21 CA 8D 1D 08 00
    00 00 42 C0 C8 03 33 0D 24 82 41 00 81 EA 01 00 00 00 BB DE 00 00 00
    2B D1 8D 97 CB 00 00 00 BA E2 00 00 00 03 D1 8B D1 42 2C 03 8B DE BB
    FF 00 00 00 0B 15 62 8D 41 00 81 C3 01 00 00 00 BB BA 00 00 00 03 D0
    33 C9 4A 2B D6 C0 C0 02 8D 15 05 00 00 00 C7 45 E4 6C 00 00 00 2B D6
    8D 0D 0C 00 00 00 49 04 DB 81 EB 02 00 00 00 8D 9F 07 00 00 00 89 75
    E0 BB A5 00 00 00 4B 80 EC 4F 8B DF D1 C3 2B 1D F9 76 41 00 D1 C3 03
    DA C1 C3 05 81 C3 73 00 00 00 C1 C3 02 81 EB 1B 09 00 00 4B 01 1D C2
    71 41 00 8D 9B FC 7B 41 00 D1 CB 03 DB 81 C3 8C 0C 00 00 79 03 }
$h15 = { AC 04 E8 C7 45 FC 0F 00 00 00 81 C1 01 00 00 00 3B 1D 6C 3D 41 00
    0F BE CB 04 97 8D 99 0E 00 00 00 33 15 12 37 41 00 8D 9E 08 00 00 00
    8B D3 81 EA 02 00 00 00 87 D3 2C 69 81 C2 02 00 00 00 BB CF 00 00 00
    8B D9 89 75 AC 33 15 31 34 41 00 8B DF 33 D1 21 C1 42 99 32 C4 2B D8
    2B CA 81 EA 01 00 00 00 41 89 7D A4 8D 91 0F 00 00 00 83 C9 38 C0 C0
```

```
    01 8D 0D 0F 00 00 00 C7 45 9C 7D 00 00 00 66 B9 98 86 C7 45 F0 15 00
    00 00 33 1D 9E 3B 41 00 2B D7 83 CA 23 02 C4 8B D1 B9 8F 00 00 00 81
    C1 01 00 00 00 BA 0A 00 00 00 8B CA 43 32 C4 8D 15 F1 00 00 00 2B D8
    BA 32 00 00 00 8D 1D C0 00 00 00 B9 7D 00 00 00 03 D8 3B 1D D8 3E 41
    00 89 5D F4 32 C4 81 C1 02 00 00 00 81 E9 02 00 00 00 B9 07 00 00 00
    8D 92 04 00 00 00 0F BF D9 03 C8 03 D3 2C 5A 03 DE 03 DE 0F 9D C3 2B
    15 7A 32 41 00 03 D3 8D 0D C3 00 00 00 8D 1D 01 00 00 00 3B 0D C9 34
    41 00 2B D3 42 C0 C8 03 2B D7 03 CE 83 F9 F9 78 0C }
$h16 = { AC C0 C8 01 8D 1D 97 00 00 00 81 EB 01 00 00 00 33 DB 03 DA 3B 15
    0C 94 42 00 03 DF 0F B6 DB 2B 15 AB 9D 42 00 41 32 C4 8D 1D DF 00 00
    00 81 EA 01 00 00 00 3B 0D C0 90 42 00 81 C3 02 00 00 00 BB 0B 00 00
    00 2C C4 85 F1 75 03 8B DF 4B 81 C2 02 00 00 00 2B 15 08 A4 42 00 81
    EA 01 00 00 00 C0 C8 03 BA 6A 00 00 00 C7 45 E8 0D 00 00 00 33 CE 81
    EA 02 00 00 00 0B 15 EE A4 42 00 03 D1 42 02 C4 8D 92 23 00 00 00 8D
    9A 08 00 00 00 89 7D CC 81 EA 01 00 00 00 2A C4 81 C2 01 00 00 00 81
    EB 01 00 00 00 81 EB 01 00 00 00 1B CB 8B DF 8D 8B 0C 00 00 00 03 DE
    03 DF 2C BF 3B 15 84 B2 42 00 8B D9 03 DE 03 D6 81 C2 02 00 00 00 2B
    CE 03 CB 4A C0 C0 02 81 EB 01 00 00 00 33 D8 BB 32 00 00 00 C7 45 AC
    3A 00 00 00 41 03 D8 42 80 EC 0A BA A0 05 00 00 D1 CA 42 03 15 7D 90
    42 00 4A 81 EA 01 02 00 00 74 02 }
$h17 = { AC C0 C8 01 83 C9 6B 03 DF 8D 1D B1 00 00 00 8D 9E 0B 00 00 00 B9
    86 00 00 00 0F 95 C3 03 D1 2C 96 0B 15 BC 55 40 00 85 DA 74 03 89 4D
    FC 8D 0D 0E 00 00 00 81 EA 02 00 00 00 03 D7 33 D1 49 32 C4 8D 15 0F
    00 00 00 81 EB 02 00 00 00 2B 1D FD 5E 40 00 8B D3 81 EA 01 00 00 00
    8D 88 01 00 00 00 34 8F 81 EB 01 00 00 00 4A 81 C2 02 00 00 00 89 55
    FC 81 EA 02 00 00 00 81 C1 01 00 00 00 1B DB 8B DE 2B C8 C0 C0 03 2B
    DA 66 21 D3 83 CA 27 03 D7 81 EB 01 00 00 00 8D 15 D7 00 00 00 89 55
    FC 43 2C AE 8D 15 06 00 00 00 81 EB 02 00 00 00 66 09 C1 BB 26 00 00
    00 2B D3 81 E9 01 00 00 00 C7 45 FC 2B 00 00 00 C0 C0 01 81 C2 02 00
    00 00 89 7D FC 2B D3 8B D1 8D 1D 06 00 00 00 03 DF C0 C0 02 83 C9 02
    1B CA 81 C2 01 00 00 00 BB 9F 00 00 00 BA FE 00 00 00 83 C9 41 32 C4
    4B 8D 8B 6E 00 00 00 C7 45 FC 0F 00 00 00 03 D8 8D 0D 92 00 00 00 83
    C9 2D 4A 2A C4 81 C1 02 00 00 00 33 D0 81 E9 01 00 00 00 1B DA 81 C2
    02 00 00 00 2B D0 41 2A C4 BB 7A 00 00 00 66 81 FE E6 9A 75 07 }
$h18 = { AC 02 C4 8D 99 78 00 00 00 66 09 FA 83 CB 16 3B 1D 8D D1 48 00 03
    CE 03 D1 02 C4 BA 93 00 00 00 03 D8 81 EA 01 00 00 00 03 D6 83 C9 58
    81 C3 02 00 00 00 8D 0D CE 00 00 00 85 D7 74 01 42 C0 C0 03 BB 7A 00
    00 00 3B 0D F2 84 48 00 81 E9 01 00 00 00 83 FF 76 7A 06 81 EB 02 00
    00 00 89 55 F8 C0 C0 03 03 D3 8D 9F 8C 00 00 00 8B DA BA 63 00 00 00
    81 C2 02 00 00 00 43 32 C4 81 EB 02 00 00 00 81 C3 02 00 00 00 0B 0D
    D1 81 48 00 03 CE 0B 1D 3C CD 4A 00 2B DE B9 D7 00 00 00 8B DA C0 C0
    03 33 DD BA A4 00 00 00 C7 45 9C 05 00 00 00 81 C3 01 00 00 00 8D 88
    0B 00 00 00 41 8D 15 DC 00 00 00 33 DF 99 04 93 81 C1 02 00 00 00 89
    45 C8 81 E2 7E 00 00 00 B9 EC 00 00 00 89 55 A8 03 CE C0 C8 01 81 EB
    01 00 00 00 81 E9 02 00 00 00 87 D2 81 EB 01 00 00 00 3B 15 5C 25 49
    00 81 C3 01 00 00 00 2C 93 0F AD CA 81 C2 01 00 00 00 B9 C8 00 00 00
    81 E9 02 00 00 00 42 1B CB C0 C8 03 C7 45 C0 4B 00 00 00 8B D6 8D 91
    F0 00 00 00 8D 0D 15 00 00 00 83 C9 48 81 EB 01 00 00 00 49 C0 C8 03
    03 D3 C7 45 9C 0A 00 00 00 8D 1D 33 00 00 00 81 EA 01 00 00 00 03 CE
    8D 15 DB 00 00 00 33 15 9D C1 49 00 33 DB 2C 95 83 FB 20 7C 0C }
$h19 = { AC 8D 0D 00 00 00 00 33 DA 3A D6 71 02 51 5B 87 D3 8D 0D 00 00 00
    00 81 E1 00 00 00 00 03 D6 8D 1D 00 00 00 00 3B E1 71 05 3A DA 52 5B
    49 33 CD 8D 1D 01 00 00 00 57 59 BA 00 00 00 00 85 F8 74 04 3C C2 3B
    D4 33 D7 57 5B 32 C4 B9 00 00 00 00 85 CF 74 05 1B DE 3B C4 43 21 FB
```

```
    53 5A 8D 15 01 00 00 00 B9 00 00 00 00 8D 0D 00 00 00 00 87 D3 8D 15
    01 00 00 00 C1 E1 16 8D 1D 00 00 00 00 2B CF BA 01 00 00 00 BA 01 00
    00 00 3B D6 71 05 2B DD 21 EB 43 0F B7 CB 8D 15 00 00 00 00 BB 01 00
    00 00 4B 04 8F B9 01 00 00 00 8D 15 01 00 00 00 8D 0D 00 00 00 00 21
    FA 53 59 B9 00 00 00 00 B9 00 00 00 00 B9 01 00 00 00 2B DF 8D 0D 00
    00 00 00 8D 15 01 00 00 00 42 04 89 21 EB BA 01 00 00 00 85 F2 74 02 }
$h20 = { AC 2A C4 2B 1D 42 01 43 00 03 D0 81 E9 01 00 00 00 81 C2 02 00 00
    00 1B D1 2B D6 49 34 B4 66 B9 3C C7 81 E9 01 00 00 00 81 C3 01 00 00
    00 0B 1D 92 0C 43 00 8D 0D 91 00 00 00 03 CE 42 43 2C 6E 8D 9F 0A 00
    00 00 2B CB 03 D7 8D 1D 5E 00 00 00 03 D0 33 0D 03 04 43 00 03 C8 4B
    8D 8B FD 00 00 00 41 03 CB 43 C0 C8 01 03 D7 C7 45 C8 5F 00 00 00 BB
    2E 00 00 00 33 15 F2 0A 43 00 B9 36 00 00 00 03 CF BA CC 00 00 00 89
    4D E8 41 2C 1A BB 0C 00 00 00 F7 DB B9 BC 00 00 00 8D 1D 05 00 00 00
    8D 91 68 00 00 00 C7 45 E8 24 00 00 00 03 D7 C0 C0 01 81 C1 01 00 00
    00 2B 0D 5B 18 43 00 3B 0D F9 05 43 00 89 55 DC 2A C4 03 DF 81 C1 01
    00 00 00 B9 6A 00 00 00 8B DE 2B D6 8D 8F 0C 00 00 00 2A C4 8D 1D 0D
    00 00 00 8D 15 06 00 00 00 2B CE 33 0D BD 27 43 00 41 2B D6 4A 32 C4
    81 C1 01 00 00 00 66 21 F3 42 8B C8 C7 45 CC 33 00 00 00 8D 0D F2 00
    00 00 8B D6 32 C4 0B 15 0B 0B 43 00 33 15 E1 10 43 00 8D 0D 42 00 00
    00 8B DF 4B C0 C0 03 33 DF 03 DA 83 C9 0F C7 45 B0 4E 00 00 00 81 E9
    02 00 00 00 8B DF 81 C3 02 00 00 00 34 61 8D 1D 51 00 00 00 49 03 C8
    BB 2F 00 00 00 8D 15 09 00 00 00 81 C3 02 00 00 00 83 CA 7A 34 41 B9
    8D 00 00 00 89 7D C4 83 CA 1B 81 E9 01 00 00 00 03 D0 8D 15 03 00 00
    00 2B DA 42 04 C4 03 DE 03 D9 81 EB 01 00 00 00 81 EA 02 00 00 00 83
    FA C8 72 05 }
$h21 = { AC C0 C0 03 33 CE 2B DA 8D 1D 08 00 00 00 3B 0D A0 E0 41 00 03 D7
    8D 9B D3 00 00 00 81 E3 9E 00 00 00 03 CF 34 22 8D 1D CF 00 00 00 8B
    D1 81 E9 01 00 00 00 81 EB 01 00 00 00 66 21 FA 8B DE 8B CA 99 2C BD
    8B CF 8D 93 06 00 00 00 03 D8 8D 15 0E 00 00 00 03 D8 89 7D F8 03 D7
    3B 1D A9 CF 41 00 C0 C0 02 03 C8 BB 5F 00 00 00 89 7D FC BA 11 00 00
    00 81 C2 02 00 00 00 33 DE C0 C0 01 81 E9 02 00 00 00 2B DF 8D 92 1D
    00 00 00 BA 28 00 00 00 03 DA 41 34 19 8D 1D 16 00 00 00 8B D6 33 D0
    B9 8E 00 00 00 03 CA 81 C2 02 00 00 00 03 DA 33 DA 66 B9 71 C0 89 75
    F8 34 99 0F B6 D9 8B D6 8D 1D 06 00 00 00 B9 98 00 00 00 03 D8 03 D6
    41 32 C4 89 4D FC 33 15 C9 E0 41 00 8D 1D 0A 00 00 00 81 EA 02 00 00
    00 8D 15 04 00 00 00 33 D3 03 DA 42 02 C4 8D 15 81 00 00 00 BB E3 00
    00 00 C7 45 F8 5E 00 00 00 8D 99 CF 00 00 00 81 C3 02 00 00 00 BA 70
    00 00 00 42 C0 C8 02 0B 0D 38 E6 41 00 8D 15 F8 00 00 00 81 EA 01 00
    00 00 F7 DA 03 D9 BB 79 00 00 00 C0 C8 01 0B 1D 9F E5 41 00 81 E9 02
    00 00 00 81 C2 02 00 00 00 2B DA 41 C0 C0 03 0F 9C C2 89 4D FC BB FA
    00 00 00 89 75 FC 8D 15 1F 00 00 00 0B 1D FC DC 41 00 03 C8 2B CB C0
    C4 01 33 C9 D1 E1 C1 C9 02 49 29 0D 39 C1 41 00 8B D1 D1 CA 81 EA 1B
    09 00 00 81 C2 22 05 00 00 2B D0 01 15 FE C7 41 00 33 C9 C1 C1 02 C1
    E1 04 81 E9 F7 0D 00 00 C1 E9 02 41 29 0D F6 C1 41 00 81 C1 79 05 00
    00 78 06 }
$h22 = { AC 34 F0 03 DE 8D 9F 8B 00 00 00 81 E9 01 00 00 00 81 EB 01 00 00
    00 BA 02 00 00 00 49 34 FF BB D8 00 00 00 0F CA B9 00 00 00 00 8D 1D
    03 00 00 00 BA 85 00 00 00 B9 48 00 00 00 49 89 55 FC 49 02 C4 B9 42
    00 00 00 87 CB 8B C8 87 C9 2B CF 81 C2 01 00 00 00 83 F8 4F 76 0A 8D
    0D 06 00 00 00 83 CA 3E 4A 89 55 FC 32 C4 81 E9 01 00 00 00 BB AC 00
    00 00 8D 88 0C 00 00 00 81 E9 01 00 00 00 89 7D FC 02 C4 81 E9 01 00
    00 00 8B DF C7 45 FC 24 00 00 00 89 45 FC BA B4 00 00 00 8D 9E 0E 00
    00 00 03 D3 8D 1D 70 00 00 00 34 A2 81 EB 02 00 00 00 3B 0D 2F 1B 4D
    00 C7 45 FC 2A 00 00 00 81 C3 01 00 00 00 8D 9B 7A 00 00 00 2B C8 42
```

```
    34 D0 81 EB 01 00 00 00 03 D7 2B CF 2B 15 8F DB 4B 00 03 CA 8D 0D 0C
    00 00 00 2A C4 81 C3 02 00 00 00 81 C3 01 00 00 00 8D 9A 1B 00 00 00
    03 D3 43 2C D4 33 1D CF C7 4A 00 81 C2 01 00 00 00 C7 45 FC 4D 00 00
    00 33 0D A3 33 4E 00 83 C9 4C 43 2A C4 8D 9E 09 00 00 00 81 EA 01 00
    00 00 8B CE 33 15 77 A1 49 00 8D 91 61 00 00 00 66 81 FB B7 8D 7F 03 }
$h23 = { AC 34 10 81 F8 2B 85 68 08 7E 09 8D 0D B2 00 00 00 8B CB 41 8D 89
    08 00 00 00 03 DF 49 04 38 BB 7A 00 00 00 8D 8E 97 00 00 00 2B 15 92
    B7 40 00 2B CA 8D 15 F4 00 00 00 8B CE 34 0C 83 FE 4C 74 04 8B CA 33
    CE 2B CF 8D 15 05 00 00 00 8D 98 A1 00 00 00 33 CD 8B D7 41 2C 32 89
    4D E8 03 D6 F7 D1 85 D3 75 02 2B D7 03 CB BB 5D 00 00 00 83 CA 0A 8B
    D1 02 C4 03 CA 0B 0D E4 B3 40 00 81 C2 02 00 00 00 B9 4F 00 00 00 4A
    2B DE 49 02 C4 C7 45 E4 44 00 00 00 8D 1D C8 00 00 00 03 D3 C7 45 C4
    39 00 00 00 89 75 D0 1B D7 04 CF 81 C1 02 00 00 00 BB CD 00 00 00 B9
    A4 00 00 00 B9 AA 00 00 00 03 CF 8D 15 06 00 00 00 BA F4 00 00 00 89
    75 D0 C0 C8 02 F7 D2 8B D1 B9 97 00 00 00 83 CA 51 03 D8 89 4D B8 3B
    1D A7 B0 40 00 3B 0D F1 BA 40 00 8B CE 4A 32 C4 81 EA 02 00 00 00 8B
    D3 0B 1D 67 B2 40 00 03 D9 83 C9 15 89 5D BC 89 45 B0 49 C0 C0 03 43
    8B CA 2B 1D 38 B5 40 00 2B D0 0B 15 04 B6 40 00 81 EB 01 00 00 00 B9
    C9 00 00 00 2C BD 8D 0D 70 00 00 00 83 C9 11 C7 45 F8 27 00 00 00 81
    C3 01 00 00 00 8D 93 01 00 00 00 0F 93 C1 C0 C8 03 81 C1 01 00 00 00
    81 E9 02 00 00 00 C7 45 E4 2C 00 00 00 81 E9 02 00 00 00 03 CA 03 DF
    49 04 93 89 7D F4 83 C9 3A 03 CE 03 DE 03 D1 8D 8A 38 00 00 00 3B 15
    6C B9 40 00 2B D6 03 D6 2C 0E 2B DF 2B CB BB 5A 00 00 00 81 C3 02 00
    00 00 66 81 FF AC 89 74 0C }
$h24 = { AC C0 C0 03 81 E9 02 00 00 00 8D 1D F4 00 00 00 81 E9 02 00 00 00
    8D 89 6A 00 00 00 8D 0D AA 00 00 00 8B D9 03 DA 4A 2C 32 C7 45 C8 2F
    00 00 00 81 EA 01 00 00 00 89 75 EC 89 7D 90 8D 1D 40 00 00 00 34 02
    81 C3 01 00 00 00 3B 0D E5 1A 43 00 8D 8B 74 00 00 00 8D 1D 08 00 00
    00 81 EA 02 00 00 00 F7 D9 4B 02 C4 81 E9 02 00 00 00 81 F9 6D A2 D4
    00 76 0C 8D 93 06 00 00 00 8B C8 8B CF 03 D9 33 DA 33 D7 C0 C8 03 4B
    BA EA 00 00 00 21 DB 0F B6 DA 83 CB 47 C7 45 98 23 00 00 00 8D 1D F6
    00 00 00 C7 45 90 6F 00 00 00 83 CA 0E 2C 1F 81 C2 02 00 00 00 2B 0D
    22 04 43 00 8D 93 05 00 00 00 81 E9 02 00 00 00 43 2C E2 BB 73 00 00
    00 81 C2 02 00 00 00 81 C1 02 00 00 00 8D 1D 68 00 00 00 81 C3 02 00
    00 00 2A C4 89 55 BC 8D 0D 04 00 00 00 81 EB 02 00 00 00 85 DE 75 02 }
$h25 = { AC 32 C4 8D 0D 0C 00 00 00 8D 9F 0D 00 00 00 89 45 E0 2B CF 4A 8D
    8F C6 00 00 00 81 C3 02 00 00 00 89 55 F4 02 C4 BA B2 00 00 00 81 C3
    02 00 00 00 C7 45 E0 5D 00 00 00 81 E9 01 00 00 00 81 C2 02 00 00 00
    89 7D E0 43 34 86 8D 8A 68 00 00 00 8D 15 1A 00 00 00 B9 9F 00 00 00
    C7 45 F0 31 00 00 00 2C D5 66 81 F9 9B C0 7C 0D 8B CB 81 C3 02 00 00
    00 8B C8 66 09 CB 81 E9 02 00 00 00 0F BC D3 33 DE 04 8F 0F CA C7 45
    EC 12 00 00 00 8D 99 04 00 00 00 2B DE B9 A9 00 00 00 83 CB 5A 43 04
    36 81 EB 01 00 00 00 BA E5 00 00 00 8B DA 8B C8 81 EB 02 00 00 00 03
    D7 8D 15 0B 00 00 00 C0 C8 03 8D 8F B5 00 00 00 8D 15 64 00 00 00 BB
    F7 00 00 00 8D 98 BD 00 00 00 89 7D DC 03 D0 04 1A 2B 15 8D 2F 42 00
    03 D9 3B 1D 82 30 42 00 C7 45 F4 49 00 00 00 81 C1 01 00 00 00 B9 D6
    00 00 00 2B D9 1B D9 32 C4 83 CB 2E 8D 1D 0A 00 00 00 BB EE 00 00 00
    81 E9 02 00 00 00 8B D9 99 04 DB 03 D7 33 0D 51 35 42 00 83 C9 32 66
    21 C2 BB 29 00 00 00 8B DA 34 9F 8D 97 C8 00 00 00 8D 15 00 00 00 00
    83 CB 18 49 8D 15 09 00 00 00 8D 15 87 00 00 00 8B CB 34 9D 0B 1D B6
    2C 42 00 8D 0D 0E 00 00 00 2B D8 03 D1 85 D0 74 01 }
$h26 = { AC 8D 15 00 00 00 00 C1 E1 09 C1 EA 08 C1 E1 09 8D 1D 01 00 00 00
    8D 0D 01 00 00 00 3A E3 7E 04 33 C8 53 5A 81 E3 00 00 00 00 BA 01 00
    00 00 8D 15 01 00 00 00 BA 00 00 00 00 BA 01 00 00 00 8D 0D 00 00 00
```

```
        00 8D 0D 00 00 00 00 2C 8D 21 F9 3C F4 78 04 C1 E9 0B 41 8D 0D 01 00
        00 00 87 D3 81 E2 00 00 00 00 C1 E3 14 BA 00 00 00 00 81 E2 00 00 00
        00 8D 15 01 00 00 00 03 D0 8D 1D 00 00 00 00 3A E1 7E 02 85 F1 3B DF
        72 01 4A C1 E9 02 87 D9 66 99 49 02 C4 BB 00 00 00 00 8D 0D 01 00 00
        00 0F B7 C9 BB 01 00 00 00 81 E1 01 00 00 00 33 D0 81 E3 01 00 00 00
        87 D1 3C F8 73 03 }
    $h27 = { AC 2C 01 03 CB 2B 0D 08 67 40 00 8D 8B 97 00 00 00 8D 9E 03 00 00
        00 0B 0D F0 66 40 00 81 C3 02 00 00 00 42 2B CB 02 C4 81 C1 01 00 00
        00 C7 45 D8 45 00 00 00 81 E9 02 00 00 00 33 15 8D 6A 40 00 BB 99 00
        00 00 02 C4 66 BB A4 3A C7 45 E8 7F 00 00 00 8D 15 0A 00 00 00 B9 05
        00 00 00 43 C0 C0 02 81 F8 4C 93 76 E7 7C 06 89 4D BC 03 D8 49 2B D6
        83 CB 66 03 D6 2B D7 F7 D2 0B 15 29 61 40 00 83 CA 12 43 02 C4 2B 15
        12 6B 40 00 BA 75 00 00 00 03 D7 8D 90 0F 00 00 00 03 D0 81 C2 02 00
        00 00 C7 45 B4 44 00 00 00 8B D1 42 34 A3 66 81 F8 D5 72 7C 0D }
    $h28 = { AC 02 C4 33 DD 03 D8 81 EB 02 00 00 00 8D 99 B0 00 00 00 66 B9 37
        01 BB B5 00 00 00 8D 1D 6A 00 00 00 49 02 C4 03 CB 33 DD 2B DF 8D 8A
        02 00 00 00 C7 45 D0 7B 00 00 00 8B DF BA 23 00 00 00 4B C0 C0 02 83
        CB 1A 2B 0D BF EE 40 00 2B DA 81 EA 01 00 00 00 2B 15 1E EC 40 00 49
        41 2C 94 8B D8 0B 0D 77 E4 40 00 8D 9A 03 00 00 00 8D 15 0F 00 00 00
        43 02 C4 89 45 A4 81 C3 02 00 00 00 C7 45 D4 72 00 00 00 81 EB 01 00
        00 00 8B D8 33 CF 02 C4 89 5D C0 3B 15 BF EC 40 00 81 C2 02 00 00 00
        BA 1B 00 00 00 4A 32 C4 C7 45 A4 2B 00 00 00 41 81 C3 02 00 00 00 83
        CB 44 66 09 D3 8D 15 0E 00 00 00 C7 45 CC 30 00 00 00 F7 DA 43 04 D8
        BA A6 00 00 00 81 C2 02 00 00 00 81 E9 01 00 00 00 33 DA 89 7D BC 41
        2C 56 81 E9 01 00 00 00 83 CA 2B 03 DA 8D 93 0F 00 00 00 81 C2 01 00
        00 00 87 D1 C0 C4 01 BB 9A 03 00 00 81 EB BC 08 00 00 C1 EB 06 03 1D
        88 E1 40 00 4B 8B D3 C1 CA 02 81 EA F1 0C 00 00 72 02 }
    $h29 = { AC C0 C8 02 66 09 EB 8D 90 7C 00 00 00 8B DA 8D 90 20 00 00 00 8B
        D0 81 EB 02 00 00 00 83 CB 3F 4B 04 1E 83 CB 09 8D 89 BB 00 00 00 C7
        45 D8 06 00 00 00 66 09 D3 89 7D C4 33 D7 89 45 D0 8B D8 41 02 C4 66
        81 F9 EE 0E 73 06 89 7D E8 8B D0 49 BA 80 00 00 00 0F BC DA 81 E9 02
        00 00 00 0F BF DA 8B CB 02 C4 33 C8 83 CB 1D 85 DF 74 01 }
    $h30 = { AC C0 C0 03 81 EA 01 00 00 00 33 0D FF 76 43 00 BB 9F 00 00 00 8B
        D3 89 45 F8 8D 9B 0D 00 00 00 1B D9 BB 91 00 00 00 8B CB 2A C4 8D 0D
        47 00 00 00 0B 1D 4F 8E 43 00 83 C9 7B 4B BA 20 00 00 00 C7 45 F4 69
        00 00 00 BB 79 00 00 00 99 34 52 8B CA 03 CF 8D 90 40 00 00 00 2B DE
        BB 8D 00 00 00 2B D1 33 CB 04 3F 21 F2 8B D6 66 81 FF 1F 10 7D 0D 2B
        D0 BA D1 00 00 00 21 C3 03 DA 03 D6 66 81 FF 8E 50 7A 04 89 45 FC 4B
        4B 34 EA 81 C2 01 00 00 00 F7 D2 81 EA 01 00 00 00 81 E9 02 00 00 00
        3B 0D 93 9B 43 00 2B CA 33 D9 33 D6 2C 4B 83 F8 4B 76 0D }
condition:
    pe.is_pe and
    any of them
}
```

Príloha Q

Cryptic SimdaPacked varianta B

```
import "pe"

rule simda_packed_b_known_sequences
{
    meta:
        author = "David Duchon, Gen"
        description = "detection based on known static sequences"
        strain = "SimdaPacked"
        type = "cryptic"
        severity = "malware"
    strings:
        $h00 = /(\xA8\xA9\xAA[\x00-\xFF]){100}/
    condition:
        pe.is_pe and
        for any entry in pe.resources :
        (
            (entry.length == 0x10a8) and
            (entry.type == pe.RESOURCE_TYPE_ICON) and
            ($h00 in (entry.offset .. entry.offset + entry.length))
        )
}
```

Príloha R

Obsah na cloudovom úložisku

- `malware.zip` – Archív chránený heslom `infected` obsahujúci vzorok z každej analyzovanej a detekovanej Cryptic rodiny
- `yara_rules` – Priečinok obsahujúci každé vytvorené YARA pravidlo
- `yara` – Spustiteľný ELF súbor obsahujúci YARA s modulmi `dotnet`, `pe` a `math`
- `README` – Text popisujúci spôsob otestovania YARA pravidiel