VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Samuel Sidor

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VYLEPŠENÝ SANDBOXING PRO POKROČILÉ KMENY MALWARU
ENHANCED SANDBOXING FOR ADVANCED MALWARE FAMILIES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Samuel Sidor

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Jan Hajný, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Samuel Sidor ***ID:*** 195167
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Vylepšený sandboxing pro pokročilé kmeny malwaru

POKYNY PRO VYPRACOVÁNÍ:

Práce bude zaměřena na analýzu vzorků pokročilých kmenů malwaru a na vylepšení jejich dynamické analýzy v sandboxu. Nejprve se bude jednat o nastudování existujícího sandboxing řešení společnosti Avast Software. Dále pak o hloubkovou analýzu vybraných kmenů malwaru (pomocí reverzního inženýrství) s cílem odhalit v nich obsažené techniky pro detekování běhu v sandboxu a související akce. Následně bude mít student za úkol navrhnout signatury, které tyto techniky budou automaticky odhalovat. Pokud to bude v konkrétních případech technicky možné, student navrhne a částečně i realizuje vylepšení současného sandboxu, který pak bude proti těmto technikám rezistentní. Výsledek musí být otestován v praxi.

DOPORUČENÁ LITERATURA:

[1] Richard Mička. Automatická detekce použité kryptografie v kódu. Brno, 2018, 58 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[2] Michael Sikorski and Andrew Honig. Practical Malware Analysis The Hands-On Guide to Dissecting Malicious Software, 2012, ISBN-13: 9781593272906.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant: Jakub Křoustek (Avast Software s.r.o.)

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práca pojednáva o témach so zameraním na reverzné inžinierstvo využité na malvéri. Čitateľ bude oboznámený o teoretickom popise statickej a dynamickej analýzy a neskôr budú tieto typy analýz využité pri analýze 5 malvérových rodín so zameraním na detekciu v nich využitých anti-sandbox techník. Následne je navrhnutý teoretický návrh rozšírenia sandboxu o funkcionalitu na detekciu alebo úplnu elimináciu týchto anti-sandbox techník s cieľom následného otestovania a nasadenia zmien na hlavný malvérový sandbox firmy Avast Software. Z nasadenia a otestovania na reálnych vzorkách čitateľ zistí ako boli jednotlivé vylepšenia sandboxu efektívne a aké benefity priniesli.

KĽÚČOVÉ SLOVÁ

Cuckoo, sandbox, malvér, reverzné inžinierstvo, anti-sandbox, Cuckoo signatúry, uniknutie zo sandboxu

ABSTRACT

This Master's thesis describes reverse engineering with focus on malware analysis. Reader will be informed about theoretical description of static and dynamic analysis. These techniques are later used on analysis of 5 malware families with focus on detection of used anti-sandbox techniques. After that new theoretical improvements are proposed with detection of anti-sandbox techniques or fully avoiding such anti-sandbox evasion techniques. Finally these changes are implemented on main sandbox of Avast Software from which reader can see how effective these improvements are.

KEYWORDS

Cuckoo, sandbox, malware, reverse engineering, anti-sandbox, Cuckoo signatures, sandbox evasion

SIDOR, Samuel. *VYLEPŠENÝ SANDBOXING PRO POKROČILÉ KMENY MALWARU*. Brno, 2021, 63 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. Jan Hajný, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „VYLEPŠENÝ SANDBOXING PRO POKROČILÉ KMENY MALWARU“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno . . . .

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi doc. Ing. Janu Hajnému, Ph.D. za užitočné pripomienky, ochotu a usmernenie pri písaní práce. Taktiež ďakujem Ing. Jakubovi Křoustokovi, Ph.D. zo spoločnostni Avast za odborné konzultácie, trpezlivosť, podnetné návrhy a cenné rady. Nakoniec by som ešte chcel poďakovať svojej priateľke, ktorá ma vytrhávala z dennej depresie pri písaní tejto práce a rovnako tak aj spoločnosti Google za videá s mačičkami na službe Youtube, ktoré boli ako balzam na nervy. V poslednom rade ďakujem hlavne sebe, lebo bezomňa by som napísanie tejto práce nedokázal. Ďakujem si za motiváciu a neustále povzbudzovanie aj v tých najťažších chvíľach, ktorých nebolo málo.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

So stúpajúcim trendom výskytu škodlivého kódu v každodennom živote bežného používateľa je nevyhnutné sa proti týmto hrozbam brániť. Keďže množstvo malvéru presahuje kapacitu manuálneho vyhodnocovania analytikov, je nutnosť použitia automatizovanej analýzy. Avšak vo väčšine prípadov nie je automatizovaná statická analýza postačujúca na detekovanie väčšiny malvérov, nakoľko ich tvorcovia využívajú rôzne druhy obfuskácie. Práve z toho dôvodu sa využívajú sandboxy, ktoré sú schopné odhaliť behaviorálne prvky malvéru, nakoľko sa v nich daný malvér spustí. Tvorcovia malvéru o tomto spôsobe analýzy vedia a snažia sa, aby práve ich malvér odhalil, že je spustený v sandboxe a tým pozmenil svoje správanie, alebo aby vykonal akcie, ktoré zamedzia tomu, aby bol analyzovaný. Obzvlášť znalosť spôsobu analýzy tvorcov malvéru je kľúčový prvok vzniku tejto práce a z toho dôvodu je hlavnou motiváciou vylepšiť súčastnú implementáciu hlavného Windows sandboxu firmy Avast Software a tým pomôcť zvýšiť detekciu komplexných malvérov, ktoré obsahujú tieto techniky. Z výsledkov práce budú mať hlavný úžitok najmä samotní používatelia, ktorí budú v konečnom dôsledku lepšie chránení. Výsledky by mali v prvom rade priniesť prehľad najčastejšie používaných techník na obchádzanie sandboxov a následne štatistiky vyhotovené na základne novonavrhnutých signatúr, ktoré odhalia množstvo malvéru, ktoré tieto techniky využíva. Tým sa zlepší detekcia sandboxu a je to prvý krok k tomu, aby sa voči týmto technikám vykonali náležité opatrenia.

Prvá kapitola čitateľovi ukáže stručný náhľad do problematiky analýzy škodlivého kódu zahŕňajúc najdôležitejšie nástroje a taktiež úvod do sandboxovacích techník. Druhá kapitola sa bude venovať konkrétnej implementácii sandboxu Cuckoo, ktorý sa používa na automatizovanú dynamickú analýzu v spoločnosti Avast Software. Kapitola číslo tri popíše techniky, ktoré sa používajú na detekovanie analýzy v sandboxe, alebo dokonca jej úplne zabránenie. Po prejdení štvrtej kapitoly získa čitateľ prehľad o postupe analýzy pokročilého malvéru. Detekčnými signatúrami, ich tvorbou, spôsobom detekcie, overovaním a rovnako tak aj zistením problémov súčastnej implementácie Cuckoo sandboxu sa zaoberá kapitola číslo päť. V poslednej kapitole obsahujúcej experimentálne výsledky bude ukázané akú účinnosť mali jednotlivé vylepšenia sandboxu.

1 Spôsoby analýzy škodlivého kódu

S tým aký je škodlivý kód (malvér) rôznorodý a komplikovaný, existuje viacero techník na jeho analýzu. Tieto techniky sa delia do dvoch kategórií a to na statickú a dynamickú analýzu. Každá z týchto analýz je vhodná na získanie iných informácií za účelom lepšieho pochopenia vnútornej funkcionality malvéru a jeho štruktúry [1]. V rámci funkcionality je analytik s dostatočným množstvom času schopný presne určiť aké akcie malvér vykonáva a za akých okolností sú tieto akcie realizované.

1.1 Statická analýza

Statická analýza je spôsob anylyzovania malvéru bez nutnosti jeho spustenia. Je to väčšinou prvý krok, ktorý analytik vykonáva na začiatku každej analýzy, aby pochopil hlavné znalosti o danej vzorke. Medzi tieto znalosti môže patriť napríklad zistenie využitého prekladača, typ súboru, využívané Windows API funkcie. Avšak najpodstatnejší je práve preklad binárneho súboru do symbolických inštrukcií (disassemblovanie), alebo dokonca získanie takmer pôvodného zdrojového kódu pomocou dekompilátoru.

Disassembler

Čitateľ bol už oboznámený s pojmom disassemblovanie. Na vykonanie tohto procesu sa najčastejšie využíva nástroj tzv. disassembler, ktorý získa ako vstup binárny súbor a na výstupe vráti v textovej podobe assemblerovské inštukcie celého programu alebo jeho zvolenej časti. Asi najznámejším disassemblerom využívaným pri odbornej analýze malvéru je IDA PRO[1] (Interactive Disassembler Professional) [1].

Dekompilátor

Nie všetky programovacie jazyky sa počas prvotného prekladu (kompilácie) prekladajú do natívneho kódu [2]. Jazyky ako .NET alebo Java využívajú koncept zvaný JIT (Just in time) preklad, ktorý spočíva v tom, že sa pôvodný zdrojový kód preloží do tzv. p-code (portable kódu,) taktiež nazývaného ako intermediate language (IL) alebo bytecode. Tento IL sa následne až za behu prekladá do inštrukcií špecifických pre daný procesor. A práve z tejto znalosti čerpá dekompilátor. Je to program, ktorý umožňuje spätne preložiť spomínaný IL do takmer pôvodného zdrojového kódu, keďže IL obsahuje väčšie množstvo informácií a štruktúr ako samotný assembler.

[1] https://www.hex-rays.com/

1.2 Dynamická analýza

Na rozdiel od statickej analýzy sa tá dynamická zaoberá samotným správaním daného malvérového vzorku [3]. Práve z toho dôvodu, že je nevyhnutné malvér spustiť, je potreba zabezpečiť, aby nepoškodil zariadenie na ktorom je táto analýza vykonávaná. Práve preto sa využívajú rôzne techniky virtualizácie v ktorých sa tento druh analýzy vykonáva. Následne môže analytik s pomocou rôznych nástrojov sledovať reálne správenie malvéru na danom zariadení.

Debugger

Je najčastejšie využívaný nástroj na sledovanie správania malvéru resp. jeho ladenie. Dôvod využitia je ten, že aj keď analytik teoreticky dokáže zistiť celú logiku programu statickou analýzou, tak je tento spôsob veľmi zdĺhavý a vyžaduje väčšie množstvo času. Zatiaľ čo pri debuggingu môže sledovať skutočnú funkcionalitu počas behu programu. Túto schopnosť mu zabezpečuje nastavovanie breakpointov (bodov prerušenia), ktorými si môže zastavovať beh programu podľa potreby. Jedná sa o rovnaký proces, aký využívajú softvéroví inžinieri na ladenie počas vývoja programov [2].

Automatizovaná analýza

Tento druh analýzy je najpodstatnejší pre plošné testovanie behaviorálnych vzorov a heuristík na základe ktorých dokážu antivírusové programy sami vyhodnocovať potencionálne riziko spusteného programu.

Kľúčovím bodom je sandbox. Jedná sa o virtualizované prostredie, ktoré je oddelené od prístupu hlavného operačného systému (OS) a jeho prostriedkov z dôvodu možnej nebezpečnej činnosti potencionálne škodlivého programu. Je známych viacero typov sandboxu. Najznámejšie sú sandboxy využívajúce user mode hooking, kernel mode hooking, hypervisor alebo emuláciu. Hookovanie je proces, pri ktorom je prepísané pôvodné volanie funkcie na volanie inej funkcie. Je tak možné presmerovať pôvodné volanie do vlastného kódu, kde je možné narábať so získaným tokom programu. Pomocou hookov je tak možné transparentne pre bežiaci program zbierať dáta o jeho volaniach a parametroch v nich použitých [4].

Procesor v počítači so systémom Windows má dva rôzne režimy: používateľský režim (UM - user mode) a režim jadra (KM - kernel mode). Procesor prepína medzi týmito dvoma režimami v závislosti od toho, aký typ kódu v procesore beží. Aplikácie sa spúšťajú v UM a základné komponenty operačného systému sa spúšťajú v KM. Zatiaľ čo veľa ovládačov pracuje v KM, niektoré ovládače môžu bežať aj režime užívateľskom [5]. Z toho vyplýva, že existujú 2 široko akceptované metódy na prerušovanie funkcií programu (hookovanie) a nimi sú UM hooky a KM hooky. UM je

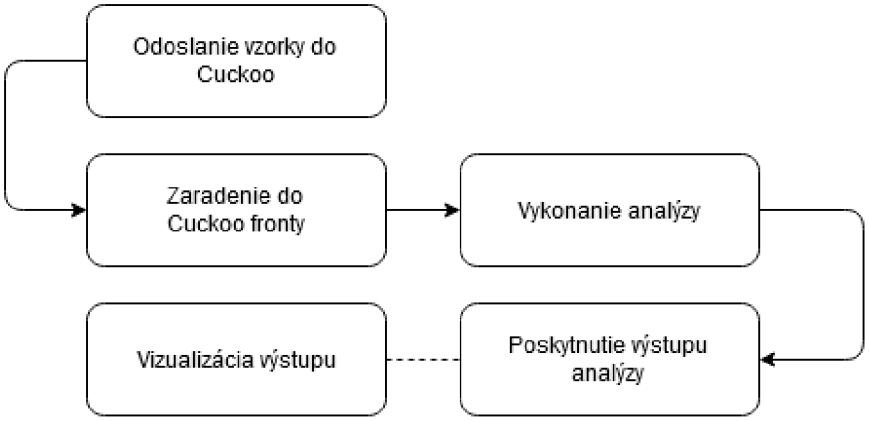
špeciálny mód na spracovanie úloh, ktoré sú viditeľné pre užívateľa, avšak majú menšie oprávnenia ako KM. KM na druhú stranu je značne globálnejšia implementácia, ktorá pracuje priamo s jadrom operačného systému a spracováva príkazy špecifické pre daný systém. Zjednodušene môže byť tento koncept braný ako „front end“ a „back end“ [6].

Ďalším spôsobom sandboxovania je použitie hypervisoru. Hypervizor, taktiež známy ako monitor virtuálneho stroja alebo VMM. Je to softvér, ktorý vytvára a spúšťa virtuálne stroje (VM). Hypervizor dovoľuje jednému hostiteľskému (host) počítaču spúšťať viacero hosťovských (guest) VM pomocou virtuálneho zdieľania ich prostriedkov ako pamäť a procesorový čas. Existujú 2 hlavné typy hypervizorov a nimi sú typ 1 (bare-metal) a typ 2 (hosted). Typ 1 sa správa ako ľahký OS a beží priamo na hardvéri hostiteľa. Zatiaľ čo typ 2 beží ako softvérová vrstva na hostiteľskom OS ako iný počítačový program [7]. Nakoľko sa hypervisor nachádza medzi virtualizovanými zariadeniami a hardvérom, tak má prístup k zdrojom týchto virtuálnych strojov a tým pádom sandbox implementácia tohto typu má viaceré výhody. Najväčšou z nich je že na virtualizovanom operačnom systéme nemusí prebiehať žiadny typ hookovania a tým pádom sa virtualizovaný systém javí ako bežné zariadenia, čo je najžiadanejšia vlastnosť pri sandboxovaní malvéru.

Posledný známy typ používaný pri sandboxovaní je emulácia. Emulácia je zameraná na reprodukciu vonkajšieho správania totožného s emulovaným systémom. Je to v podstate úplná replika iného systému a rovnako je aj binárne kompatibilná so vstupmi a výstupmi emulovaného systému, ale pracuje v inom prostredí ako prostredie pôvodného emulovaného systému. Z toho dôvodu sa dostatočne dobrý emulátor môže stať úplnou náhradou za pôvodný systém [8]. Avšak emulácia so sebou prináša viacero nevýhod. V prvom rade praktické nasadenia plnohodnotnej emulácie je drahá záležitosť, nakoľko je vyžadovaná neustála podpora a vylepšenie nutné na emuláciu nových alebo už existujúcich API funkcií. Ďalej sa rovnako vyžaduje zvýšená pozornosť pri technikách využívaných na obchádzanie sandboxov z toho dôvodu, že emulátor má od emulovaného systému odlišnú implementáciu a tento rozdiel dokáže malvér veľakrát detekovať [9].

2 Cuckoo Sandbox

Cuckoo je open-source systém písaný v jazyku Python 2 a slúži na automatizovanú analýzu malvéru. Môže na ňom bežať virtuálne prostredie jedného z hlavných operačných systémov ako sú Windows, Linux, macOS alebo Android. Proces fungovania spočíva v automatickom spustení a analýzy súborov, zberu výsledkov z dokončených analýz a ich ďalšie spracovanie, ktoré priblíži aké akcie malvér vykonáva zatiaľ čo beží v tomto izolovanom operačnom systéme. Tento sandbox je taktiež veľmi modulárny, čo umožňuje jednoduché rozšírenie existujúcej funkcionality [10].



Obr. 2.1: Ukážka náväznosti jednotlivých blokov Cuckoo spracovania.

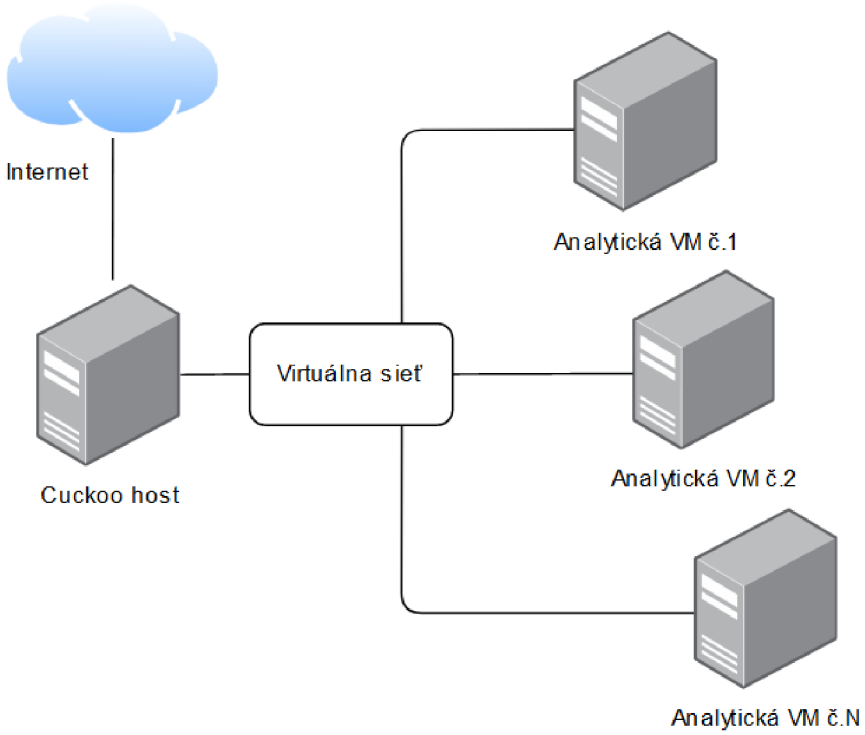
2.1 História

Projekt Cuckoo Sandbox[1] začal ako projekt Google Summer of Code v roku 2010 pod vedením vývojára Claudio „nex“ Guarnieri, ktorý je doteraz líder a hlavný vývojár. V roku 2011 sa rozšíril tím vývojárov a koncom tohto roku bola vydaná stabilná verzia 0.3. V roku 2012 poskytol Cuckoo tím funkčnú online implementáciu na stránke malwr.com zdarma. Po ďalšom rozšírení tímu sa verzia posunula až na 0.5. Nasledujúce roky nasledoval neustály vývoj najmä v rámci funkcionality, praktickej použiteľnosti Cuckoo a taktiež bol rozšírený o analýzu macOS malvéru. V súčasnosti je aktuálna verzia 2.07. Cuckoo projekt má na github viac ako 1 500 kópií hlavného repozitára (forknutí) a veľkú komunitu či už používateľov alebo aj prispievateľov zdrojového kódu. Je licencovaný pod GNU (General Public License) v3.0 [11].

[1] https://github.com/cuckoosandbox

2.2 Architektúra

Cuckoo Sandbox pozostáva z centrálneho softvéru na správu, ktorý zaisťuje vykonávanie a analýzu vzoriek. Každá analýza sa spúšťa na čistom a izolovanom virtuálnom alebo fyzickom stroji (po každej analýze sa vrátia zmeny do pôvodného stavu). Hlavnými súčasťami infraštruktúry Cuckoo sú hostiteľský počítač (softvér na správu) a niekoľko hosťujúcich strojov na analýzu (virtuálne alebo fyzické stroje). Hostiteľský počítač prevádzkuje hlavnú súčasť sandboxu, ktorá riadi celý proces analýzy, zatiaľ čo hosťovské počítače sú izolované prostredia, v ktorých sa vzorky škodlivého softvéru vykonávajú a analyzujú [11]. Nasledujúci obrázok vysvetľuje hlavnú architektúru Cuckoo:



Obr. 2.2: Architektúra Cuckoo Sandboxu [11].

2.3 Funkcionalita

Hlavnou výhodou Cuckoo Sandboxu je fakt, že dokáže analyzovať väčšinu známych súborov ako napríklad: PE (Portable Executable), Microsoft Office dokumenty, PDF (Portable Document Format) súbory, emaily atď. Okrem súborov je schopný aj analýzy

webových stránok. Umožňuje sledovať API volania a správanie spustených súborov a následne tieto údaje prefiltruje a normalizuje do čitateľného formátu. Taktiež dokáže analyzovať sieťové pripojenie a to aj v prípade, že sa používa SSL/TLS šifrovanie. Okrem iného je schopný preposielať všetku komunikáciu zo sieťového rozhrania cez VPN. Jednou z hlavných funkcií je ešte vykonávanie pokročilej analýzy pamäte infikovaného virtualizovaného systému [10].

Cuckoo Avast implementácia nie je postavená na hlavnej vetve Cuckoo sandboxu, ale na vetve, ktorá sa odčlenila a implementovala si množstvo vlastných zmen. Táto vetva sa nazýva Cuckoo-modified[2] a vychádza z Cuckoo verzie 1.2. Oficiálny vývoj tejto vetvy bol ukončený, ale interne v rámci Avastu sa na ňom naďalej pokračuje.

2.4 Popis analýzy

O analýzu programu bežiacom v Cuckoo sa stará skript `analyzer.py`, ktorý na získanie informácií využíva metódu hookovania za pomoci `cuckoomon.dll`. Jedná sa o dynamicky linkovanú knižnicu DLL (na rozdiel o statického linkovania nemusí byť súčasťou spusteného programu), ktorá je injektovaná do analyzovaného programu pri jeho spustení. DLL injektovanie je technika využívaná na spúšťanie kódu v pamäti iného programu, nútiac ho načítať túto knižnicu a následne ju spustiť aj napriek tomu, že táto knižnica nepatrí do pôvodného návrhu programu [12]. Na zasielanie informácií späť do hlavného analyzačného procesu sa používajú Named Pipes (je to jeden zo spôsobov medziprocesovej komunikácie). Po inicializácii injektovanej DLL sa nastavia v bežiacom programe hooky známych Windows API funkcií. Pojem hook predstavuje vloženie funkcionality medzi volanú metódu a kód, ktorý ju volal. Týmto spôsobom je možné sledovať aktivitu týchto volaní, ba dokonca aj meniť poslané parametre, odpoveď Windows API funkcií, ako aj úplne zakázanie komunikácie s volanými API funkciami.

Všetky zaznamenané API volania a ich parametre sa ukladajú do formátu BSON (Binárny JSON) a po skončení analýzy sú spolu s inými zaznamenanými dátami ako napr. časťami alokovaných bufferov, vytvorených súborov a sieťovou komunikáciou poslané hostiteľskému počítaču na spracovanie pre ďalšie použitie.

2.5 Spracovanie a zobrazenie výsledkov analýzy

Po analýze dát je nutné ich patričným spôsobom spracovať. Túto funkciu vykonávajú Cuckoo moduly `Processing` a `Signature`. Prvá časť spracovania prebieha v module `Processing`, ktorý analyzuje priamo dáta z behu programu v binárnej podobe [4].

[2]https://github.com/spender-sandbox/cuckoo-modified

Analyzujú sa prijaté záznamy použitých API volaní, ktoré sa upravia do lepšej strojovo-spracovateľnej podoby. V rámci jazyka Python je to uloženie do dátových štruktúr typu list alebo slovník.

Predchádzajúci krok spracovania bol dôležitý pre modul `Signature`, ktoré tieto dáta ďalej používa. Tento modul pracuje s takzvanými signatúrami, ktoré slúžia na odhalenie istého správania, alebo neobvyklých javov zo spracovaných dát modulu `Processing`.

Po spracovaní všetkých potrebných informácií prichádza na rad ich zobrazenie. O to sa stará webové rozhranie napísané v Python frameworku Django. Táto reprezentácia dát je pre analytikov malvéru veľmi dôležitá, nakoľko sa tú nachádzajú informácie o behu analyzovaného procesu. Medzi tieto informácie patria obzvlášť výsledky signatúr, behaviorálne prvky ako manipulácia so súbormi a registrami, sieťová komunikácia a použité synchronizačné objekty ako napr. mutexy a semafory. Z týchto dát je potom analytik schopný zistiť, čo je hlavnou funkciou analyzovaného programu, alebo je schopný zostaviť detekčné pravidla na jeho detekciu.

3 Techniky vyhnutia sa behu v sandboxe

Jedná sa o široký súbor techník, ktorých hlavným účelom je zabezpečiť, aby program, ktorý má byť analyzovaný, bol schopný odhaliť, že je v sandboxe, alebo aby využil špecifickosť daných sandbox implementácií a vďaka tomu sa správal inak než ako v klasických zariadeniach. Ďalej sa môže jednať o vykonanie akcií, ktoré ťažia z toho, akým spôsobom sa program analyzuje a vykonajú také akcie, aby k jeho úplnej analýze nedošlo (tieto akcie sa vykonajú bez ohľadu na to či je program spustený v sandboxe). Nakoľko definovanie tohto správania je veľmi všeobecné, tak bude presnejšie opísané pri jednotlivých najčastejšie používaných technikách v nasledujúcej podkapitole 3.1.

3.1 Delenie

Interakcia používateľa

V tomto prípade kontroluje malvér či sa v istom časovom intervale pohol kurzor myši, alebo či bola stlačená nejaká klávesa. Jedná sa o techniky, ktoré majú zistiť či bol malvér spustený užívateľom, ktorý so systémom interaguje alebo o strojové automatické spustenie v sandboxe. Môže sa jednať napríklad o kontrolu toho, či bol vykonaný istý počet klikov a až po tejto akcii sa malvér spustí.

Detekcia virtuálneho prostredia

Veľa virtuálnych prostredí (VP) má špecifické artefakty, alebo špecifický spôsob ako je v nich implementovaná istá funkcionalita. Môže sa napríklad jednať o súbory, procesy, predvolené mená a názvy komponentov, alebo aj typy konfigurácií, ktoré tam bývajú nastavené. Jedným zo spôsobov detekcie VP je využitie inštrukcie CPUID na získanie CPU špecifikácie. Táto špecifikácia sa vo VP líši od klasických systémov.

Detekcia sandboxu

Rovnako ako virtuálne prostredia využité v sandboxoch, tak aj funkcionalita využívaná na analyzovanie behu programu zvyčajne obsahuje veľké množstvo artefaktov špecifických len pre daný sandbox a túto znalosť využívajú tvorcovia malvéru taktiež vo svoj prospech. Medzi artefakty, ktoré môže malvér detekovať patria súbory nevyhnutné na analýzu malvéru. V rámci Cuckoo sandboxu sa môže jednať napríklad o `cuckoomon.dll`.

Kontrola operačného systému

Dalo by sa povedať, že kontrola samotného OS by sa dala radiť tiež pod detekciu VP, avšak keďže môže byť malvér spustený aj na hlavnom hardvéri a nie len na virtuálnom stroji, tak bolo nutné túto časť osamostatniť. V tomto prípade hľadá malvér isté stopy v operačnom systéme, ktoré by mu prezradili, na čo je tento systém určený. Môže sa jednať napríklad o výpis logov používania, časy reštartov, počty a typ použitých hardvérových prostriedkov a podobne. Na základe týchto informácií môže malvér usúdiť, že sa s veľkou pravdepodobnosťou nenachádza na systéme bežného užívateľa a nespustiť sa. Konkrétny príklad môže byť získanie súčtu záznamov z logu používania a akonáhle tento súčet neprevyšuje istý počet, tak malvér berie systém ako sandbox.

Predĺženie analýzy

Kvôli limitovaným zdrojom každého sandboxu je analýza obmedzená na istý časový interval. Tento interval je rádovo v jednotkách minút. Kvôli tomu sa veľakrát malvér snaží pozdržať spustenie svojho hlavného kódu, aby sa vyhol analýze. Na pozdržanie analýzy využíva malvér viacero časových techník. Najviac používané sú napr. konštatný čas, po ktorom malvér spustí škodlivú činnosť, takže po spustení sa malvér napr. na 2 minúty zastaví alebo viacnásobné pozdržanie analýzy, ktoré prebieha viacnásobným volaním časových funkcií. Tzn. malvér napr. zavolá 1000krát `Sleep` funkciu s časovým intervalom 10ms.

Zahltenie monitoru

Monitorovacie prostriedky v rámci sandboxu slúžia na zaznamenávanie použitých API funkcií. Avšak veľa malvérov sa pokúša na istú časovu dobu zahltiť tieto monitory veľkým množstvom volaných API funkcií a tým sa úplne vyhnúť analýze. Toto zahltenie sa môže vykonávať volaním 1 alebo väčšej skupiny API volaní. Môže sa jednať napríklad o slučku s miliónom opakovaní v ktorej sa bude volať metóda `GetTickCount` a `GetNativeSystemInfo`.

Obchádzanie detekcie API volaní

Tu sa môže radiť malvér, ktorého tvorca si je vedomý toho, ako sandboxy fungujú a tým pádom sa snaží vyhnúť logovaniu, alebo meneniu použitých API funkcií v jeho malvére. Môže to byť viacero spôsobmi. Napríklad tým, že hooky ktoré používa sandbox jednoducho vráti do pôvodného stavu a tým pádom už sandbox nevidí aké API volania sa vykonávajú. Ďalšími spôsobmi sú napríklad priame volanie funkcionality kódu API funkcií alebo volanie API funkcií z kópie DLL knižnice.

Detekcia obchádzacích techník

Sandboxy dokážu vo veľa prípadoch spomínané techniky detekovať a vyhnúť sa im. Napríklad v prípade časovom predlžení analýzy preskočia niektoré volania `Sleep` metód. A túto skutočnosť sa práve snažia detekovať tieto malvéry. Detekovanie v tomto prípade môže pozostávať z volaní 3 API metód. Prvou malvér získa aktuálny čas, následne spustí funkciu na pozastavenie (napr. `Sleep`) a potom opäť získa aktuálny čas. Na koniec porovná 2 časové hodnoty a pokiaľ ich rozdiel je menší ako čas na ktorý malvér pozastavil svoje vykonávanie tak vie, že bola použitá technika na preskakovanie týchto funkcií a preto sa malvér ukončí.

4 Hĺbková analýza malvéru

Pre správne pochopenie funkcionality malvéru je potrebné vykonať hĺbkovú analýzu s cieľom odhaliť čo daný malvér vykonáva a aké prostriedky na vykonávanie svojej funkcionality používa. Jedná sa o kombináciu statickej a dynamickej analýzy za pomoci rôznych komplexných softvérov, ale aj hlbokej znalosti ako operačného systému tak aj jeho funkcií. Kľúčovu úlohu na zisťovanie anti-sandbox techník zohráva ich prvotná znalosť, ktorá bola bližšie opísaný v kapitole 3.1. Vo väčšine prípadov sa následne zistí či malvér obsahuje niektorú z techník jednoduchým pozorovaním toho, že sa plne nespustí v sandboxe alebo debuggovaním kódu malvéru. To poskytne nutný základ, alebo minimálne predstavu toho o akú techniku by sa mohlo jednať a tomu sa prispôsobia aj nasledujúce kroky. Ďalej bude táto kapitola obsahovať opis 5 malvérových rodín, ich históriu ako aj analýzu a zdôraznenie techník, ktoré sa v nich využívajú. Bude analyzovaná najmä najdôležitejšia časť, ktorá poslúži na ziskanie prehľadu implementácie spomínaných techník. Dôvod prečo boli vybraté tieto malvérové rodiny je ten, že obsahujú viacero anti-sandbox techník, ktoré náležite poslúžia ako forma ukážky ich praktickej implementácie.

Pre upresnenie, každá jedna analýza obsahuje hash SHA-256 analyzovaného binárneho súboru. Spôsob pomenovania škodlivých vzoriek ich hash hodnotou, ktorá zvyčajne býva typu MD5, SHA-1 alebo SHA-256 je zaužívaná prax medzi malvérovými analytikmi. Mimo to sa jedná o pseudo jedinečný identifikátor pod ktorým bývajú tieto súbory uvádzané a je možné ich dohľadať napríklad pomocou služby Virus Total. Každý jeden analyzovaný súbor je možné nájsť na priloženom CD.

4.1 Emotet

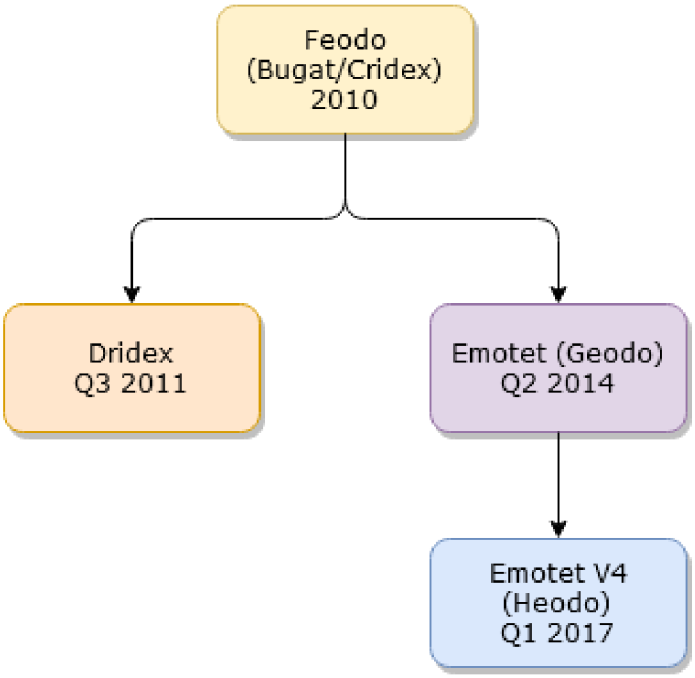
Emotet je modulárny malvér, ktorý je v dnešnej dobe často používaný ako loader iných malvérov. Bol prvýkrát identifikovaný v roku 2014. Pôvodne sa jedna o bankového trojského koňa určeného na kradnutie finančných informácií z online bankovníctva pomocou man-in-the-browser (MITB) útokov. Avšak od roku 2017 je jeho využitie najmä na distribúciu iných malvérových rodín ako napríklad IdedID, Zeus Panda a TrickBot. Malvér je aktívne vyvíjaný a každá nová verzia menila, alebo rozširovala jeho schopnosti [13].

Schopnosti Emotetu sú celkom rozsiahle. Ako bolo spomenuté, tak dokáže sťahovať a spúšťať iné malvérové rodiny, zvyčajne napríklad iné bankové trojské kone. Ďalej sa snaží hrubou silou prelomiť slabé heslá na službách bežiacich na napadnutom OS pomocou zabudovaného slovníku najčastejšie používaných hesiel. Taktiež umožňuje kradnúť prihlasovacie údaje z webových prehliadačov a emailových klientov za pomoci

dôveryhodných softvérov tretích strán ako napr. NirSoft Mail PassView. Okrem hesiel od emailov dokáže ukradnúť celé zoznamy emailových kontaktov. Jedna z jeho silných stránok je práve to, že po nakazení počítača dokáže toto zariadenie následne zneužiť na ďalšie rozosielanie malvéru a taktiež toto napadnuté zariadenia je zaradené ako bot do Emotet botnet siete a plní príkazy command and control (C2) bodu.

Emotet taktiež využíva niekoľko anti-analyzačných techník určených na zmarenie detekcie tohto malvéru. Jedná sa napríklad o využitie polymorfického pakeru, ktorý zbalí pôvodný binárny súbor malvéru a vytvorí z neho nový s inou veľkosťou a vnútornou štruktúrou. Obsahuje šifrované importy a názvy funkcií, ktoré sú deobfuskované a následne dynamicky nalinkované počas behu malvéru. Technika známa ako self-injection, ktorá injektne svoj kód sama do seba sa tu využíva tiež často. A nakoniec je to ešte šifrovanie komunikácie s C2 kanálom cez HTTP. Emotet využíva na komunikáciu symetrickú šifru AES. V starších verzicách sa jednalo o prúdovú šifru RC4 a dáta zaslané C2 neboli obsiahnuté v parametroch HTTP požiadavky, ale v jeho Cookie hlavičke [13].

Na základe pozorovaní vývinu malvéru koluje domienka, že Emotet zdieľa časť zdrojového kódu so starším banking trojanom Feodo taktiež známym ako Bugat alebo Cridex. Ich spojitosť je možné vidieť na diagrame 4.1.

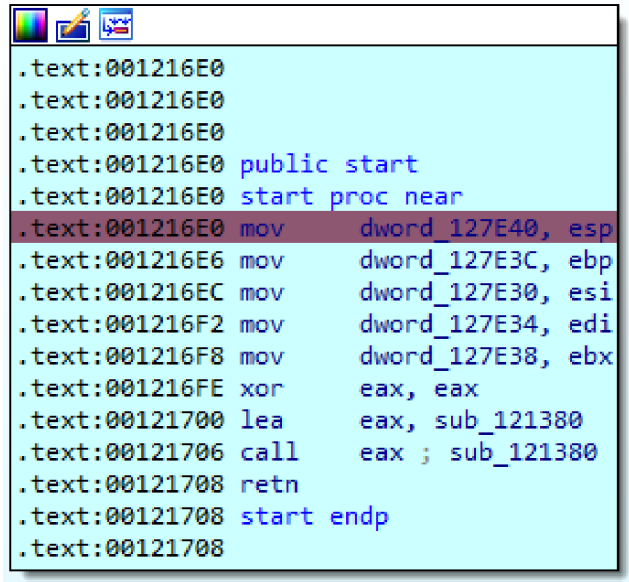


Obr. 4.1: Stromová štrukúra vývinu malvérovej rodiny Emotet [13]

Analýza

Vzorka: 0dab41c3567f3702a146e94950f82daa6176435fedf157ec203ef16b38eac1af

Prvý krok v analýze je zistenie kompilátora resp. jazyka v ktorom je vzorka napísaná, čomu sa následne prispôsobia ďalšie kroky analýzy ako aj použité nástroje. Na to sa použije nástroj Detect It Easy (DIE). V tomto prípade sa jedná o Microsoft Visual C/C++ a z toho dôvodu bude následná analýza pokračovať v disassembleri IDA PRO. V prvom rade bola skontrolovaná tabuľka importovaných Windows API funkcií, ktoré program bude používať. Na prvý pohľad v nej nebolo nič nezvyčajné. Po tom čo IDA ukončila statickú analýzu vzorky zobrazila jej funkcie, ktoré sa tam nachádzajú. Nakoľko ich nebolo moc tak nasledovala kontrola práve týchto funkcií. Okrem toho, že IDA je disassembler obsahuje aj pokročilý debugger. Ďalším krokom teda bolo spustenie malvéru v debuggeri. Tento debugger umožňuje aj debuggovanie programu na vzdialenom počítači cez nadviazané TCP spojenie tzv. Remote debugging (RD). Vo virtuálnom stroji s OS Windows 7 bol nainštalovaný IDA remote debugger server, ktorý bude prijímať príkazy z IDA klienta a na základe týchto príkazov bude vykonávať akcie debuggovania a späť bude zasielať debuggovacie údaje ako sú napríklad hodnoty registrov, hodnoty na zásobníku a iné. Server je v pôvodnom nastavení nastavený na zasielanie požiadavkov v rámci localhostu s portom 23 946. Tieto údaje sa špecifikujú do RD nastavení na hlavnom počítači v IDE. Následne sa vloží breakpoint na start funkciu ukázanú na obrázku 4.2, ktorá spúšťa hlavný kód a zapne sa debugging.



Obr. 4.2: Debugger vo funkcii start

Breakpoint zastaví vykonávanie behu programu na adrese `1216e0`. Jednou z naj-

dôležitejších inštrukcii na ktoré sa treba zamerať je call, ktorý volá iné funkcie. Po prejdený do funkcie `sub_121380` je vidno hneď niekoľko volaní API funkcií ako napríklad `GetCurrentDirectoryA`, `GetCurrentThreadId`, `GetCommandLineW`. Prvá volaná funkcia `sub_1212A0` vytvára semafor čo je synchronizačný objekt využívaný pri ovládaní zdieľaného prostriedku, ktorý dokáže podporovať len obmedzený počet používateľov [14]. Avšak čo je neobvyklé tak ten istý semafor vytvára 20x. Ďalej je taktiež možné vidieť volanie tejto funkcie v cykle spolu s funkciou `PathFileExistsW`. Jedná sa o prvú anti-sandbox techniku zvanú API-hammering. Princíp tejto techniky spočíva v zahltení analyzačného nástroja veľkým množstvom API volaní za účelom oneskorenia analýzy, alebo jej úplneho zrútenia. V tomto prípade sa jednalo len o zhruba 400 volaní funkcie `OpenSemaphoreW` s parametrom `lpName` „PSNGUbgM“. Ďalej funkcia `sub_121380` obsahuje ešte niekoľko API volaní bez väčšieho významu. Nakoniec sú tu ešte 2 špecifické funkcie. Funkcia `sub_121D80` ktorá obsahuje ďalšiu funkciu `sub_1211F0` obsahujúcu inú anti-sandbox techniku, tentokrát zameranú na detekovanie preskakovania `Sleep` funkcie keďže veľakrát keď sa malvér snaží zdržať analýzu metódami ako `Sleep`, tak sandbox na také správanie reaguje čiastočným alebo úplným preskočením takého volania čo spôsobí, že sa analýza nepredĺži. Pre lepšiu prehľadnosť bude zobrazená táto funkcionalita v pseudo-kóde IDA dekompilátoru na obrázku 4.3.

```
1  BOOL sub_1211F0()
2  {
3    DWORD v1; // [esp+Ch] [ebp-2Ch]
4    struct mmtime_tag v2; // [esp+14h] [ebp-24h] BYREF
5    int v3; // [esp+20h] [ebp-18h]
6
7    v3 = 553592537;
8    v2.wType = 1;
9    timeGetSystemTime(&v2, 0xCu);
10   v1 = v2.u.ms;
11   Sleep(50u);
12   v2.wType = 1;
13   timeGetSystemTime(&v2, 0xCu);
14   return v2.u.ms - v1 < 40;
15 }
```

Obr. 4.3: Emotet Sleep anti-sandbox technika

Zmysel tejto funkcie je v tom odhaliť či sandbox skráti alebo úplne preskočí pozastavenie programu pomocou `Sleep` funkcie, ktoré má trvať 50 milisekúnd. Na riadku 14 sa kontroluje či je rozdiel volaní API funkcií `timeGetSystemTime` menší ako 40 milisekúnd čo sa za behu na klasickom zariadení nestane. Ďalej sa preskúma metóda `sub_121E80`, ktorá obsahuje veľké množstvo iných funkcií. V nich sa nenašlo

nič podstatené a kvôli tomu sa preskočí na adresu `122070` s inštrukciou `call ecx`. `sub_A1B0C` obsahuje ďalšiu techniku detekcie sandboxu na základe užívateľského mena, ktoré zistí pomocout funkcie `GetUserNameA` a názvu počítaču `GetComputer-NameA`. Tvorca Emotet malvéru zistil mená najčastejšie používaných užívateľov, ktoré sa používajú v sandbox analýzach, rovnako ako aj názvy počítačov, ktoré sa následne porovnávajú so zistenými menami a na základe toho pokračuje malvér vo svojej činnosti alebo sa ukončí.

```
if ( kernel32_lstrcmp(computer_name, "TEQUILABOOMBOOM") )
{
  if ( kernel32_lstrcmp(user_name, "Wilbert")
    || kernel32_lstrcmp(a28, &unk_123110) && kernel32_lstrcmp(a28, &unk_123113) )
  {
    if ( kernel32_lstrcmp(user_name, "admin")
      || kernel32_lstrcmp(coumputer_name_ex, "SystemIT")
      || (v45 = kernel32_CreateFileA("C:\\Symbols\\aagmmc.pdb", 0x80000000, 0, 0, 3, 0, 0),
          kernel32_CloseHandle(v45),
          v46 = 1,
          v45 == -1) )
    {
      if ( kernel32_lstrcmp(user_name, "admin")
        || (v46 = 1, kernel32_lstrcmp(computer_name, "KLONE_X64-PC")) )
      {
        a31 = 0;
        a30 = 0;
        if ( kernel32_lstrcmp(user_name, "John Doe") || kernel32_lstrcmp(&a30, "BEA-CHI") )
        {
          if ( kernel32_lstrcmp(user_name, "John")
```

Obr. 4.4: Emotet anti-sandbox technika pomocou známych názvov

Na obrázku 4.4 si môžete všimnúť, že na porovnávanie sa používa funkcia `lstrcmp` a blacklist užívateľských mien v tomto prípade pozostáva z: Wilbert, admin, John Doe a John. Názvy počítačov nesmú obsahovať názvy ako: TEQUILABOOMBOOM[1], KLONE_X64PC, SystemIT a BEA-CHI. Avšak pri menách a názvoch počítačov to nekončí. Treťou skupinou sú súbory, ktoré sa na sandboxoch môžu nachádzať. V tomto prípade sa overujú pomocou funkcie `FileWriteW`, ktorá v prípade, že daný súbor už existuje, vráti chybu. Vďaka tomu program vie, že daný súbor existuje a ukončí sa. Jedná sa o súbory:

- C:\a\foobar.gif
- C:\a\foobar.doc
- C:\email.doc
- C:\email.htm
- C:\123\email.doc

[1]Názov TEQUILABOOMBOOM využíva vo svojich sandbox riešeniach známa služba VirusTotal čo je hlavný dôvod prečo sa tu tento názov vyskytuje.

- C:\123\email.docx

To je v rámci anti-sandbox technik tohto malvéru všetko. Na koniec tejto analýzy ešte stojí za zmienku, že malvér využije self-injection pomocou `NtCreateSection` a `NtMapViewOfSection` a následne už vykonáva svojú hlavnú škodlivú činnosť.

4.2 NjRat

NjRat taktiež nazývaný Bladabindi je Remote Access Trojan (RAT), ktorý sa prvýkrát objavil v júni 2013 a jeho prvé vzorky sa datujú do novembra 2012. Tento malvér je postavený na .NET frameworku. Bol vyvinutý a taktiež podporovaný arabsky hovoriacimi obyvateľmi a primárne používaný na počítačovú kriminalitu skupinami útočiacimi obzvlášť na Blízkom východe. Okrem útokov cielených na vlády v tejto oblasti je tento malvér použitý na ovládanie botnetov a vykonávanie ďalších typických činností v oblasti počítačovej kriminality. Svoje obete primárne infikuje phishingovými útokmi, cez infikované USB kľúče, alebo sieťové disky. Môže sťahovať a spúšťať ďalší malvér, spúšťať shell príkazy, čítať a modifikovať hodnoty registrov, zaznamenávať snímky obrazovky, zaznamenávať stlačené klávesy a špehovať obete cez webové kamery. V roku 2014 spoločnosť Symantec analyzovala vzorky NjRat malvéru a odhalila 542 doménových mien serverov C2 a 24 000 infikovaných počítačov po celom svete. 80 percent serverov C2 bolo umiestnených práve na Blízkom východe a v severnej Afrike [15].

NjRat využíva niekoľko techník, vďaka ktorým vie zabrániť detekcii antivírusovým softvérom. Napríklad používa viacero obfuskátorov na zneprehľadnenie a skrytie častí svojho kódu. Ďalšou technikou, ktorú tento malware používa, je maskovanie sa ako kritický proces. Táto metóda blokuje schopnosť používateľa vypnúť daný škodlivý proces, čo komplikuje odstránenie tohto malvéru z infikovaných počítačov. Okrem iného umožňuje tento RAT aj deaktivovať procesy patriace antivírusovým softvérom, čo mu umožňuje fungovať nepozorovane. Niektoré NjRat verzie sú schopné zistiť, či bežia na virtuálnom stroji, čo pomáha útočníkom v tvorbe protiopatrení proti analytikom alebo automatizovaným analýzam v sandboxoch [16].

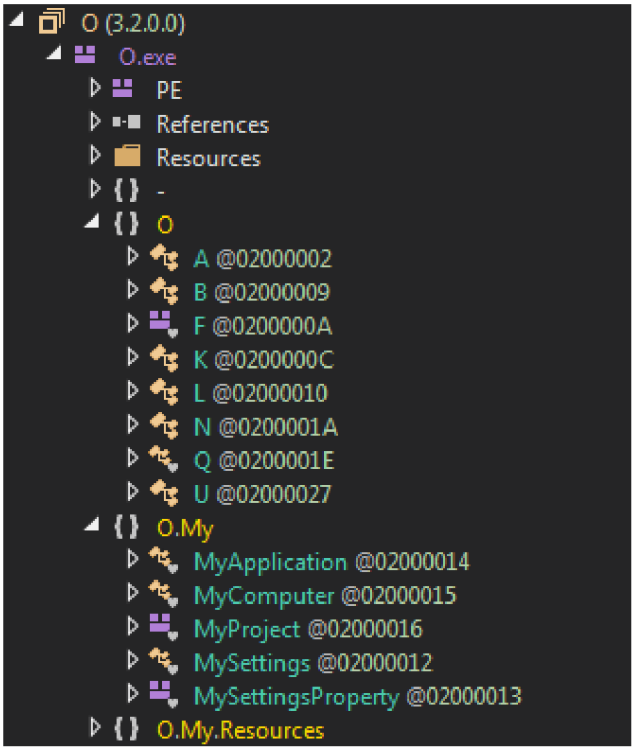
Analýza

Vzorka: ba504e5caeef1b53814887a0db2f0b9ead419ff10b8515599a69fb8d0e67832d

Pomocou nástroja DIE sa zistilo, že je malvér napísaný v programovacom jazyku VB.NET čo robí jeho analýzu jednoduchšou, nakoľko je možné využiť príslušný dekompilátor. Po otvorení tejto vzorky v dekompilátore DnSpy[2] je možné už na

[2]https://github.com/dnSpy/dnSpy

prvý pohľad povedať, že táto vzorka obsahuje základné prvky obfuskácie ukázané na obrázku 4.5. Ktorými sú jednopísmenové pomenovania tried podľa písmen abecedy.



Obr. 4.5: Ukážka NjRat obfuskácie

Avšak po náhľade do tried samotných sa zdá, že buď sú aj názvy niektorých metód obfuskované, alebo autor malvéru nepísal názvy metód dostatočne zrozumiteľne. V tomto prípade, keď je telo metód čitateľné a bez nutnosti deobfuskácie, zvyčajne postačuje statická analýza kódu, bez nutnosti jeho spustenia a následného debuggovania. Keďže sa anti-sandbox techniky vyskytujú hlavne na začiatku spustenia malvéru tak analýzu začneme priamo z entry pointu. Na entry pointe sa spúšťa `WinForm` trieda s názvom `A`. V tom prípade sa začne analýza na konštruktore tejto triedy nakoľko kód konštruktoru sa vykonáva prvý. Prvé riadky konštruktoru obsahujú pridanie akcií v prípade, že sa zavolajú udalosti `FormClosed`, `FormClosing` alebo `A_Load`. Metódy spúšťané pri zatvorení sa len jednoducho snažia spustiť daný malvér nanovo, zatiaľ čo metóda `Load` býva zvačajne využívaná na spustenie prvotnej funkcionality. Pred posunutím sa do tejto metódy stoja za povšimnutie premenné, ktoré sa v konštruktore tejto triedy inicializujú a deklarujú. Jedná sa napríklad o názov mutexu, IP, port a iné konfiguračné vlastnosti viď. obrázok 4.6, ktoré následne upravujú chod samotného malvéru a ovplyvnia aké jeho funkcie sa budú vykonávať a aké naopak nie.

```
3   public A()
4   {
5       base.FormClosed += this.A_FormClosed;
6       base.FormClosing += this.A_FormClosing;
7       base.Load += this.A_Load;
8       this.host = "MTI3LjAuMC4x";
9       this.ID = "HacKed";
10      this.port = "NTAwMA==";
11      this.EXE = "Svchost.exe";
12      this.flder = "Microsoft";
13      this.appdata = "True";
14      this.Temp1 = "False";
15      this.Documents = "False";
16      this.melt = "True";
17      this.Star = "True";
18      this.Hideme = "True";
19      this.RT = "((Mutex))";
20      this.BN = null;
21      this.SPR = Conversions.ToBoolean("True");
22      this.BD = Conversions.ToBoolean("True");
23      this.Usname = "SPK";
24      this.NOCOP = "False";
25      this.STC = "Svchost";
26      this.Protc = "True";
27      this.RegStr = "True";
```

Obr. 4.6: Ukážka konfiguračného súboru NjRat

Po vstúpení do metódy `A_load` malvér získa zoznam všetkých aktuálne bežiacích procesov a postupne ich jeden po druhom porovnáva s údajmi aktuálneho procesu. Jedná sa pravdepodobne o prvotnú kontrolu toho, či už jedna inštancia tohto procesu nebeží a v prípade, že áno tak sa malvér ukončí. Prekvapivo za touto funkcionalitou malvér skúša otvoriť mutex `((Mutex))`. V prípade, že by sa mu to podarilo a uspel by, tak by vedel, že už jedna inštancia tohto malvéru beží a ukončil by sa. V tomto prípade by ale pokračoval ďalej. Keďže malvér vie, že mutex s takým menom neexistuje a zároveň potrebuje zabezpečiť, aby malvér nebežal vo viacerých inštanciách tak taký mutex sám vytvorí. Ďalej pokračuje funkcionalita, ktorá by sa snažila pozastaviť beh programu, avšak hodnota `this.NOCOP` z konfiguračného súboru je nastavená na `False` a tým pádom sa táto časť kódu preskočí. Za tým nasledujú akcie ovplyvnené tiež nastavením konfiguračného súboru v tomto prípade sa jedná o umiestnenie kam sa daný malvér presunie a následne spustí. Je to z toho dôvodu, že keď obeť spustí malvér napr. z pracovnej plochy, tak aj bežný užívateľ ho vie vymazať. Avšak len málo užívateľov vie o `TEMP` priečinkoch kde sa tento malvér uloží nanovo. Po všetkých týchto akciách nevyhnutných na nastavenia konečnej lokácie sa volá metóda `Sleep`, ktorá má v sebe hodnotu z konfiguráčného súboru, ktorá bola naplnené dynamicky v konštruktore triedy hodnotou 100 000, čo znamená 100 sekund, nakoľko parametre

tejto funkcie z knižnice `kernel32.dll` sa udávajú v milisekundách. 100 milisekund je dostatočne dlhý čas na to, aby sa prípadná analýza v sandboxe oneskorila natoľko, že pred jej ukončením sa nevykoná žiadna škodlivá časť malvéru a tým pádom nebudú detekované žiadne behaviorálne prvky tohto malvéru. Po ukončení `Sleep` funkcie sa malvér prekopíruje do inej zložky, nastaví atribút tohto novo nakopírovaného súboru na `hidden`, čím znemožní užívateľovi, ktorý nemá zobrazené skryté súbory ho vidieť, spustí novú inštanciu a hneď ukončí tú starú. Nová inštancia vykoná všetky doteraz spomínané kroky až na opätovné kopírovanie a spúšťanie ďalšej inštancie a namiesto toho spustí už hlavný škodlivý kód.

4.3 BaneChant

Jedná sa o malvér typu backdoor, ktorý sa objavil ešte v roku 2013. Šírenie malvéru bolo v rámci spear phishing kampane, ktorá obsahovala ako prílohu dokument, ktorý následne stiahol tento malvér. BaneChant obsahuje techniky, ktoré majú zabezpečiť jeho nepozorované fungovanie na systéme zahrňajúc práve aj anti-sandbox techniky. Názov škodlivého dokumentu ktorým sa malvér šíril bol preložený ako `Islamic Jihad.doc`. A kvôli tomu existuje podozrenie, že tento dokument bol použitý na zacielenie na vlády Blízkeho východu a Strednej Ázie [17].

Tento malware je významný z niekoľkých dôvodov. V provm rade je to fakt, že zisťuje či so systémom interaguje skutočný používateľ pomocou počítania kliknutí myši. Samozrejme, že taká technika sa vyskytla u malvéru aj pred tým, avšak v minulosti sa jednalo iba o detekciu jednoho kliknutia, vďaka čomu bola táto technika veľmi jednoducho prekonateľná. Ďalej má tento malvér aj anti-forenzné schopnosti ako napríklad to, že nezačne naplno fungovať dokým sa nezapne internetové pripojenie. Po zapnutí internetu sa do pamäte stiahne dodatočný škodlivý kód a následne sa aj spustí. To je jedna z vecí čím sa líši od väčšiny malvéru, ktoré spustia okamžite svoju škodlivú funkcionalitu. Táto funkcionalita v skutočnosti neslúží na nič iné, len na zabránenie analýzy a následne stiahnutie skutočého škodlivého kódu. Bolo zistené že stiahnutá dodatočná funkcionalita slúži na odosielanie informácií o počítači a vytvára v počítači backdoor na vzdiálený prístup pre útočnika. Následne tento backdoor umožňuje útočníkovi vykonávať veľké množstvo škodlivých činností.

Dôvod prečo sa malvér nazýva BaneChant je ten, že pri odosielaní dát na server má vo svojich HTTP požiadavkach spomenutý tento názov. Názov patrí zvukovému záznamu ktorý vytvoril Hans Zimmer pre film „The Dark Knight Rises“[3] [17].

[3] https://www.youtube.com/watch?v=qBZ9i5BNjvE

Analýza

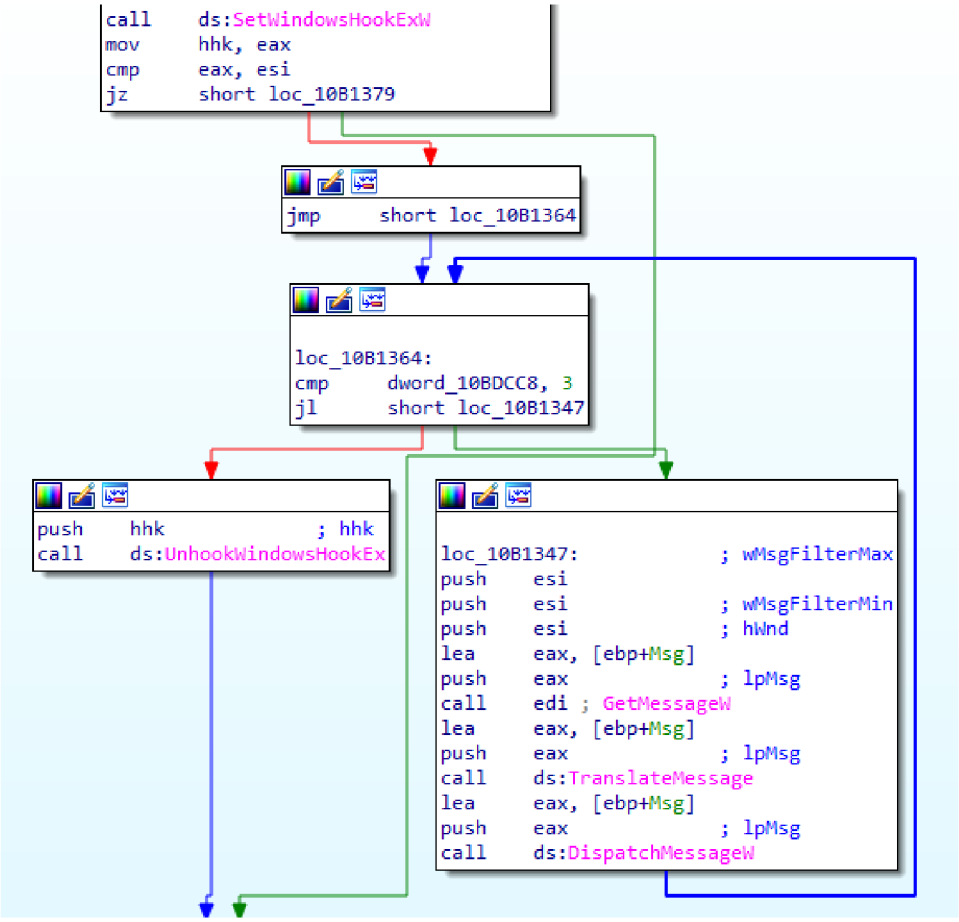
Vzorka: 2488e0fe5d01dfd2b9ed782f07910109fa9468c47fe5ee1777766c6a2acc7603

Po načítaní vzorky nástrojom DIE je zrejmé, že sa jedná o pakovaný súbor. Použitý je paker UPX. Avšak tento paker umožňuje okrem pakovania aj unpakovanie po špecifikovaní parametru „-d“. Unpakovací proces je možné vidieť na obrázku 4.7.

```
PS C:\Users\sidor\Desktop\upx308w>
PS C:\Users\sidor\Desktop\upx308w> .\upx.exe -d ..\upx_2488
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2011
UPX 3.08w       Markus Oberhumer, Laszlo Molnar & John Reiser

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     50688 <-     25088   49.49%    win32/pe     upx_2488

Unpacked 1 file.

PS C:\Users\sidor\Desktop\upx308w>
```

Obr. 4.7: UPX unpacking

Po unpakovacom procese sa v DIE overí, že súbor už skutočne neobsahuje UPX artefakty a následne sa môže začať analýza v IDE. Po načítaní vzorky je zrejmé, že sa nejedná o moc komplikovaný malvér. Stačí si všimnúť množstvo metód a taktiež veľkosť vzorky. Čo sa samotnej funkcionality týka tak je viditeľné, že sa najprv nastaví hook pomocou funkcie `SetWindowsHookExW` na event `WH_MOUSE_LL`, ktorý monitoruje udalosti kliknutia myši. Parameter `lpfn` v tejto funkcii predstavuje funkciu, ktorá sa vykoná v prípade, že bola takáto udalosť obdržaná. Obrázok 4.8 znázorňuje časť hook funkcionality.



Obr. 4.8: Ukážka BaneChant hooku

Hlavná logika využívajúca informácie z hooku je na `loc_10B1364`, kde premenná `dword_10BDCC8` predstavuje počet stlačení myši a číslo 3 zase predstavuje minimálny počet stlačení pre pokračovanie ďalšej dôležitej funkcionality. Tieto akcie predstavujú ďalšiu známku anti-sandbox techniky, ktorou je prerušenie analýzy dokým sa nevykoná akcia používateľa. V tomto prípade niekoľkonásobné kliknutie myši. Hodnotu počtu stlačení myši mení funkcia špecifikovaná pri volaní `SetWindowsHookExW`. Po 3 a viacerých kliknutiach sa zavolá funkcia `UnhookWindowsHookEx` po ktorej nasleduje 2x volanie metódy `sub_10B1053`, ktorá slúži na dešifrovanie názvu webu s ktorým bude potom malvér komunikovať. Dešifrovaná doména má tvar `hxxp://kibber[.]no-ip[.]org/adserv/logo[.]jpg`. Následne skontroluje či existuje súbor `csetup32.dll`, a ak nie pokračuje ďalej na funkciu `sub_10B118F` kde sa snaží stiahnuť hlavný payload pomocou funkcie `InternetReadFile`, ktorý sa následne spustí a BaneChant loader sa ukončí.

4.4 NanoCore

NanoCore RAT sa prvýkrát objavil v roku 2013 a začal sa predávať na rôznych hackerských fórach. Tento malvér má veľké množstvo funkcií, ako napríklad keylogger alebo nástroj na krádež hesiel, ktorý môže zasielať tieto údaje v reálnom čase operátorovi tohto malvéru. Má tiež schopnosť manipulovať a prezerať si záznam z webových kamier, manipuláciu, sťahovanie a krádež súborov a registrov ako aj veľa ďalších funkcií. Aktuálne sa NanoCore šíri prostredníctvom spamových kampaní využívajúcich sociálne inžinierstvo. Napríklad sa môže jednať o e-mail obsahujúci falošné potvrdenie o bankovej platbe alebo o obdržaní balíka. Takéto e-maily veľakrát obsahujú škodlivé prílohy s rôznym množstvom prípon, alebo ich kombinácií ako napríklad `.cmd`, `.bat`, `.pdf.exe` a iných. Mimo e-mailových kampaní sa NanoCore šíri aj vo phishingových kampaniach využívajúcich špeciálne vytvorené ZIP archívy, ktoré sú navrhnuté tak, aby obchádzali zabezpečené e-mailové brány. Iné súborové formáty využívané pri podobných emailových alebo phisingových kampaniach sú aj tie patriace do balíka MS Office. Jedná sa predovšetkým o Word, Excel alebo PowerPoint. Konkrétne pri týchto typoch kampaní sa do spomínaných Office dokumentov vloží škodlivé makro, ktoré po spustení stiahne malvér na počítač obete [18].

Tvorca tohto malvéru Taylor Huddleston, alias „Aeonhacks“, v roku 2016 priznal, že vyvíjal, predával a distribuoval NanoCore na hackerských fórach v rokoch 2012 až 2016. Následne bol zatknutý a odsúdený na tri roky vo federálnom väzení za napomáhanie a vnikanie do počítačov. To však, bohužiaľ, nespomalilo šírenie tohto malvéru. Hackeri udržiavali Taylorov výtvor pri živote a vydávali nové verzia v priebehu nasledujúcich rokov [19].

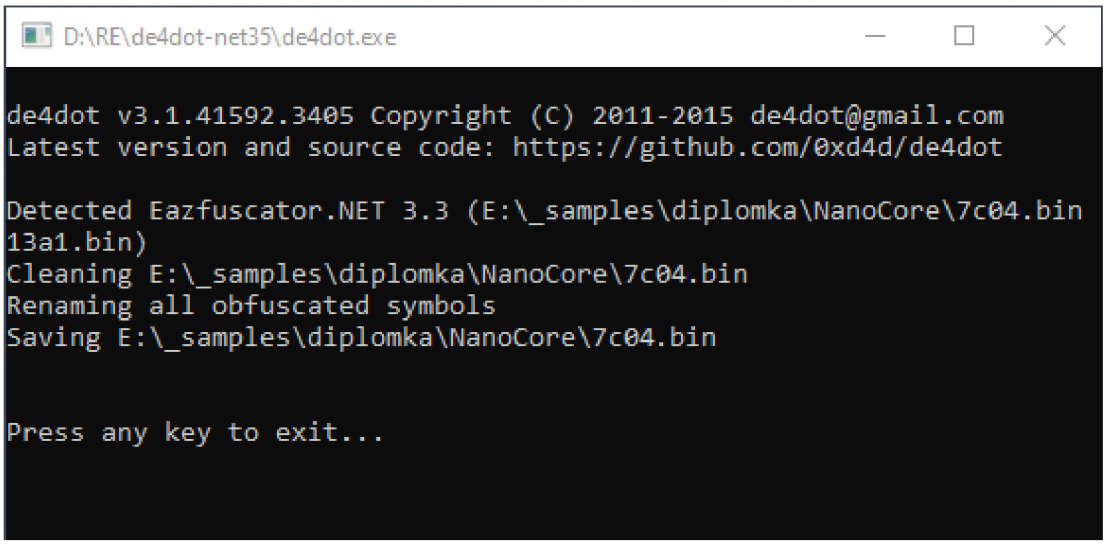
Analýza

Vzorka: 7c046dcff9b4a07a5c17d8561046b6c5e727cbe611aca12080237001507d13a1

Pomocou programu DIE sa zistilo, že je malvér písaný v jazyku VB.NET a tým pádom sa môže opäť využiť dekompilátor DnSpy ako aj pri analýze NjRatu. V tomto prípade sa však jedná o vzorku ktorá je obfuskovaná. Na túto skutočnosť upozorňuje DIE a bližšie špecifikuje, že sa jedná o Eazfuscator.NET[4]. Kvôli tomuto je nutné vzorku najprv deobfuskovať. Na to slúži nástroj De4dot[5], ktorý sa využíva na deobfuskáciu .NET malvéru, viď. obrázok 4.9.

[4]https://www.gapotchenko.com/eazfuscator.net

[5]https://github.com/de4dot/de4dot



Obr. 4.9: NanoCore deobfuskácia pomocou D4dot

Pre porovnanie na obrázku 4.10 môžte vidieť metódu pred obfuskáciou a následne na obrázku 4.11 metódu po obfuskácii.

```
// Token: 0x060001CE RID: 462 RVA: 0x0000C480 File Offset: 0x0000A680
[STAThread]
public static void Main()
{
    Application.Run(#=q_jQLaNdtSDa6ovA0VGw50w==.#=qqROT7DfncW7strhZvp0iRQ==()
                    .#=qbzig1$2CwLluEJt5uPtpgqPx5y_2S$GoPgJP36N8bTE=());
}
```

Obr. 4.10: Ukážka obfuskovaného kódu

```
// Token: 0x060001BC RID: 444 RVA: 0x00002E97 File Offset: 0x00001097
[STAThread]
public static void Main()
{
    Application.Run(Class1.smethod_3().method_0());
}
```

Obr. 4.11: NanoCore kód po deobfuskácii

Nakoľko použitá obfuskácia nenávratne zmenila mená tried a metód, tak už nie je možné žiadným automatizovaným nástrojom obnoviť tieto pôvodné názvy. Avšak deobfuskáciou sa značne zvýšila čitateľnosť kódu. DnSpy je okrem dekompilátoru aj

debugger a nakoľko by statická analýza kódu bola v tomto prípade komplikovanejšia, tak sa vykoná dynamická analýza práve za pomoci tohto debuggeru. Pri spustení debuggovania sa vyberie možnosť pozastavenia programu na Entry Pointe. Entry Point v tomto prípade odkazuje na `Main` metódú v triede `ClientLoaderForm`, ktorá volá vytvorenie objektu `ClientLoaderForm`. V konštruktore tohto objektu je možné vidieť udalosti na ktoré sa nastavujú volania metód. V prípade udalosti `FormClosing`, ktorá sa volá počas zatvárania formuláru sa vykoná metóda obsahujúca funkcionalitu, ktorá v prípade, že je to v konfiguračnom súbore povolené, nastaví aktuálne bežiaci proces ako kritický proces. Tým pádom po prípadnom ukončení tohto procesu by sa systém zrútil a zobrazila by sa modrá obrazovku s diagnostickými údajmi (táto modrá obrazovka sa často nazýva aj Blue Screen of Dead čiže skrátene BSOD). Následne je nutný reštart OS. Druhá udalosť `Shown` slúži na spustenie hlavnej funkcionality po dokončení inicializácie celého `Form` objektu užívateľského rozhrania (GUI). Avšak aj keď sa jedná o `Form objekt`, inak povedané základný objekt užívateľského rozhrania, tak sa nezobrazí žiadne okno. To zabezpečuje vlastnosť `WindowState` nastavená na `Minimized`, čo zabezpečí, že sa okno spustí minimalizované a zároveň sa neukáže na lište spustených programov lebo vlastnosť `ShowInTaskbar` je nastavené na `False`. Po tejto prvotnej analýze je zrejmé, že ďalšia funkcionalita programu bude pokračovať po volaní metódy `ClientLoaderForm_Shown`. V nej sa dodatočne ešte nastavuje parameter `Visible`, ktorý ešte dodatočne zabezpečí aby sa skutočne žiadne GUI nezobrazilo a pokračuje ďalej metodou `Class8.smethod_0`. V tejto metode sa najprv nastavuje spracovanie výnimiek programu vlastnou metodou `Class8.smethod_58` a `Class8.smethod_57`. Za tým nasleduje funkcionalita `AssemblyResolve` udalosti, ktorá slúži na vkladanie NanoCore pluginov za behu, nakoľko sa jedná o modulárny RAT. Metóda `Class8.smethod_13` zabezpečí načítanie prostriedkov malvéru z `resource` sekcie. Metóda `smethod_20` kontroluje, či je povolený debug mód a ak áno, tak by do debug konzole začala pomocou metody `Console.Write` vypisovať debug hlášky. `Smethod_22` sa následne pokúša vytvoriť mutex vo formáte `Global{{GUID}}`, kde GUID je špecifická alfa-numerická hodnota vo formáte zapísanom vo forme regulárneho výrazu ako: `\w{8}-\w{4}-\w{4}-\w{4}-\w{12}`. V tomto prípade `c3ba576a-7ae0-4621-8da9-a372fd624ac2`. Ak sa vytvorenie mutexu nepodarí, tak sa malvér ukončí, nakoľko predpokladá, že už na počítači raz beží. Ďalej prichádza prvá anti-sandbox technika ukázaná na obrázku 4.12 a tou je oneskorenie spustenia ďalšej funkcionality v metóde `Class8.smethod_22`. Malvér skontroluje v konfiguračnom súbore hodnotu `RunDelay`, na koľko milisekúnd by mal pozastaviť vykonávanie svojej funkctionality a ak to nie je 0 tak pomocou metody `Thread.Sleep` sa toto pozastavenie vykoná.

Nasleduje získanie odtlačku systému (fingerprint) metódou `smethod_33`. V prípade, že má obeť vypnuté notifikácie o hláške „spustiť ako admin“, alebo je užívateľ

```
// Token: 0x06000073 RID: 115 RVA: 0x00002410 File Offset: 0x00000610
private static void smethod_22()
{
    if (Class24.smethod_17() != 0)
    {
        Thread.Sleep(Class24.smethod_17());
    }
}
```

Obr. 4.12: NanoCore anti-sandbox technika

prihlásený s administrátorskými právami tak sa malvér neskôr taktiež spustí s týmito vyššími právami. Aby malvér mohol fungovať aj po reštartovaní systému je nutné aby si zabezpečil, aby sa po každom zapnutí systému zapol tiež spolu s inými programami alebo službami (tento proces sa tiež nazýva perzistencia). Malvér si teda nastaví perzistenciu za pomoci zmeny programov spúšťaných počas štartu systému a tým pádom sa bude zapínať po každom novom načítaní OS (po reštarte alebo vypnutí a následnom zapnutí). Okrem toho malvér zabezpečí, aby sa počítač neprepol do spánkového režimu a tým následne nepozastavil jeho funkcionalitu. Po všetkých týchto krokoch sa spúšťa už samotná škodlivá činnosť, ktorá pozostáva z kradnutia užívateľských hesiel, odpočúvania stlačených kláves, sledovania webkamery a iných škodlivých techník.

4.5 Dridex

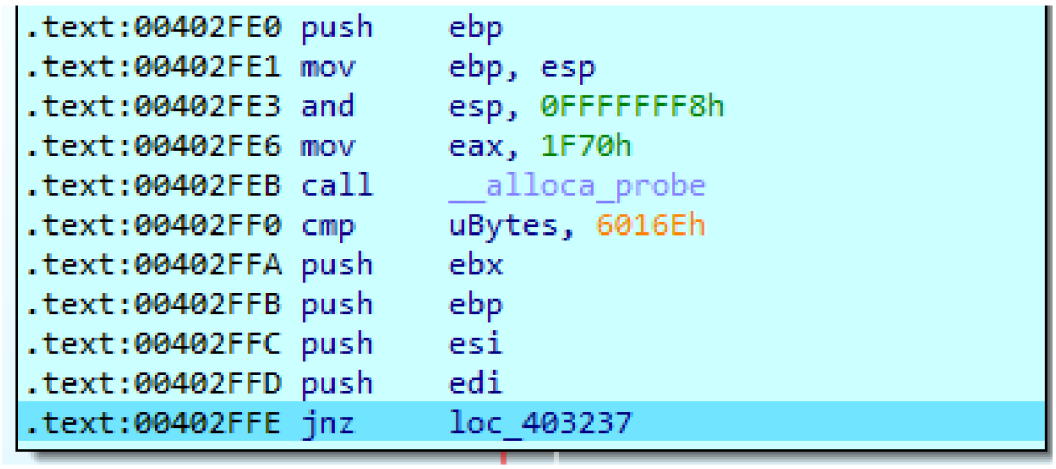
Dridex je škodlivý softvér (malvér), ktorý sa zameriava na prístup do bankových a finančných služieb užívateľov. Hlavný spôsob šírenia zabezpečuje využívanie makier balíka Microsoft Office v e-mailových kampaniach. Po infikovaní počítača môžu útočníci Dridexu ukradnúť bankové údaje, ako aj ďalšie osobné informácie užívateľov, aby následne získali prístup k ich finančným záznamom a službám [20]. Dridex sa samozrejme vyvýjal postupne vo viacerých verziách od jeho počiatku. Hlavným dôvodom týchto aktualizácií bolo primárne prispôsobovanie sa novým verziám prehliadačov. Funkcionalita Dridex malvéru pozostáva z viacerých modulov, ktoré môžu byť stiahnuté dohromady ako celok, alebo následované prvotným stiahnutím a spustením tzv. `Loader` modulu. Moduly obsahujú funkcionalitu na zhotovenie snímkov obrazovky, virtualizáciu, alebo napr. zahrnutie počítača obete do botnetu. Naprieč svojou históriou Dridex využíval viacero zraniteľností a spôsobov spúšťania zahŕňajúc modifikácie súborov, používanie súborov obnovy systému na zvýšenie svojich oprávnení, alebo úprava firewallových pravidiel na uľahčenie peer-to-peer (P2P) komunikácie na odoslanie užívateľských dát. Po stiahnutí a spustení má Dridex širokú škálu funkcií, od sťahovania ďalšieho softvéru cez vytvorenie virtuálnej siete až po mazanie súborov.

Primárnou hrozbou pre finančné aktivity je schopnosť Dridexu infiltrovať prehliadače, získať prístup k aplikáciám a webovým stránkam online bankovníctva a injektovať malvér, alebo softvér na zaznamenávanie kláves pomocou hookovania API funkcií a následného odcudzenia prihlasovacích údajov. Po odcudzení prihlasovacích údajov majú útočníci potenciál na vykonanie rôznych podvodných platieb, otváranie nových účtov, alebo využitie týchto údajov na iné podvodné účely [21].

Analýza

Vzorka: eb2c147c44265a3690c21048e4ac0a29acb05a30c3de0c8838b802ba9c893e65

Program DIE detekoval vo vzorke použitie Microsoft Visual C/C++ kompilátoru a taktiež, že sa jedná o konzolovú aplikáciu. Zvyčajne hlavná funkcionalita programu začína až na funkcii `WinMain`, avšak to neznamená, že to je prvá funkcionalita ktorá je v programe vykonávaná. Medzi začiatkom a `WinMain` funkciou sa nachádza ešte ďalší kód, ktorý rieši import modulov a nastavuje rôzne iné veci nevyhnutné na správny beh programu. Pokročilý malvér môže aj do tejto časti kódu vložiť škodlivý kód a tým pádom vykonať škodlivú činnosť skôr než by analytik očakával. Jedným z takých spôsobov môže byť napríklad volanie konštruktorov objektov, ktoré sú deklarované a inicializované pred hlavnou `WinMain` funkciou, tzn. ich funkcionalita sa vykoná skôr než sa začne vykonávať kód v hlavnej funkcii. To však nie je prípad tohto malvéru a kvôli tomu sa pri analýze prejde hneď na `WinMain` funkciu. Je tam možné vidieť jedno API volanie `Module32First`, ktoré vracia informáciu o prvom module špecifikovaného procesu. V prípade, že funkcia skončí chybou tak malvér pokračuje vykonávanie programu ďalej do funkcie `sub_402FE0`. Dôvod prečo sa pokračuje v prípade, že skončí funkcia neúspechom je ten, že jej vstupný parameter `hSnapshot` je nezmyselný a neobsahuje skutočný handle na objekt z funkcie `CreateToolhelp32Snapshot`. Tým pádom skončí vždy volanie chybou. Túto techniku je možné brať ako anti-sandbox techniku, keďže niektoré sandboxy pri podobných chybách môžu vrátiť hodnotu `True` a tým pádom by malvér nepokračoval do hlavnej funkcionality, ale na druhú stranu by sa ukončil. Tým pádom sa pokračuje do funkcie `sub_402FE0`. V tejto funkcii ukázanej na obrázku 4.13 sa premenna `uBytes` porovnáva s pevne stanovenou hexadecimálnou hodnotou `0x6016E`.



Obr. 4.13: Porovnanie konštanty s premennou `uBytes`

Výsledok tejto podmienky je vždy hodnota `False` tzn. Zero Flag bude nenulové číslo a tým pádom podmienka `jnz loc_403237` bude úspešná. V tomto prípade sa autor malvéru snažil zmiasť analytikov, nakoľko táto podmienka bude vždy splnená a preto sa nikdy neprejde do druhej vetvy kódu. Spomínaná druhá vetva kódu je ukázaná na obrázku 4.14 a obsahuje viacero volaní API funkcií s veľa nezmyselnými argumentami a zároveň sa výstupy týchto volaných funkcií ani nikde ďalej nevyužívajú.

```
IsBadStringPtrA(0, 0);
SetFileValidData(0, 0i64);
IsValidLocale(0, 0);
EnumTimeFormatsA(0, 0, 0);
FindFirstVolumeMountPointW(
  L"tibunerugu vicixazokipofezuyuxumi nesugivohetorufoyuxipibuyisita",
  szVolumeMountPoint,
  0);
DefineDosDeviceA(
  0,
  "kulavefewipurerumozuvohesike henecidufibayizajuwameki tubuvayecirugunogabawepu",
  "nagataxegeducuru xuru garayuvajadeta patuhebulojamizujiyana");
GetStringTypeExW(0, 0, L"ligavexuliwibopelewugawa giniri focihudirisami", 0, CharType);
FormatMessageA(0, 0, 0, 0, Buffer, 0, 0);
GetProfileIntW(
  L"yacikojuvewatepajiyuladi yolexuvulemoletipufuwalodoyi xivujopobakumukotagocoma",
  L"zixuwonuhimitominuzoto",
  0);
WaitNamedPipeW(L"zefajenogekapolacevimofejede", 0);
```

Obr. 4.14: Ukážka redundantnej funkcie

Následne kód pokračuje k volaniu ďalších API funkcií, ktorými sú `GetLastError`, `GetTickCount` a `GetNativeSystemInfo`. Tieto metódy sa volajú v cykle s počtom

opakovaní `0x1B9AB1` čo je zhruba 1,8 milióna volaní. Takže sa jedná o anti-sandbox techniku API hammeringu, ktorá sa snaží pozdržať analýzu v sandboxe ako aj o zahltenie analyzačného nástroja sandboxu. Pre lepší prehľad na zariadení s procesorom Intel Core i7-8750H 2.20GHz trvalo vykonanie takého množstva volaní takmer 2 minúty čo je dostatočne dlhý čas na to, aby sa analýza hlavnej časti kódu vo väčšine sandboxov vôbec nevykonala. Následne malvér naalokuje pamäťový blok, ktorému nastaví pomocou funkcie `VirtualProtect` prístupové práva RWX. Z toho je zreteľné, že sa z toho pamätového bloku bude spúšťať nejaký kód. V tomto prípade to bude shellcode, ktorý prepíše hlavný PE súbor novým a následne ho spustí. Jedna sa o techniku zvanú self-injection. Po použití tejto techniky malvér ďalej pokračuje vykonávaním ďalšej škodlivej činnosti, avšak to už nie je relevatné k zameraniu tejto práce.

5 Vylepšenie sandboxingu

Vylepšenie samotnej sandboxovacej techniky a vytvorenie prostredia, ktoré bude rezistentné voči špecifickým anti-sandbox technikám je hlavnou náplňou tejto práce.

5.1 Realizácia zlepšenia detekcie

Manuálnou hĺbkovou analýzou boli zistené viacere problémy aktuálnej Cuckoo sandbox implementácie. Z dôvodu hľadania nových rodín používajúcich tieto techniky a zároveň detekcie týchto techník v už známych rodinách je preto ďalším krokom systematická detekcia analyzovaných anti-sandbox techník. Táto systematická detekcia bude realizovaná za pomoci Cuckoo signatúr.

Vďaka Cuckoo je analytik schopný vytvárať špecifické signatúry, ktoré sa v sandboxe spúšťaju na výsledky analýzy za účelom nájdenia istých preddefinovaných vzorov. Spomínané vzory môžu reprezentovať isté škodlivé alebo podozrivé správanie. Tieto signatúry sú veľmi užitočné na získanie kontextu analýz, nakoľko značne zjednodušujú interpretáciu výsledkov rovnako ako aj automatickú detekciu malvérových vzoriek [22]. Keďže výstup analýzy sandboxu spočíva z veľkej časti z množstva použitých API volaní, ktorých ďalšie spracovanie by bolo komplikované, je voči nim spustený modul na detekciu signatúr, ktorý z nich dokáže získať veľmi dôležité informácie a vďaka tomu aj znížiť množstvo informácií určených na ďalšie spracovanie alebo zobrazenie. Boli navrhnuté nasledujúce detekčné signatúry:

antisandbox_sleep_with_ping

Účel signatúry 5.1 je detekovať vzorky, ktoré používajú funkciu ping (ICMP echo požiadavka) na oneskorenie analýzy. Existujú 2 spôsoby tohto oneskorenia. Prvý je zasielanie ping dotazu na localhost adresy istý početkrát. Každý jeden poslatý ping má nastavenú pôvodnú dĺžku odozvy 1 sekundu, takže volanie 60 ping funkcií bude trvať 60 sekúnd. Druhým spôsobom je zasielanie ping požiadavok na nedostupné adresy. Tu sa používa parameter na nastavenie doby, po ktorej ak koncová adresa neodpovie, ping volanie sa ukončí. Ak sa tento parameter nastaví na veľkú hodnotu, tak ping na adresu, ktorá je nedostupná zabezpečí, že sa vzorka v sandboxe nestihne analyzovať pred uplynutím časového intervalu.

```
import re
from lib.cuckoo.common.abstracts import Signature


class AntiSandboxSleepWithPing(Signature):
    name = "antisandbox_sleep_with_ping"
    description = "A process attempted to delay analysis using ping
                   sleeping technique"
```

```
    severity = 2
    categories = ["anti-sandbox"]
    authors = ["Samuel Sidor"]
    minimum = "1.2"
    evented = True

    def __init__(self, *args, **kwargs):
        Signature.__init__(self, *args, **kwargs)

    def on_call(self, call, process):
        if call["api"] == "IWbemServices_ExecQuery":
            query = self.get_argument(call, "Query").lower()

            if "win32_pingstatus" in query:
                self.process_command(query)

    def on_complete(self):
        executed_commands = self.results["behavior"]["summary"]
                                .get("executed_commands", [])
        for command in executed_commands:
            command = command.lower()
            if r"c:\windows\system32\ping.exe" in command:
                self.process_command(command)

        if len(self.data) <= 0:
            return False

        self.data.append({"Summary": "{} ping sleep attempt/s."
                                     .format(len(self.data))})

        return True

    def process_command(self, command):
        if self.contains_localhost_alias(command):
            self.data.append({"Sample":
                              "tried to sleep with use of ping to
                               localhost. Used command: '{}'"
                              .format(command)})
        elif self.contains_unreachable_ip(command):
            self.data.append({"Sample":
                              "tried to sleep with use of ping to
                               unreachable ip. Used command: '{}'"
                              .format(command)})

    @staticmethod
    def contains_localhost_alias(text):
        match = re.search(r"localhost|loopback|127\.\d+\.\d+\.\d+|127\.1|
                            ::1|0:0:0:0:0:0:0:1", text)
        if match:
            return True
        return False

    @staticmethod
    def contains_unreachable_ip(text):
        match = re.search(r"192\.0\.2\.\d+|198\.51\.100\.\d+|
                            203\.0\.113\.\d+", text)
        if match:
            return True
        return False
```

Výpis 5.1: Signatúra na detekciu oneskorenia analýzy pomocou funkcie ping

Signatúry sú rovnako ako samotný Cuckoo sandbox písané v jazyku Python 2.7. Na začiatku je možné si všimnúť importované moduly. Prvý z nich sa stará o importovanie funkcionality regulárnych výrazov a druhý importuje rodičovskú triedu `Signature`. Následne je definovaná trieda, ktorá dedí zo spomínanej rodičovskej triedy. Ako prvé premenné sa tu nachádzajú meta informácie. Meno a popis hovoria samé za seba. Premenná `severity` určuje o ako moc kritickú signatúru sa týka. Jednotka predstavuje najmenšiu závažnosť a 3 najväčšiu. Ďalej je kategorizovanie tejto signatúry a nakoľko sa jedná o detekciu anti-sandbox techniky, tak je zaradená práve do tejto kategórie. Premenna `minimum` predstavuje minimálnu použitú verziu Cuckoo a premenná `evented`, že sa jedná o signatúru, ktorá reaguje na udalosti tzn. využíva metódu `on_call`, ktorá je volaná v prípade všetkých API volaní z výstupnej analýzy. Prvou definovanou metódou je konštruktor, ktorý volá konštruktor rodičovskej triedy s danými hodnotami. Ďalej pokračuje metóda `on_call` s argumentami `call`, ktorý predstavuje objekt volanej API funkcie s jej parametrami a argument `process`, obsahujúci názov procesu, ktorý dané volanie vykonal. V tele tejto metódy sa najprv overuje či bola volaná API funkcia `IWbemServices_ExecQuery`. V prípade, že áno, získa sa jej argument `Query` a porovná sa s výrazom `win32_pingstatus`, ktorý predstavuje volanie ICMP echo dotazu. V prípade, že dané volanie bolo volanie funkcie ping, tak sa v metóde `process_command` vykonajú 2 kontroly. Prvou je, že či sa náhodou v tomto dotaze nenachádza localhost alias. Tým pádom je daný dotaz nezmyselný a žiadnym spôsobom nekontroluje dostupnosť vzdialenej služby. Druhá kontrola zase overuje IP adresu na ktorú je dotaz smerovaný. Je možné vidieť, že sa v metóde `contains_unreachable_ip` hľadajú len IP adresy, ktoré sú rezervované pre ukážky v článkoch a tým pádom by nemali byť nikde použité [23]. Po prejdení všetkých API volaní danej vzorky sa zavolá metóda `on_complete`, ktorá ešte dodatočne skontroluje či sa ping funkcia nevolala z príkazovej riadky a v prípade, že áno opäť vykoná nad jej argumentmi príslušné akcie. Ak sa v API volaniach vyskytovalo spomínané správanie, tak metóda do výstupu vloží všetky údaje zadané pomocou `self.data.append` a signatúru vyhodnotí ako `True`.

Signatúra bola získaná z dropperu, ktorý následne spúšťal malvér zvaný Disabler a využívala oneskorenie analýzy pomocou volania príkazu `ping -n 60 127.0.0.1` z príkazovej riadky. V danom príkaze parameter „n“ a číslo za ním predstavuje počet volaní ICMP echo požiadavku na ďalej uvedenú IP adresu. Bez uvedenia parametru je táto hodnota nastavená na 4.

recon_checkdomain

Aby sa obsah práce zbytočne nepredlžoval, tak zo signatúr budú ďalej v texte odobraté importy, meta informácie a ďalšie opakujúce sa časti. Celé signatúry je možné nájsť

na priloženom CD.

Nasledujúca signatúra 5.2 slúži na zistenie či vzorka využíva niektorú z API funkcií na získanie názvu domény, v ktorej sa dané zariadenie nachádza. Využívajú sa na to volania API funkcií `GetComputerNameExW`, `DsGetDcName` a `DsRoleGetPrimaryDomainInformation` so špecifickými parametrami, ktoré budú spomenuté pri popise signaútúry.

```
class CheckDomain(Signature):
    filter_apinames = {
        "GetComputerNameExW",
        "DsGetDcNameA",
        "DsGetDcNameW",
        "DsRoleGetPrimaryDomainInformation",
    }

    def on_call(self, call, process):
        if call["api"] == "GetComputerNameExW":
            arg_type = self.get_argument(call, "Type")
            if arg_type == "2" or arg_type == "6":
                self.data.append({"Sample":
                                  "Tried to get domain name with use of:
                                   kernel32.dll!GetComputerNameEx->{}"
                                  .format(arg_type)})
        elif call["api"] == "DsGetDcNameA" or
             call["api"] == "DsGetDcNameW":
            self.data.append({"Sample":
                              "Tried to get domain name with use of:
                               netapi32.dll!DsGetDcName"})
        elif call["api"] == "DsRoleGetPrimaryDomainInformation":
            arg_level = self.get_argument(call, "Level")
            if arg_level == "1":
                self.data.append({"Sample":
                                  "Tried to get domain name with use of:
                                  netapi32.dll!
                                  DsRoleGetPrimaryDomainInformation"})

    def on_complete(self):
        return len(self.data) > 0
```

Výpis 5.2: Signatúra kontrolujúca či vzorka zisťuje názov domény v ktorej beží

Na začiatku triedy `CheckDomain` je možné si všimnúť dátovú štruktúru set pomenovanú `filter_apinames` obsahujúcu 4 názvy API funkcií. Vďaka tejto premennej sa budú používať pre signatúru len tieto špecifikované API funkcie, čo by malo urýchliť jej porovnávanie. V metóde `on_call` sa pomocou vetvenia kontroluje či nebola nájdená jedna z týchto funkcií. V prípade, že áno tak sa pre funkciu `GetComputerNameExW` získa parameter `type`, ktorý sa následne porovnáva s dvomi hodnotami. Tieto hodnoty predstavujú názvy `ComputerNameDnsDomain` a `ComputerNamePhysicalDnsDomain`. Ak dojde k zhode, tak pravidlo pridá do premennej `self.data` informáciu o tom, že bola táto funkcia použitá s týmito špecifickými parametrami. Ďalej sa porovnávajú varianty funkcie `DsGetDcName` pre `Ascii` rovnako ako aj pre `Wide` znaky. Ak došlo k zhode, tak sa daná skutočnosť opäť zapíše do výstupneho listu `self.data`. Na koniec to je funkcia `DsRoleGetPrimaryDomainInformation`, ktorej argument `Level`

je porovnávaný s hodnotou „1“, ktorá predstavuje názov `DsRolePrimaryDomainInfoBasic` a v prípade zhody sa správa o tejto skutočnosti opäť zapíše. Po prejdení všetkých API volaní sa zavolá metóda `on_complete`, ktorá skontroluje či sa vo výstupnej premennej `self.data` nachádza nejaká hodnota. Ak sa tam nachádza, tak vráti hodnotu `True` a ak nie, tak `False`.

antisandbox_sleep_skip_detection

Nakoľko väčšina sandboxov využíva istý druh preskakovania oneskorovacích techník, resp. techník na predĺženie analýzy, tak aj tvorcovia malvéru museli na to prispôsobiť svoje výtvory. Tzn. využívajú rôzne techniky na kontrolu, či sandbox nepreskočil niektorú z funkcií využitých na predĺženie analýzy a tým sú schopní detekovať beh v sandboxe a úspešne sa vyhnuť analýze. Popísanú techniku je možné vidieť na obrázku 4.3. Tieto techniky je možné detekovať sekvenciami volaných funkcí získania aktuálneho času, funkcie pauzy, resp. oneskorenie analýzy a následne ešte jedno získanie času.

```
class Status(enum.IntEnum):
    Empty = 0
    GetTime1 = 1
    Sleep1 = 2
    GetTime2 = 3
    Sleep2 = 4
    GetTime3 = 5


class AntiSandboxSleepSkipDetection(Signature):
    checked_time_diff_twice = False
    number_of_time_checks = 0
    time_apis = [
        "GetLocalTime",
        "GetSystemTime",
        "GetTickCount",
        "GetTickCount64",
        "NtQuerySystemTime",
        "timeGetTime",
        "GetSystemTimeAsFileTime",
    ]
    sleep_apis = [
        "NtDelayExecution",
        "Beep"
    ]

    filter_process_blacklist = {
        # Office
        "excel.exe",
        "powerpnt.exe",
        "winword.exe",
        # Browsers
        "firefox.exe",
        "iexplore.exe",
        # Other
        "acrord32.exe",
        "adobearm.exe",
```

```
        "dwm.exe",
        "powershell.exe",
    }

    def __init__(self, *args, **kwargs):
        Signature.__init__(self, *args, **kwargs)
        self.last_api = Status.Empty

    def on_call(self, call, process):
        if call["api"] in self.time_apis:
            if self.last_api == Status.Empty or
               self.last_api == Status.Sleep1:
                if self.last_api == Status.Sleep1:
                    self.number_of_time_checks += 1
                self.last_api += 1
            elif self.last_api == Status.Sleep2:
                self.checked_time_diff_twice = True
                self.last_api = Status.Empty

        elif call["api"] in self.sleep_apis:
            if self.last_api == Status.GetTime1 or
               self.last_api == Status.GetTime2:
                self.last_api += 1
        else:
            self.last_api = Status.Empty

    def on_complete(self):
        if self.number_of_time_checks <= 0:
            return False

        self.data.append({"Sample":
                          "checked time skipping {} times."
                          .format(self.number_of_time_checks)})

        if self.checked_time_diff_twice:
            self.data.append({"Sample":
                              "was trying to detect sleep skipping
                               with use of 2 consecutive time
                               differences."})

        return True
```

Výpis 5.3: Signatúra na detekciu kontroly či bolo oneskorenie preskočené sandboxom

Spomínané správanie je schopné detekovať signatúra 5.3. V prípade tejto signatúry je na začiatku špecifikovaná trieda typu `Enum`, ktorá sprehľadní nasledujúci kód. Prvá premenná v hlavnej triede špecifikuje či bolo volané aspoň jedno dvojité volanie získania času. Pre lepšiu predstavu je tento konkrétny typ zobrazený obrázku 5.1.

```
firstTC = GetTickCount();
Sleep(150);
secondTC = GetTickCount();
Sleep(150);
thirdTC = GetTickCount();
if ((secondTC - firstTC) < 100 && (thirdTC - firstTC) < 200 )
    ExitProcess();
```

Obr. 5.1: Ukážka použitia oneskorovacej techniky

Následne je špecifikovaná premenná, ktorá slúži ako počítadlo počtu kontrôl aktuálneho času. Zoznam `time_apis` obsahuje všetky API funkcie využívané na získanie času alebo získanie istého časového rozdielu na systéme Windows. Ďalší list `sleep_apis` špecifikuje API funkcie používané na oneskorenie analýzy. Týchto funkcií môže byť viacero, ale tieto 2 sú najčastejšie používané s najmenším počtom falošne pozitívnych zhôd. Následne sa vyfiltrujú procesy na ktorých sa časo nachádzali falošne pozitívne zhody. Jedná sa o programy balíka office, prehliadače, Adobe Reader a ďalšie. Metóda `on_call` kontroluje sekvenciu použitých API volaní. Táto sekvencia musí pozostávať najprv zo získania času, samotného oneskorenia pomocou jednej z funkcií `NtDelayExecution` alebo `Beep` a opätovné získanie času. Taktiež je tu kontrola dvojitého volania, takže v prípade, že sa po prvých troch krokoch zavolá opäť oneskorenie a za ním bude nasledovať získanie času, tak aj toto bude detekované a premenná `checked_time_diff_twice` bude naplnená hodnotou `True`. Následne sa pri volaní `on_complete` skontroluje či boli zistené nejaké zhody s touto signatúrou a v prípade, že áno, tak sa vypíšu ich počty a signatúra vráti výslednú hodnotu ako `True`.

antisandbox_sleep_1ms_10ms

Po tom ako boli nájdené problémy v implementácii preskakovania časových oneskorení analýzy, bolo nutné nazbierať vzorky, ktoré tieto oneskorenia používajú. K tomu slúži 6 signatúr na detekciu istých časových intervalov, ktoré sú v daných vzorkách použité a vďaka tomu následne realizovať konečné vylepšenie sandboxu. Pre ilustračné účely bude vybratá len jedna, nakoľko sú tieto signatúry veľmi podobné. Zvyšné signatúry budú uložené na priloženom CD.

```
class AntiSandboxSleep1ms10ms(Signature):
    waited_counter = 0
    max_waited_counter = 15
    different_sleep_counter = 0

    def on_call(self, call, process):
        if self.waited_counter > self.max_waited_counter:
            return
```

```
        if call["api"] in self.sleep_apis:
            ms = self.get_argument(call, "Milliseconds")
            if ms is None:
                ms = -1
            ms = int(ms)

            if 0 < ms < 10:  # 1-9ms
                self.different_sleep_counter = 0
                self.waited_counter += 1
                if self.waited_counter > self.max_waited_counter:
                    self.data.append({"Sample": "waited {}x with sleep of
                                      0-9ms".format(self.waited_counter)})
                    return True
            else:
                self.different_sleep_counter += 1
                if self.different_sleep_counter >= 3:
                    self.different_sleep_counter = 0
                    self.waited_counter = 0

    def on_complete(self):
        pass
```

Výpis 5.4: Signatúra na detekciu potenciálneho oneskorovanie behu v sandboxe

Signatúra 5.4 opäť na začiatku obsahuje filtrovanie známych procesov, ktoré spôsobujú falošné zhody nad čistým softvérom. Rovnako sa tu používajú aj najčastejšie API funkcie na predĺženie analýzy. Ďalej sú špecifikované 3 premenné. Premenná `waited_counter`, ktorá ukladá počet oneskorení v časovom rozmedzí 1-9 ms. Ďalej premenná `max_waited_counter`, ktorá špecifikuje maximálny počet týchto volaní po ktorom sa vykoná istá akcia a nakoniec premenná `different_sleep_counter`, ktorá kontroluje počet volaní funkcií špecifikovaných v zozname `sleep_apis`, avšak s hodnotou času mimo interval 1-9 ms. V metóde `on_call` sa naprv kontroluje či už nebol maximálny počet vykonaných volaní dosiahnutý. Následne sa porovnáva volané API s API volaniami špecifikovanými v zozname `sleep_apis`. Ak sa podmienka splní, tak sa následne získa počet milisekúnd, ktoré predstavujú dané oneskorenie analýzy alebo v tomto prípade aj pozastavenie vykonávania samotného vzorku na špecifickú dobu. Skontroluje sa či tento parameter nie je prázdny a následne sa kontroluje či patrí do rozsahu 1-9 ms. Ak áno, tak sa navýši počítadlo úspešných volaní a vynuluje sa počítadlo volaní s časmi mimo daný časový interval. Tento postup sa opakuje až dokým počítadlo nedosiahne maximálnu hodnotu a nakoniec vráti správu s detailom, že bol vykonaný istý počet týchto volaní a signatúra následne vráti `True`. V prípade, že sa jedná o oneskorenie z iného časového intervalu, tak sa počet týchto oneskorení zaznamená a ak dosiahne počet väčší alebo rovný 3, tak sa počítadlo časových intervalov v rozsahu 1-9ms vynuluje. Slúži to na to, aby sa signatúra zameriavala len na dlhé sekvencie špecifikovaného časového intervalu.

5.2 Návrh a realizácia rezistencie sandboxu

Z predchádzajucej kapitoly rovnako ako aj z kapitoly 4 je zrejmé, že najväčší známy problém v aktuálnej Cuckoo implementácii spôsobujú anti-sandbox techniky zamerané na predlžovanie analýzy. Analýza viacerých malvérových rodín obsahujúcich tieto techniky poskytla dobrý základ na pochopenie v akých prípadoch sa tieto techniky vyskytujú a ako sú reprezentované pomocou API volaní. Vďaka tomu bolo možné navrhnúť novú teoretickú implementáciu slúžiacu na obchádzanie týchto techník. Okrem toho boli implementované ďalšie 2 anti-sandbox techniky.

Zdrojový kód ako aj presný teoretický návrh novej implementácie a rovnako aj iných pomocných skriptov nebude v práci popisovaný kvôli možnosti prípadného zneužitia. Po vyžiadaní je prístup k zdrojovým kódom novej implementácie udelený len vedúcim a oponentovi na nahliadnutie.

Anti-sandbox technika predlžujúca analýzu (NtDelayExecution)

Signatúry z kapitoly 5.1 boli napísané za účelom získať potenciálne vzorky, vďaka ktorým sa následne spomínaná teoretická implementácia nasadí a optimalizuje. Po prvotnom teste avšak bolo zreteľné, že nový návrh je rovnako efektívny ako stará implementácia a neprináša žiadne ďalšie dodatočne zlepšenia. Problémy v tomto návrhu spočívali vo veľmi špecificky nastavených hraniciach medzi jednotlivými časovými úsekmi. To znamená, že v prípade, že vzorka používala viacero oneskorení ako napr. 9 ms, 100 ms a 1,5 s tak každý jeden z týchto časov bol spracovaný inou logikou. S tým bol spojený ďalší problém. Nie len, že vďaka tomuto návrhu sandbox na podobné prípady moc nereagoval, ale ak vôbec, tak tieto oneskorenia skracoval v istom nepomere, čo spôsobovalo buď problémy behu legitimných aplikácií alebo skrátka bol pokročilejší malvér toto správanie schopný detekovať za pomoci behu na viacerých vláknach. Pre ilustráciu je toto správanie ukázané na nasledujúcom pseudo-kóde.

```
DWORD evil_thread(void *p) {
  Sleep(10*60*1000); //10 min
  do_smth_evil();
}
CreateThread(..., &evil_thread, ...);
Sleep(10*3600*1000); //10 hod
TerminateProcess(-1, 0);
```

Výpis 5.5: Ukážka viacvláknovej anti-sandbox techniky

Z tohto príkladu je zreteľné, že ak sa oneskorenia skrátia v nepomere, napríklad obe `Sleep` funkcie budú skrátené na rovnaký čas, tak sa nestihne vykonať škodlivá `do_smth_evil()` časť kódu a vďaka tomu sa tento malvér bude pre sandbox javiť

ako čistá aplikácia. Z toho dôvodu sa následnými iteráciami návrhov, implementovania a testovania došlo k finálnej logike, ktorá rozširuje súčastnú implementáciu o najčastejšie sa vyskytujúce prípady oneskorovacích anti-sandbox techník. Jedná sa primárne o volanie veľkého množstva rovnakých alebo podobných oneskorení (napr 100 000 1 ms volaní `NtDelayExecution`) alebo jednorázové volania veľkých oneskorení (napr. pozastavenia aplikácie na 5 hodín pred vykonaním jej hlavného kódu).

Anti-sandbox technika predlžujúca analýzu pomocou ICMP echo dotazu

Dotazy typu ICMP echo sa zvyknú vykonávať za pomoci príkazového riadku príkazom `ping`. Po potvrdení sa na pozadí volá aplikácia `C:\Windows\System32\PING.EXE`. V kapitole 5.1 bolo v krátkosti spomenuté, že je dôležité v tomto príkaze sledovať parametre „-n“ a „-w“ ktoré najviac ovplyvňujú dobu trvania. Na to, aby bol sandbox rezistentný voči tejto technike, je nutné ovplyvniť práve hodnoty týchto parametrov. Je hneď niekoľko spôsobov ako to docieliť. Zvolila sa avšak metóda pri ktorej je hooknutá API funkcia `CreateProcessInternalW`. Táto nedokumentovaná Windows funkcia je relatívne vysoko v kaskáde volaní API funkcií, ale dôvod prečo sa rozhodlo upravovať práve jej parametre je ten, že sa z pohľadu prípadných problémov a implementácie jedná o rozumný kompromis.

`CreateProcessInternalW` obsahuje viacero parametrov. Najdôležitejšie z nich sú `lpApplicationName` a `lpCommandLine`. Parameter `lpApplicationName` obsahuje cestu a názov spusteného súboru a parameter `lpCommandLine` obsahuje vstupné parametre daného súboru. Napríklad:

```
lpApplicationName: C:\Windows\System32\PING.EXE
lpCommandLine: ping -n 60 127.0.0.1
```

Výpis 5.6: Ukážka niektorých parametrov funkcie CreateProcessInternalW

Z vyššie popísaného vyplýva, že rozšírenie sandboxu využíva a upravuje tieto 2 parametre za istých špecifikovaných okolností a tým následne ovplyvňuje dobu trvania vykonávania `ping` príkazu.

Anti-sandbox technika hľadajúca sandbox artefakty

V kapitole 3.1 bolo priblížené, že aj samotné sandboxy majú isté artefakty, ktoré môže malvér detekovať a následne upraviť svoje správanie. Ani Cuckoo sandbox nie je v tomto prípade výnimkou a súbory využívané pre injektovanie, ako aj ďalšie súborové závislosti sa tu nachádzaju s veľmi špecifickými názvami. Jeden z týchto

súborov sa volá napríklad `cuckoomon.dll` a jednoduchá kontrola, či tento súbor existuje v istom umiestnení na disku by bola dostatočná na to, aby sa daný malvér nespustil. Z toho dôvodu je nutné tento súbor skryť. Prvý spôsob takéhoto skrytia je vyfiltrovať tento súbor z výsledkov API volaní ako napríklad `FindFirstFile`, `FindFirstFileEx`, `FindNextFile` a viacero ďalších. Jednalo by sa o dostatočne efektívny spôsob, avšak technicky náročnejší, nakoľko existuje viacero nepriamych spôsobov ako overiť existenciu súborov na istom umiestnení. Kvôli tomu sa zvolilo iné riešenie. Jednoducho sa všetky tieto súbory náhodne premenujú pred spustením každej analýzy.

Čiastočne bolo toto riešenie už implementované aj predtým, avšak pri analýze sandboxu samotného sa narazilo na to, že sa tieto súbory vôbec nepremenovávali kvôli objavenej chybe v zdrojovom kóde. Táto chyba bola odstránená a následne bolo toto premenovanie aplikované na ďalšie artefakty, vďaka ktorým mohol byť daný sandbox potenciálne detekovaný.

6 Experimentálne výsledky

Táto kapitola je venovaná dosiahnutým výsledkom po implementovaní detekčných signatúr z kapitoly 5.1 rovnako ako aj samotných vylepšení Cuckoo sandboxu v kapitole 5.2.

6.1 Výsledky signatúr

Predstavený bude počet detekcií rovnako ako aj ďalšie skutočnosti, ktoré boli vďaka týmto signatúram zistené. Aj napriek tomu, že oneskorovanie analýzy sandboxu pomocou funkcie ping nepatrí medzi najčastejšie používané techniky oneskorenia, tak signatúra `AntiSandboxSleepWithPing` za obdobie 1 mesiaca detekovala viac ako 250 000 vzoriek. Medzi takým veľkým množstvom vzoriek boli nájdené aj nové doposiaľ nepopísané malvérové rodiny.

Malvérov, ktoré kontrolovali či sa počítač resp. v tomto prípade sandbox nachádza v danej doméne s následným upravením svojho správania, alebo prípadného ukončenia bolo možné získať v časovom horizonte jedného mesiaca pomocou detekcií signatúry `CheckDomain` viac ako 886 vzoriek.

Po nasadení signatúry `AntiSandboxSleepSkipDetection` bolo zistené, že sa líšia volania `Sleep` funkcie v natívnom kóde a v p-code jazykov ako napríklad .NET. Na základe tejto znalosti bolo nutné jej dodatočné upravenie a obmedzenie prípadov falošne pozitívnych zhôd. Detekcie tejto signatúry sa teda znížili z 225 000 na 23 400.

Poslednou skupinou sú signatúry `AntiSandboxSleep`, ktoré boli vyhotovené pre 6 rôznych časových intervalov. Čo sa týka množstva dát detekovaných týmito signatúrami, tak dokopy sa jedná o viac 424 gigabajtov rôznych vzoriek za posledný mesiac. Počty detekcii v rámci jednotlivých signatúr sú zoradené od najkratších detekčných časových úsekov vyššie. Množstvo detekovaných vzoriek pre spomínané signatúry je možné vidieť v tabuľke 6.1.

Tab. 6.1: Prehľad množstva detekcií jednotlivých signatúr

Názov signatúry	Počet detekcii
`antisandbox_sleep_10ms_1s`	5 584
`antisandbox_sleep_10s_30s`	1 869
`antisandbox_sleep_1ms_10ms`	35 642
`antisandbox_sleep_1s_5s`	765
`antisandbox_sleep_30s_1w`	170
`antisandbox_sleep_5s_10s`	206 258

Samozrejme, nejedná sa len o vzorky malvéru, ale istá časť patrí aj čistému softvéru. Avšak tieto dáta budú slúžiť ako referencia a testovacia dátová množina pre testovanie novej rezistenčnej Cuckoo implementácie.

6.2 Výsledky rezistenčnej implementácie

Všetky výsledky sú testované na vzorkách z dňa 01.05.2021. Dôvod prečo je tomu tak a nevyužili sa vzorky zo starších dátumov je ten, že staršie vzorky zvyknú veľakrát prestať fungovať lebo server na ktorý sa pripájajú alebo odkiaľ sťahujú payload už môže byť nedostupný. Pri prvotnom testovaní sa narazilo na takýto prípad práve pri malvére NanoCore.

Anti-sandbox technika predlžujúca analýzu (NtDelayExecution)

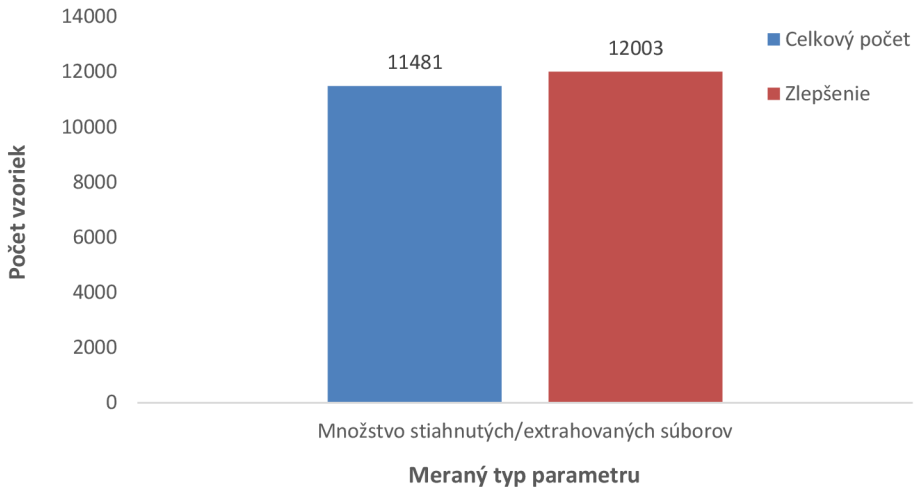
Nakoľko nová sleep-skipping implementácia predstavuje väčší zásah do sandboxu a ovplyvní všetky vzorky využívajúce funkciu NtDelayExecution, tak po jej implementovaní bolo nutné túto logiku aj patrične otestovať. Testovanie nespočíva len v overení detekcie nových vzoriek, ale aj v overení, či sa náhodou analýzy neukončujú na chybách. Následne sa musí taktiež vykonať aj overenie, že sa staré detekcie nezhoršili. Z toho dôvodu bolo pre retrospektívne testovanie vybratých 1 000 čistých najčastejšie sa vyskytujúcich vzoriek, ktoré využívajú funkciu `NtDelayExecution`.

Na prvotné overenie toho či sa nezhoršili staršie detekcie a zároveň či sa zvýšil počet novo-detekovaných vzoriek boli využité vzorky nazbierané na základe signatúr, avšak na finálne overenie bolo nutné viac špecifikovať finálnu množinu vzoriek. Boli vybrané vzorky, ktoré sa na základe istej heuristiky posielajú na reanalýzu s dlhším časovým intervalom na to, aby bez zásahu do logiky aplikácie skrátka vyčkali dlhšiu dobu na škodlivé správanie potenciálne nebezpečného vzorku. Jednalo sa zhruba o 20 000 súborov z ktorých boli vyfiltrované také, v ktorých boli detekované file-infektory. Dôvod filtrácie súborov obsahujúcich file-infektory je ten, že tvorili takmer 40% spomínaného prefiltrovaného množstva vzoriek a z prechádzajúcich zistení sme nepozorovali žiadne významné zlepšenia pri testovaní na novej implementácii. Následne boli odfiltrované vzorky, ktoré mali len čisto textovú formu (ostali teda primárne PE a archívy). Po tomto vyfiltrovaní sa počet vzoriek znížil na 9 314 jedinečných súborov.

Následne bol zoznam hashov týchto vzoriek vložený do porovnávacieho skriptu, ktorý bol vytvorený ako vedľajší produkt tejto práce. Funkcia tohto skriptu spočívala v tom, že sa vzal zoznam SHA-256 záznamov a tieto záznamy sa následne posielali na jeden z produkčných serverov vyhradených na účely tejto práce. Každá vzorka sa analyzuje dvakrát. Raz v pôvodnej a raz v upravenej implementácii. Následne

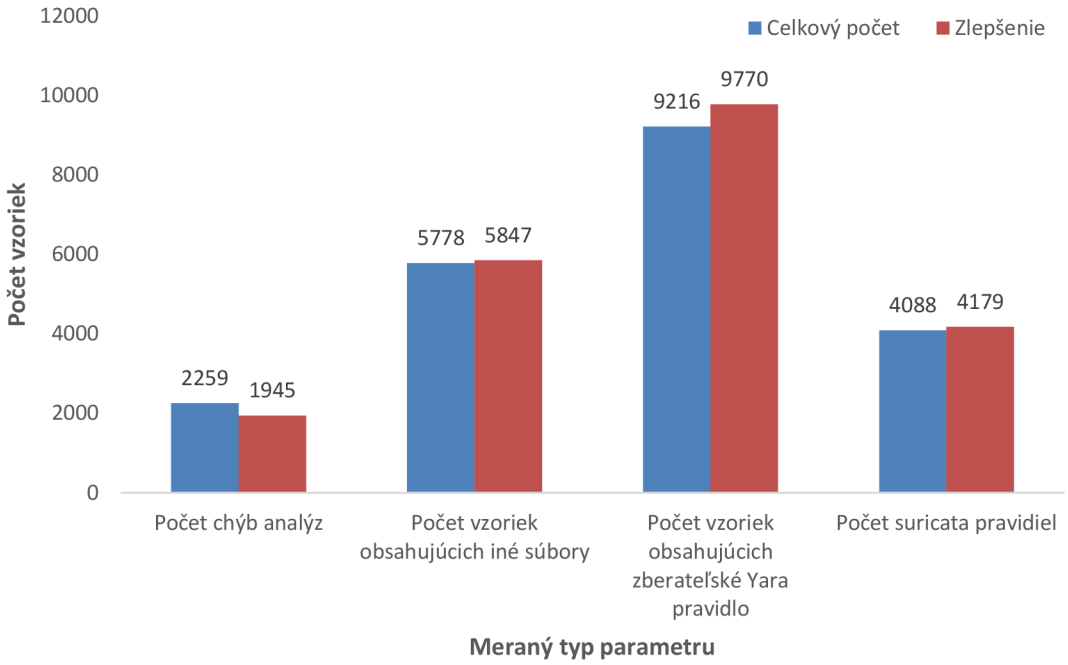
sa stiahnu výsledky analýz, vymažú sa z nich nepotrebné údaje (použitie stratovej kompresie bolo nutné kvôli potenciálnemu zaplneniu disku pri väčšom množstve analyzovaných vzoriek) a nakoniec sa porovnajú výsledky. Čo sa porovnávania týka, sledujú sa nasledujúce parametre: doba analýzy, veľkosť výstupu analýzy, náhle ukončenia analýz alebo vzoriek samotných počas behu v sandboxe, počet API volaní, počet API volaní `NtDelayExecution`, počet súborov stiahnutých alebo inak extrahovaných z pôvodnej vzorky (tzv. droppy) a nakoniec počet zhôd Yara pravidiel a Suricata pravidiel. Server, na ktorom je sandbox nasadený spracovává vzorky rýchlosťou zhruba 2 000 analýz za hodinu. Tým, že sa každá vzorka testuje dvakrát, tak sa jedná o spracovanie 1 000 jedinečných vzoriek za hodinu. Server obsahuje 4 kontainery, z ktorých každý obsahuje 24 virtuálnych počítačov. Zo spomínaných kontajnerov 2 slúžia na testovanie pôvodnej implementácie a 2 na testovanie nových vylepšení. Výsledky týchto dvoch inštancií sa následne porovnávajú. Dôvod prečo sa to robí na 1 serveri je minimalizácia prípadných odlišností vo výsledkoch. Pre minimalizáciu odlišností v analýzach sa v rámci tejto práce testovali vzorky len na 64 bitových virtuálnych počítačoch.

Najdôležitejšia metrika pri testovaní 1 000 čistých vzoriek bol počet náhlych ukončení analýz alebo vzoriek samotných. Výsledky ukázali, že nedošlo k zhoršeniu a okrem toho sa dokonca zlepšil počet súborov stiahnutých alebo extrahovaných z týchto vzoriek (tzv. droppy). Obrázok 6.1 zobrazuje toto zlepšenie počtu droppov o 522, čo predstavuje nárast o 4,5%. Týmto testom bolo overené, že nová implementácia anti-sandbox techniky nezhoršuje klasické čisté detekcie a práve naopak, dokonca zlepšuje jednu meranú metriku.



Obr. 6.1: Zlepšenie počtu extrahovaných alebo stiahnutých súborov z čistých analyzovaných vzoriek

Čo sa týka testovania 9 314 potenciálne nebezpečných vzoriek, tak pri testovaní došlo k zlepšeniu všetkých meraných parametrov, z ktorých tie s vyšším prínosom sú ukázané na obrázku 6.2. V prvom rade sa znížil počet analýz, u ktorých bolo vynútené ukončenie. Zníženie bolo o 13,9% čo vypovedá o tom, že viacero vzoriek bolo schopných dobehnúť do konca a sandbox nebol nútený ich násilne ukončiť. Čo sa týka chýb vzoriek počas analýzy, tak tu nastalo zlepšenie o 17,24%. V prípade počtu vzoriek, ktoré obsahujú iné súbory (vyšší počet stiahnutých alebo extrahovaných súborov) bolo preukázané nízke zlepšenie a to o 1,19%. Vrámci malvérových detekcií sa pôvodných 75 detekcií zvýšilo na 85, čo predstavuje 13% zlepšenie. Zastúpenie jednotlivých novo-detekovaných malvérových rodín je možné vidieť v tabuľke 6.2. Okrem toho bol aj zvýšený počet Yara pravidiel slúžiacich na zbieranie vzoriek na základe špecifických vlastností. Tu pribudlo 4 077 detekcií týchto pravidiel, čo predstavuje 4,03% nárast. Nakoniec bol ešte počet detekcií Suricata pravidiel navýšený o 91.



Obr. 6.2: Značné zlepšenia spôsobené novou implementáciou

Zo všetkých denne analyzovaných vzoriek sa jedná len o malú časť u ktorých bola zlepšená detekcia Yara pravidiel. Je tomu tak lebo už aj súčastná sandbox implementácia mala dostatočne dobrú rezistenciu a nová implementácia pridávala rozšírenie primárne na volanie veľkého množstva oneskorení za sebou alebo jednorázové volanie oneskorení väčších ako 1 hodina. Okrem toho sa predpokladá, že detekcie vďaka tomuto vylepšeniu ešte viac vzrastú. Dôvodom je práve zvýšenie množstva

Tab. 6.2: Zastúpenie novo detekovaných malvérových rodín

Názov rodiny	Typ malvéru	Počet nových detekcii
`Yesterday`	Password stealer	2
`Rodecap`	Trojan	2
`WizzMonet`	Adware	2
`NanoCore`	Remote Access Trojan	1
`MeanTeamViewer`	Backdoor	1
`Palevo`	Worm	1
`QuasarRAT`	Remote Access Trojan	1

extrahovaných vzoriek. Nakoľko tieto nové dáta zlepšia šancu na odhalenie nových malvérových rodín, ktoré ešte neboli pokryté a následne budú okamžite detekované len vďaka tomuto rozšíreniu.

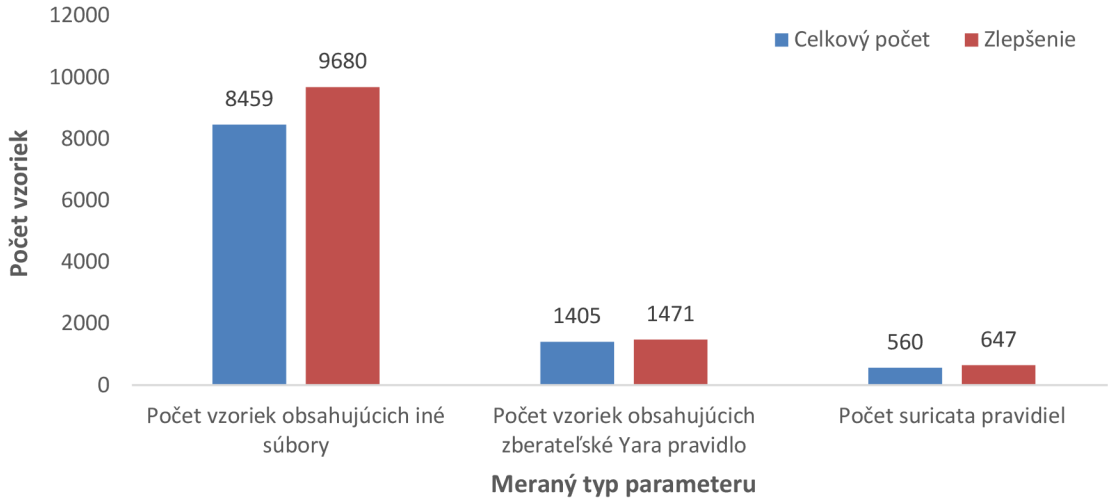
Okrem toho sa v pôvodnej implementácii našla chyba, ktorá mohla spôsobovať neočakávané správanie v istých prípadoch behu sandboxu. Jednalo sa o časť kódu, ktorá pochádzala ešte z oficiálnej implementácie Cuckoo sandboxu. Jej účel bol v prvých 5 sekundách behu aplikácie preskočiť všetky oneskorenia. Táto logika záležala na rozdiele času od začiatku spustenia sandboxu na základe čoho sa kontrolovalo prvých 5 sekúnd. Problém bol v tom, že tento časový rozdiel bol vo väčšine prípadov vždy väčší ako 5 lebo bol získavaný pred tým, než sa nastavoval náhodný čas sandboxu. Tým pádom táto logika fungovala len v malom množstve prípadov. Avšak aj toto malé množstvo prípadov bolo nežiadúce, pretože s použitím tohto 5 sekundového preskakovania oneskorení veľa vzoriek prestalo správne fungovať. Po zistení tejto skutočnosti sa do druhého dňa nasadila patričná oprava, ktorá túto časť kódu odstraňuje.

Anti-sandbox technika predlžujúca analýzu pomocou ICMP echo dotazu

Spôsob získania vzoriek pre overenie novej implementácie bol v tomto prípade získaný čisto len z Avast databázi. V prvom rade sa vyfiltrovali zo všetkých analyzovaných vzoriek z 01.05. všetky také, ktoré spúštajú príkaz (zvyčajne sa jednalo o konzolové príkazy). V rámci tohto dňa sa jednalo o 323 223 vzoriek, z toho bolo 2 806 takých, ktoré využívali príkaz `ping` s jedným zo spomínaných parametrov „-n“ alebo „-w“.

Pri tomto vylepšení boli merané rovnaké metriky ako pri predchádzajúcej rezistenčnej technike. V tomto prípade nastal najväčší rozdiel pri množstve extrahovaných súborov, ktoré sa pri týchto 1 406 vzorkách zvýšilo z 8 459 na 9 680, čo je 14% zlepšenie.

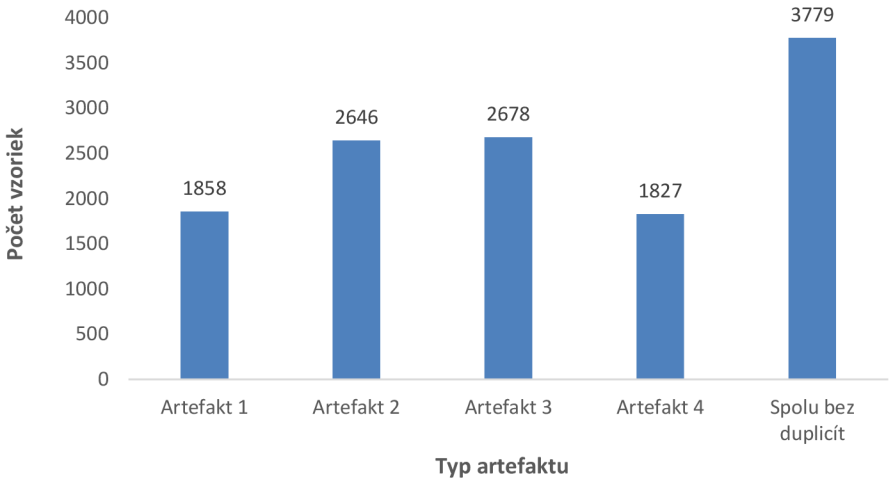
Dôvod tohto výsledku je ten, že skrátka vzorky nečakali tak dlhú dobu na oneskorení, ale vykonali počas behu analýzy viac akcií a prípadných alokácií pamäte (ktoré sandbox extrahoval) alebo extrakcií iných súborov a pamäťových oblastí (bufferov). Ďalej sa počet pomocných Yara pravidiel zlepšil o 4,3% a počet Suricata pravidiel vzrástol o 15,54%. Spomínané štatistiky je možné vidieť na grafe 6.3.



Obr. 6.3: Zlepšenia po implementovaní ping rezistencie

Anti-sandbox technika hľadajúca sandbox artefakty

Rovnako ako pri anti-sandbox technike využívajúcej ICMP echo dotazy, tak aj v prípade artefaktov boli získané vzorky obdobným spôsobom z databázy. V prvom rade sa vyfiltrovali všetky súbory, ktoré pristupovali počas behu k iným súborom na disku (496 974 vzoriek). Z toho sa následne vyfiltrovali 4 kategórie. Každá kategória predstavovala jeden artefakt. Zastúpenie jednotlivých artefaktov a ich agregácia bez duplicít je zobrazená na nasledujúcom grafe 6.4



Obr. 6.4: Zastúpenie jednotlivých artefaktov

Vzorky sa po odstránení duplicít následne vložili do porovnávacieho skriptu. Pri testovaní tohto vylepšenia nedošlo k zmene výsledkov. Dôvod prečo vzorky, ktoré k týmto artefaktom pristupovali a ich premenovanie nespôsobilo žiadnu zmenu je ten, že vzorky obsahovali primárne malvérový typ file-infektor, ktorý zvyčajne skenuje celý disk a infikuje ostatné súbory, čo bol aj aktuálny prípad. Z toho dôvodu je implementácia tejto poslednej rezistenčnej techniky primárne prevenčná. Samozrejme to, že sa 01.05.2021 nevyskytol ani jeden súbor, ktorý by tieto artefakty úmyselne hľadal, neznamená, že táto technika nie je používaná, len je skrátka veľmi ojedinelá.

Záver

V rámci práce bola naštudovaná problematika anti-sandbox techník, rovnako ako aj súčasná implementácia hlavného Cuckoo sandboxu spoločnosti Avast Software. Čitateľovi boli priblížené spôsoby statickej a dynamickej analýzy a okrem iného aj funkcionalita a účel používania sandboxu. Následne bolo popísané rozdelenie kategórií anti-sandbox techník, ktoré sa vyskytujú v pokročilom malvéri. V kapitole 4 bola za pomoci reverzného inžinierstva popísaná hĺbková analýza 5 malvérových rodín, z ktorých každá rodina obsahovala aspoň 1 anti-sandbox techniku. Táto analýza dala čitateľovi pohľad na spôsob analýzy malvéru. Najdôležitejšou časťou je práve kapitola 5, v ktorej boli opísané 2 spôsoby vylepšenia aktuálnej Cuckoo sandbox implementácie.

Prvým spôsobom je detekcia techník slúžiacich primárne na obchádzanie sandbox analýz. Tento spôsob bol implementovaný pomocou signatúr, ktoré sú schopné viaceré také techniky detekovať. Vďaka signatúram bolo možné získať veľké množstvo vzoriek využívajúcich spomínané techniky a vďaka tomu boli aj objavené nové malvérové rodiny, ktoré doteraz neboli popísané YARA pravidlami.

Druhý a najdôležitejší spôsob je návrh a realizácia obrany sandboxu (rezistencie) voči týmto technikám. Realizácia spočívala zo zmeny aktuálnej logiky Cuckoo sandboxu na obranu voči technikám oneskorenia pomocou volania funkcie `NtDelayExecution`, volania príkazu `ping` na oneskorenie analýzy alebo vyhľadnávanie súborov patriacich sandboxu. Po nasadení a patričnom otestovaní novej rezistenčnej implementácie pre spomínané anti-sandbox techniky došlo v analýzach k viacerým zlepšeniam. Jedná sa o väčšie množstvo extrahovaných súborov alebo dokonca aj zvýšenie detekovaného malvéru. Okrem toho sa vďaka tejto práci našli a opravili 2 programové chyby v starej Cuckoo implementácii. Jedna z nich mohla pri oneskorovaní analýz spôsobovať neočakávané správanie a tá ďalšia spôsobovala, že súbory sandboxu boli jednoducho dohľadateľné prehliadaním súborov na disku. Samozrejme má táto nová implementácia a oprava programových chýb priamy dopad aj na koncových užívateľov. Nie len, že sa zníži počet súborov, ktoré je nutné opätovne reanalyzovať a tým urýchliť propagáciu detekcií, ale taktiež sa zvýši počet detekcií samotných, vďaka čomu budú užívatelia lepšie chránení.

Nakoľko sa jedná o komplexnú a širokú problematiku, tak je možné nadviazať na túto prácu ďalšími iteráciami implementovania nových rezistenčných metód voči iným anti-sandbox technikám a tým pomôcť viac ako 435 miliónom užívateľov, ktorí využívajú antivírus Avast.

Literatúra

[1] SIKORSKI, Michael a Andrew HONIG. *Practical malware analysis: the hands-on guide to dissecting malicious software.* Upper Saddle River, NJ: Addison-Wesley, 2005. ISBN 0321304543.

[2] EILAM, Eldad. *Reversing: Secrets of Reverse Engineering.* New Jersey: Wiley, 2005. ISBN 0764574817.

[3] DANG, Bruce, Alexandre GAZET, Elias BACHAALANY a Sebastien JOSSE. *Practical reverse engineering: x86, x64, ARM, Windows Kernel, reversing tools, and obfuscation.* Indianapolis, Indiana: Wiley, [2014].

[4] MIČKA, Richard. *Automatická detekce použité kryptografie v kódu [online].* Brno, 2019 [cit. 2020-12-01]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/118107?zp_id=118107. Bakalárska práca. Vysoké Učení Technické V Brne. Vedoucí práce Doc. Ing. Jan Hajný, Ph.D.

[5] *User mode and kernel mode. Microsoft [online].* Redmond, Washington [cit. 2020-12-01]. Dostupné z: https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode

[6] *Kernel mode hooks or user mode hooks - What's best for firewall ? Yumpu - Publishing digital magazines worldwide [online].* [cit. 2020-12-01]. Dostupné z: https://www.yumpu.com/en/document/read/32980568/kernel-mode-hooks-or-user-mode-hooks-whats-best-for-agnitum

[7] *Hypervisor. VMware - Delivering a Digital Foundation For Businesses [online].* Palo Alto, California [cit. 2020-12-01]. Dostupné z: https://www.vmware.com/topics/glossary/content/hypervisor

[8] KŘOUSTEK, Jakub. *Rekonfigurovatelná analýza strojového kódu [online].* Brno, 2015 [cit. 2020-12-01]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/99825?zp_id=99825. Dizertačná práca. Vysoké Učení Technické V Brne. Vedoucí práce Doc. Dr. Ing. Dušan Kolář.

[9] *A Modern Hypervisor as a Basis for a Sandbox. Securelist | Kaspersky's cyberthreat research and reports [online].* Moskva, Rusko [cit. 2020-12-01]. Dostupné z: https://securelist.com/a-modern-hypervisor-as-a-basis-for-a-sandbox/81902/

[10] *Cuckoo Sandbox - Automated Malware Analysis [online].* Stichting Cuckoo Foundation [cit. 2020-11-30]. Dostupné z: https://cuckoosandbox.org/

[11] *What is Cuckoo? Cuckoo Sandbox Book [online].* Berlín, Nemecko: Cuckoo Foundation [cit. 2020-11-30]. Dostupné z: https://cuckoo.readthedocs.io/en/latest/introduction/what/

[12] *DLL INJECTION. Multimedia codecs GStreamer | Fluendo [online].* Barcelona, Španielsko [cit. 2020-12-01]. Dostupné z: https://fluendo.com/en/blog/post/dll-injection-coding/

[13] *EMOTET: A TECHNICAL ANALYSIS OF THE DESTRUCTIVE, POLYMORPHIC MALWARE. Bromium [online].* Cupertino, California [cit. 2020-11-30]. Dostupné z: https://www.bromium.com/wp-content/uploads/2019/07/Bromium-Emotet-Technical-Analysis-Report.pdf

[14] *Semaphore Objects. Microsoft [online].* Redmond, Washington [cit. 2020-11-30]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/sync/semaphore-objects

[15] *NJRat - NJCCIC Threat Profile. NJCCIC [online].* New Jersey [cit. 2020-11-30]. Dostupné z: https://www.cyber.nj.gov/threat-center/threat-profiles/trojan-variants/njrat

[16] *Cynet XRD | Autonomous Breach Protection [online].* New York [cit. 2020-11-30]. Dostupné z: https://www.cynet.com/attack-techniques-hands-on/njrat-report-bladabindi/

[17] *Trojan.APT.BaneChant. Cyber Security Experts & Solution Providers / FireEye [online].* Milpitas, California [cit. 2020-11-30]. Dostupné z: https://www.fireeye.com/blog/threat-research/2013/04/trojan-apt-banechant-in-memory-trojan-that-observes-for-multiple-mouse-clicks.html

[18] *NanoCore Malware Information. Trend Micro (UK) / Enterprise Cybersecurity Solutions [online].* Irving, Texas [cit. 2020-11-30]. Dostupné z: https://success.trendmicro.com/solution/1122912-nanocore-malware-information

[19] *NanoCore RAT — Malware of the Month. Cloud-to-Cloud SaaS Backup [online].* Austin Texas [cit. 2020-11-30]. Dostupné z: https://spanning.com/blog/nanocore-rat-malware-of-the-month/

[20] *What is Dridex malware ? Spambrella – Email Security & Awareness Services [online].* Kent, UK [cit. 2020-11-30]. Dostupné z: https://www.spambrella.com/what-is-dridex-malware/

[21] *Dridex Malware. CISA [online].* Washington, DC [cit. 2020-11-30]. Dostupné z: https://us-cert.cisa.gov/ncas/alerts/aa19-339a

[22] *Signatures. Cuckoo Sandbox Book [online].* Berlín, Nemecko: Cuckoo Foundation [cit. 2020-11-30]. Dostupné z: https://cuckoo.readthedocs.io/en/latest/customization/signatures/

[23] ARKKO, Jari, Michelle COTTON a Leo VEGODA. *IPv4 Address Blocks Reserved for Documentation.* Internet Engineering Task Force, 2010. ISSN 2070-1721.

Zoznam symbolov, veličín a skratiek

API	aplikačné programové rozhranie – Application programming interface
BSOD	modrá smrť – Blue screen of death
BSON	binárny JSON – Binary JSON
C2	velenie a riadenie – Command and Control
CD	kompaktný disk – Compact Disc
DIE	Detect It Easy
GNU	všeobecná verejná licencia – General Public License
GUI	grafické používateľské rozhranie – Graphical user interface
HTTP	hypertextový prenosový protokol – Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDA	interaktívny disassembler – Interactive Disassembler
IL	medzijazyk – Intermediate language
IP	internetový protokol – Internet Protocol
JIT	dynamický preklad – Just in time
JSON	JavsScriptovo orientovaná notácia – JavaScript Object Notation
KM	režim jadra – Kernel mode
MD5	Message-Digest algorithm 5
MITB	muž v prehliadači – Man in the browser
OS	operačný systém – Operating System
P2P	Peer-to-peer
p-code	prenositeľný kód – Portable code
PDF	Prenositeľný Dokumentový Formát – Portable Document Format
PE	profesionálny – Portable Executable
PRO	profesionálny – Professional

RAT	trojan na vzdialený prístup – Remote Access Trojan
RD	vzdialené ladenie – Remote debugging
SHA	zabezpečený hash algoritmus - Secure Hash Algorithm
SSL	zabezpečená soketová vrstva – Secure Sockets Layer
TCP	protokol riadenia prenosu – Transmission Control Protocol
TLS	zabezpečenie transpostnej vrstvy – Transport Layer Security
UM	užívateľský režim – User mode
VM	virtuálny stroj – Virtual machine
VMM	monitor virtuálnych strojov – Virtual machine monitor
VP	virtuálne prostredie

Zoznam príloh

A Obsah přiloženého CD

Príloha umiestnená na CD nosiči obsahuje vzorky 5 analyzovaných malvérov zabalených v .zip archíve s heslom „infected“ a taktiež zložku „Signatures“ obsahujúcu nové detekčné signatúry.