SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-73886

MODERNÉ TECHNIKY NA UKRÝVANIE ČINNOSTI MALVÉRU

BAKALÁRSKA PRÁCA

2020 **Lukáš Gnip**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-73886

MODERNÉ TECHNIKY NA UKRÝVANIE ČINNOSTI MALVÉRU

BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Peter Švec

Bratislava 2020 **Lukáš Gnip**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020
Evidenčné číslo: FEI-5382-73886

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Lukáš Gnip**
ID študenta:	73886
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Peter Švec
Miesto vypracovania:	Ústav informatiky a matematiky

Názov práce: **Moderné techniky na ukrývanie činnosti malvéru**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

V súčasnosti väčšina malvéru používa sofistikované techniky voči detekcii antivírusovým programom a sťaženiu analýzy škodlivého kódu. Medzi takéto techniky patrí využitie rôznych foriem packerov, ukrývanie škodlivých procesov, detekcia behu vo virtuálnom prostredí a pod. Cieľom práce je analyzovať techniky, ktoré využívajú v súčasnosti najrozšírenejšie druhy malvéru a implementovať ukážkovú aplikáciu na detekciu vybranej metódy.
Úlohy:
1. Naštudujte techniky používané súčasným malvérom.
2. Implementujte aplikáciu na detekciu vybranej techniky.
3. Zhodnoťte implementáciu a porovnajte s existujúcimi riešeniami.

Zoznam odbornej literatúry:

1. Sikorski, M. – Honig, A. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.* San Francisco: No Starch Press, 2012. 800 s. ISBN 1-59327-290-1.
2. Scholte, T. – Egele, M. – Kruegel, C. A Survey on Automated Dynamic Malware Analysis Techniques and Tools. [online]. 2011. URL: https://www.iseclab.org/papers/malware_survey.pdf.

Riešenie zadania práce od: 23. 09. 2019

Dátum odovzdania práce: 01. 06. 2020

Lukáš Gnip
študent

Dr. rer. nat. Martin Drozda
vedúci pracoviska

prof. Dr. Ing. Miloš Oravec
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Lukáš Gnip
Bakalárska práca:	Moderné techniky na ukrývanie činnosti malvéru
Vedúci záverečnej práce:	Ing. Peter Švec
Miesto a rok predloženia práce:	Bratislava 2020

abstrakt

Kľúčové slová: k1, k2, k3

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Lukáš Gnip
Bachelor Thesis:	Modern Hiding Techniques in Malware
Supervisor:	Ing. Peter Švec
Place and year of submission:	Bratislava 2020

abstact

Keywords: k1, k2, k3

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

DLL - Dynamic Link Library
EWM - Extra Windows Memory
FTP - File Transfer Protocol
APC - Asynchronous Procedure Call
CnC - Command and Control Center
BOT - RoBOT
API - Application programming interface
PEB - Process environment block
VAD - Virtual address descriptor

Zoznam algoritmov

Úvod

Poznámka: písať až úplne na záver
Úvod sa skladá z troch odsekov:

Všeobecný úvod do problematiky, spomenúť prečo je dôležité zaoberať sa malvérom, koľko nových vzoriek vzniká denne oproti minulosti a pod.

Konkrétne čím sme sa zaoberali v našej práci.

Popis členenia práce, t.j. v Kapitole X sa nachádza popis Y.

Tip: prácu treba písať v prvej osobe množného čísla, čiže nie "som popísal", "implementoval som", "rozhodol som sa použiť prostredie X" ale "popísali sme", "implementovali sme" a "rozhodli sme sa použiť prostredie X" a pod...

1 Malvér

Je to softvér, ktorého cieľom je poškodiť, zablokovať, zmocniť sa alebo odcudziť citlivé informácie uložené v počítači. Cieľom malvéru je získať informácie pre útočníka a následné zneužitie týchto informácií na rôzne druhy nelegálnej činnosti za účelom danej obeti uškodiť alebo sa na nej finančne obohatiť. Malvér sa kedysi členil na rôzne kategórie ako napr. vírusy, *backdoor*, *spyware*, *ramsomware* a pod.[**?**] V súčasnosti už toto delenie nie je veľmi aktuálne, pretože malvér v dnešnej dobe je už viacmenej kombináciou týchto kategórií a využíva rôzne komponenty z jednotlivých druhov škodlivého kódu. V nasledujúcej kapitole sa podrobnejšie zaoberáme rôznymi spôsobmi ukrývania činnosti a prítomnosti malvéru, ktoré môžu byť v súčasnosti využívané.

1.1 Techniky ukrývania činnosti

Malvér vo všeobecnosti patrí ku škodlivému kódu. Preto je nutné jeho bežiace procesy utajiť. Ak chce byť malvér úspešný v získavaní údajov alebo finančných prostriedkov, musí byť jeho beh ukrytý pred potenciálnym antivírusom, prípadne forenzným inžinierom ktorý je schopný ho odhaliť. Za týmto účelom sa využívajú rôzne techniky na ukrytie malvéru buď v pamäti počítača, rôznych shell scriptoch alebo dynamických knižniciach, aby boli čo najmenej detegovateľné.

- **DLL Injection / Reflective DLL Injection**
  DLL Injection je technika používaná na ukrytie funkcie na spustenie malvéru vo vnútri iného programu. Najžastejšie sa na to využíva alokovaná pamäť vyhradená pre beh infikovaného programu, kde sa funkcia na spustenie malvéru ukryje a pre antivírusové programy je ťažko detekovateľná. Po nakopírovaní DLL funkcie do alokovanej pamäte iného programu môže byť následne vyvolané spustenie infikovaného programu a spolu s ním aj spustenie malvéru. Hlavným cieľom *DLL Injection* je škodlivý kód ukryť do oficiálneho alebo overeného programu, kde je malvér ukrytý pred antivírovým softvérom, odkiaľ môže byť následne spustený. [**?**]

- **Proces Hollowing**
  využíva rovnaké alebo podobné princípy ukrývania škodlivého softvéru ako *DLL Injection*. Cieľom je schovať škodlivý kód do existujúce programu z ktorého sa po spustení, spustí aj volanie škodlivého malvéru. Oproti *DLL Injection* ide ale ukrytie malvérového programu do iného programu. Malvér si podľa potreby alokuje virtuálnu pamäť v pamäti iného programu. Po spustení infikovaného programu malvér

pozastaví vlákno v ktorom program beží, následne zmení obsah legitímneho súboru zmapovaním pamäte cieľového procesu.[**?**] Po zmapovaní uvoľní všetku pamäť programu a alokuje pamäť pre malvér a zapíše každú z častí malvéru do cieľovej pamäte programu. Malvér nastaví kontext vo vlátkne tak aby ukazoval vstupný bod na novú časť kódu, ktorú napísal. Na konci malvér obnoví pozastavené vlákno, aby sa proces dostal z pozastaveného stavu a nasledovným spustením programu umožní získavanie údajov.

- **Thread Execution Hijacking**
  Táto technika ukrytia malvéru spočíva v napojení sa na už existujúce vlákno vyvolaný iným programom. Po získaní prístupu do vlákna malvér uvedie vlákno do pozastaveného režimu aby vykonal vloženie volania škodlivého malvéru do vlákna iného procesu. Malvér si alokuje virtuálnu pamäť v programe a následne do tejto pamäte vloží škodlivý shell kód, ktorý obsahuje cestu k volaniu DLL so škodlivým malvérom. Po vykonaní týchto úkonov, spustí pozastavené vlákno programu. Nevýhodou takéhoto spustenia pozastaveného programu je, že môže spôsobiť zlyhanie systému v rámci systémového volania. [**?**] Preto aby sa tomu predišlo modernejší malvér proces ukončí a vyvolá ho znovu s už modifikovanou zmenou na infikovanie.

- **Portable Executable Injection**
  Výhodou tohto spôsobu ukrytia malvéru je využitie nakopírovania malvéru do už existujúceho bežiaceho procesu pomocou shell skriptu[**?**], ktorý vyvolá spustenie škodlivého malvéru. Namiesto toho aby proces prepisoval cesty volaním DLL, táto technika zapisuje obsah malvéru priamo do pamäte. Počas doby zapisovania aby malvér nebol ľahko odhalený využíva vnorené cykly, ktoré spomaľujú systém buď diagnostikou alebo volaním zbytočných funkcií a snaží sa zahltiť systém a neumožniť skorú diagnostiku malvéru. Keď malvér preformuluje všetky potrebné adresy, všetko, čo musí urobiť, je odovzdať jeho počiatočnú adresu vláknu a nechať spustiť malvér.

- **Hook injection**
  Hook injection je technika používaná na zachytávanie volania funkcií. Malvér môže využívať hook injection funkcie na načítanie škodlivého súboru v DLL pri spustení udalosti v konkrétnom vlákne. Malvér na to využíva volanie funkcie, ktorý obsahuje štyri parametre.[**?**] Prvý je tip udalosti na, ktorý sa spustí škodlivý malvér napríklad na stlačenie klávesnice alebo tlačidla na myši. Druhý je ukazovateľ na funkciu, ktorá sa ma po stlačení daného tlačidla vykonať. Tretí je modul obsahujúci danú funkciu.

Preto je pred volaním funkcie, vidieť volania do LoadLibrary. Posledný parameter je vlákno ktoré má vykonať procedúru a hook injection. Pokiaľ je posledný parameter prázdny alebo nastavený na nulu tak danú procedúru vykonajú všetky bežiace vlákna. Malvér sa ale zameriava len na jedno vlákno, kvôli zníženiu detekcii svojej práce.

- **APC Injection**
  Škodlivý softvér môže využívať výhody asynchrónnych volaní procedúr (APC), aby prinútil ďalšie vlákno spustiť svoj vlastný kód jeho pripojením do fronty cieľového vlákna.[**?**] Každé vlákno má rad asynchrónnych volaní procedúr, ktoré čakajú na vykonanie, keď cieľové vlákno vstupuje do zmeniteľného stavu. Vlákno vstúpi do výstražného stavu. Malvér zvyčajne hľadá akékoľvek vlákno, ktoré je v zmeniteľnom stave, aby zaradil APC do vlákna. Čo umožňuje jeho následným spustením infikovať zariadenie malvérom.

- **Extra windows memory injection**
  Tento spôsob schovávania softvéru sa spolieha na rozširovanie pamäte aplikačných okien v systéme. Pri otváraní nového okna aplikácie, softvér špecifikuje ďalšie bajty pamäte, ktoré rozšíria veľkosť alokovanej pamäte pre spustenú aplikáciu.[**?**] Tento proces sa nazýva *Extra Windows Memory*(EWM). V tejto časti ale nevzniká dostatok miesta na uloženie údajov. Aby sa toto obmedzenie obišlo, škodlivý softvér zapíše kód do zdieľanej sekcie aplikácie a do EWM vloží ukazovateľ na danú časť. Do tejto rozšírenej časti zieľanej pamäte ďalej softvér zapíše ukazovateľ funkcie do pamäte, ktorá obsahuje shell skript na spustenie malvéru. Malvér následne nastaví v EWM fukciu na zmenu hodnôt na zadanom ofsete. Čím malvér môže jednoducho zmeniť posun ukazovateľa funckie pamäti aplikácie a nasmerovať ho do EWM, na kód so škodlivým shell skriptom.

1.2 Súčasný malvér

Táto kapitola obsahuje opis jednotlivých malvérov používaných v roku 2019, ktoré boli detekované spoločnosťami ako Avast a McAfee. Tieto malvéry sú najčastejšie využívané v oblasti Európy. Kapitola obsahuje bližší opis malvérov, ich využitie, použité spôsoby úrokov a ukrytie malvéru v systéme.

- **Sodinokibi**

  Tento malvér bol detekovaný v období okolo apríla 2019. Patrí do rodiny ransomvéru, ktorých cieľom je šifrovať informácie v zariadení a následne za dešifrovanie pýta nemalý obnos peňazí.[**?**] Názov bol objavený v hash kóde, ktorý obsahoval názov "Sodinokibi.exe". vírus sa šíri sám zmeužívaním zraniteľnosti na serveroch Oracle WebLogic. Softvér je navrhnutý tak, aby rýchlo vykonával šifrovanie definovaných súborov v konfigurácii ransomvéru. Prvou akciou škodlivého softvéru je získať vtetky funkcie potrebné pri behu programu a vytvoriť dynamický IAT, ktorý sa pokúsi zahltiť volanie systému Windows statickou analýzou. Po zahltení systému dôjde k spusteniu malvéru. Technika využívaná na ukrytie malvéru je Portable Executable Injection ktorú volá po spomalení RunPE na jej spustenie z pamäte. táto technika ukrytia malvéru je opísaná v predošlej kapitole. Analýza spoločnosti McAfee ukazuje podobnosť s iným starším malvérom GandCrab.(pridať odkaz na analýzu respektíve literatúru)

- **Emotet**

  Emotet je malvér, ktorý sa primárne šíri pomocou rôznych spam emailov.[**?**] Na infikovanie zariadenia používa rôzne skripty, makrá v dokumetoch alebo linky. Emotet stavia na infikovaní pomocou sociálneho inžinierstva. Prezentuje sa ako hodnoverný zástupca napr. banky, Rôznych internetových obchodov, atď. . Emotet malvér sa prvýkrát objavil v roku 2014 kedy vyuťíval na infikovanie rôzne JavaScript súbory.[**?**] V roku 2019 sa tento vírus objavil znova tentokrát v pokročilejšej verzii. V novej verzii je Emotet polymorfným malvérom čo mu umožňuje vyhnúť sa klasickej detekcii. Emotet používa modulárne knižnice Dynamic Link (DLL) na nepretržitý vývoj a aktualizáciu svojich schopností. Emotet môže navyše generovať falošné indikátory, ak je spustený vo virtuálnom prostredí čo zhoršuje jeho detekciu systéme.

- **ZeuS**

  Prvýkrát odhalený v roku 2007 sa Zeus Trojan, ktorý sa často nazýva Zbot, stal jedným z najúspešnejších kúskov botnetového softvéru na svete, postihol milióny počítačov a vytvoril množstvo podobných kusov škodlivého softvéru vytvoreného z jeho kódu. Po jeho minimalizovaní sa znovu objavil v pozmenenej podobe so zameraním na odchytávanie bankových operácii (Odchytávanie prihlasovacích údajov do internet bankingu).[**?**] Dosahuje to prostredníctvom monitorovania webových stránok a zaznamenávania klávesov, keď malvér zistí, že sa používateľ nachádza na bankovej webovej stránke, začne zaznamenávať stlačenia klávesov použité na prih-

lásenie. To znamená, že trójsky kôň dokáže obísť zabezpečenie na týchto webových stránkach. Infekcia prebieha pomocou spamov. Keď užívateľ klikne na odkaz v správe alebo stiahne obsah súboru. spolu s ním stiahne a spustí aj makro. Ktoré po nainštalovaní umožňuje sledovanie zariadenia.

- **Dridex**

  Dridex je známy trójsky kôň, ktorý sa špecializuje na krádež kreditných údajov online bankovníctva. Tento typ škodlivého softvéru sa objavil v roku 2014 a stále napreduje a vyvíja. Nový variant Dridex je schoplný vyhnúť sa detekcii tradičnými antivírusovými produktami. Dridex tiež zvýšil svoju infraštruktúru knižníc, ktoré používajú názvy súborov načítané legitímnymi spustiteľnými súbormi systému Windows. Názvy súborov a hash sa však obnovujú a menia zakaždým, keď sa obeť prihlási do infikovaného hostiteľa Windows. Tento malvér je v súčastnosti schopný detekovať približne 25 až 30 percent aktuálnych antivírusových softvérov.

- **Mirai**

  Mirai je samo sa množiaci vírus botnetov. Zdrojový kód pre *Mirai* bol autorom verejne sprístupnený po úspešnom a dobre propagovanom útoku na webovú stránku Krebbs. Kód botnetu Mirai infikuje zle chránené internetové zariadenia pomocou telnetu (sieťový komunikačný protokol založený na TCP) na nájdenie tých, ktoré stále používajú svoje predvolené uživateľské meno a heslo. Účinnosť systému *Mirai* je spôsobená jeho schopnosťou infikovať desiatky tisíc týchto nezabezpečených zariadení a koordinovať ich tak, aby začali útok DDoS proti vybranej obeti. [**?**]

  Mirai má dve hlavné zložky, samotný vírus a veliteľské a kontrolné stredisko (CnC). CnC je samostatný obraz, ktorý ovláda kompromitované zariadenia (BOT) a posiela im pokyny na spustenie jedného z útokov proti jednej alebo viacerým obetiam. Proces skenera prebieha nepretržite na každom infikovanom PC pomocou protokolu telnet (na porte TCP 23 alebo 2323)

  CnC predstavuje jednoduché rozhranie príkazového riadku, ktoré umožňuje útočníkovi určiť algoritmus, IP adresu obete a trvanie útoku. CnC tiež čaká na to, aby jej existujúce BOT-y vrátili novoobjavené adresy zariadení a poverenia, ktoré používa na skopírovanie vírusového kódu a následne na vytváranie nových BOT-ov.Algoritmy sú konfigurovateľné z CnC, ale v predvolenom nastavení má *Mirai* tendenciu náhodne rozdeľovať rôzne polia (ako sú čísla portov, poradové čísla, identifikátory atď.).

- **Osiris**

Osiris je potomkom malvéru Kronos, ktorý sa zameriaval na bankovnívctvo. Podobne ako Kronos je Osiris modernejšou verziou bankového trójskeho koňa.[**?**] Táto verzia malvéru využíva na skrývanie metódu process hollowing. Umožňuje mu predsieranie legitýmnych procesov v programe a tým sťažuje jeho detekciu, no niektoré antivírusy ho nie sú schopné detekovať. Malvér spúšťa svoj útok vydávaním sa za legitímny spustiteľný súbor. (Útoky zaznamená s malvérom Osiris boli dokumenty Microsoft Wordu.) Keď je spustený malvér na infikovanie zariadenia využíva dinamické knižnice, ktoré obsahujú škodlivý kód. Vydávanie sa za iný oficiálny softvér značne sťažuje identifikáciu malvéru a obmedzuje možnosti na zastavenie útoku.[**?**] Malvér v dokumetoch Word obsahoval aj makrá, ktoré po spustení stiahli další škodlivý malvér, ktorý umožuje zahĺtiť zariadenia, sťažiť detekciu a rozširovať útok.

- **Loki**

  Loki je ďalším potomkom staršieho malvéru Kronos, rovnako ako Osiris aj Loki využíva na svoje ukrytie metódy podobné process hollowing na zahltenie procesov a vydávanie sa za legitímny softvér. Loki sa zameriava na krádeže osobných údajov, zaujíma sa najmä o prihlasovacie údaje a heslá. Od augusta 2018 až do súčasnosti sa Loki zameriava na firemné poštové schránky prostredníctvom phishingových a spamových e-mailov. Phishingové e-maily zahŕňajú prílohu súboru s príponou .iso, ktorá sťahuje a spúšťa škodlivý softvér Trojan, ktorý ukradne heslá z prehľadávačov, pošty, klientov File Transfer Protocol (FTP), aplikácií na odosielanie správ a kryptomenných peňaženiek.

	Technika ukrývania						
Názov malvéru	Sodinokibi	Emotet	ZeuS	Dridex	Mirai	Osiris	Loki
DLL Injection		X	X		X		
Process hollowing						X	X
Thread Execution Hijacking							
Portable Executable Injection	X			X			
Hook injection							
APC Injection							
Extra windows memory injection							

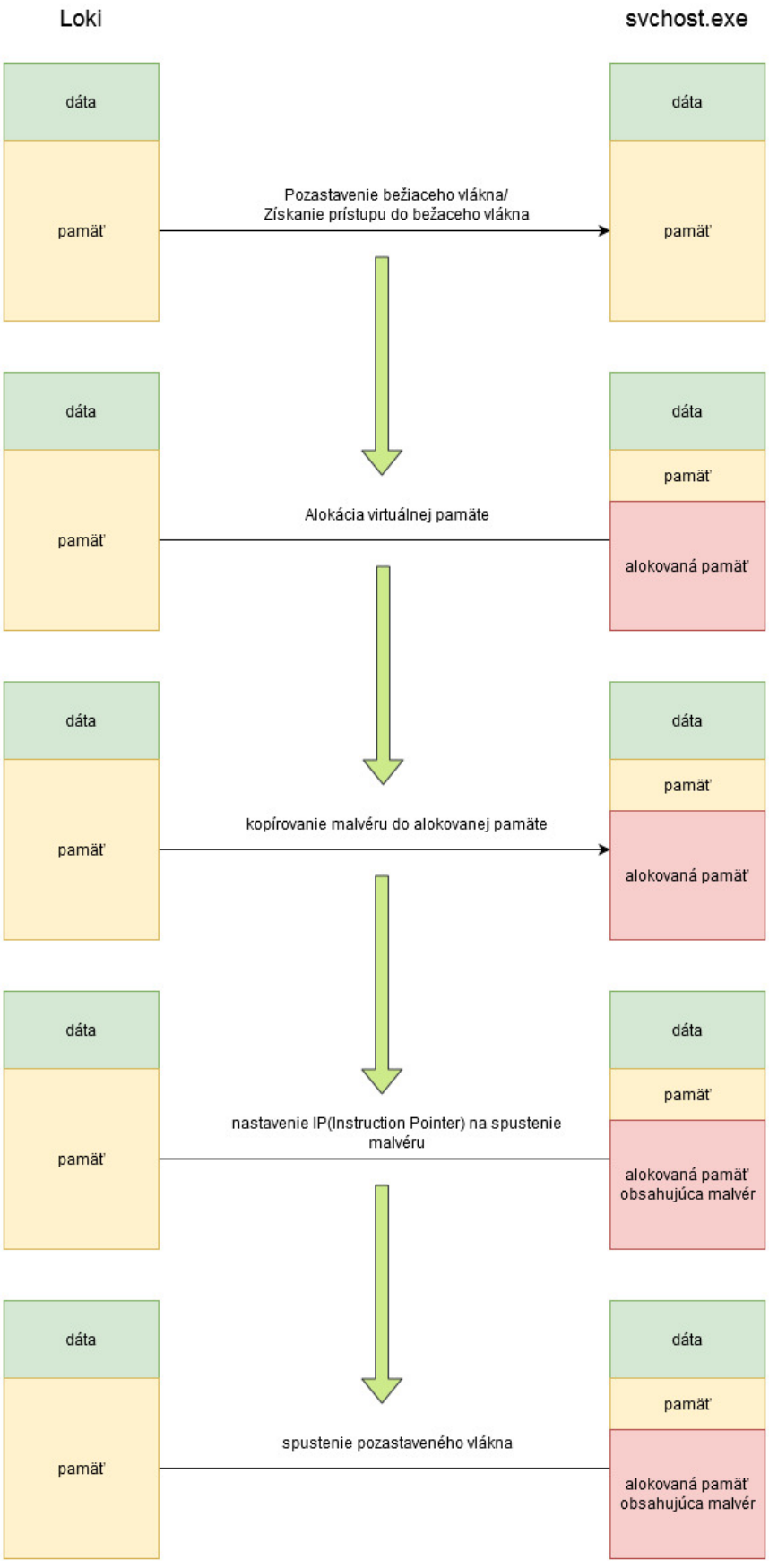
Tabuľka 1: Techniky ukrývania činnosti využívané súčasným malvérom.

2 Process hollowing

Zvolený spôsob ukrytia malvéru, ktorým sa bude táto práca zaoberať je *Process Hollowing, Hollow process*. Nasledujúca kapitola sa venuje spôsobu akým sa malvér môže ukryť. Obsahuje potenciálne API funkcie, ktoré môže *Process Hollowing* využívať na maskovanie svojho pôsobenia v systéme.

2.1 Princíp

Princíp ukrytia malvéru aký využíva *Process Hollowing* je podobný spôsobu ukrytia malvéru, ktorý využíva *DLL Injection*. Princím spočíva v ukrytí malvéru do existujúceho bežiaceho procesu, ktorý je v systéme Windows často využívaný a spustenie malvéru z tohto procesu by spôsobovalo najmenšie podozrenie. Takýmto demonštratívnym procesom môže byť *svchost.exe*, ktorý je v systéme Windows často využívaný. *Process Hollowing* pozastaví bežiace vlákno procesu *svchost.exe* a následne vytvorí vlákno, ktoré alokuje dostatočnú veľkú práznu virtuálnu pamäť pre malvér v procese. Po alokovaní virtuálneho adresného priestoru v pamäti procesu do alokovanej časti nakopíruje škodlivý kód a následne nastaví kedy sa daný škodlivý kód má spustiť. Malvér sa môže v tomto prípade spustiť po spustení pozastaveného vlákna alebo pri volaní nejakej konkrétnej funkcie. Po



Obrázok 1: Ukážka zmien adresného priestoru počas Hollowingu.

nastavení spúšťača malvéru sa *Process Hollowing* spustí pozastavené vlákno. Malvér môže pri následnom volaní funkcie, ktorá bola nastavená ako spúšťač škodlivého kódu spustiť svoju činnosť pod legitimným procesom.

2.2 Využívané API funkcie

Process Hollowing využíva na svoje fungovanie rôzne API funkcie a ich volania. Nasledujúce nami vybrané API funkcie, môže *Process Hollowing* najpravdepodobnejšie využívať na svoje fungovanie k ukrytiu škodlivého kódu do nejakého bežiaceho procesu.

- **CreateThread**
  Vytvorí vlákno na vykonanie procesu vo virtuálnom adresovom priestore volajúceho procesu.

- **CreateRemoteThread**
  Vytvorí vlákno, ktoré beží vo virtuálnom adresovom priestore iného procesu.

- **CreateRemoteThreadEx**
  Funkcia vytvára vlákno, ktoré sa spúšťa vo virtuálnom adresovom priestore iného procesu a prípadne špecifikujte rozšírené atribúty. Napríklad nekonečnosť skupiny procesorov.

- **ResumeThread**
  Spúšťa pozastavené vlákno.

- **SuspendThread**
  Pozastaví zadané vlákno.

- **SwitchToThread**
  Spôsobí, že volajúce vlákno poskytne vykonanie inému vláknu, ktoré je pripravené na spustenie v aktuálnom procesore. Operačný systém vyberie ďalšie vlákno, ktoré sa má vykonať.

- **CreateProcessA**
  Vytvorí nový proces a jeho primárne vlákno. Nový proces beží v bezpečnostnom kontexte volajúceho procesu.

- **CreateProcessW**
  Vytvorí nový proces a jeho primárne vlákno. Nový proces beží v bezpečnostnom kontexte volajúceho procesu.

- **VirtuallAlloc**
  Rezervuje, potvrdzuje alebo mení stav oblasti stránok vo virtuálnom adresovom priestore volaného procesu. Pamäť pridelená touto funkciou sa automaticky inicializuje na nulu.

- **VirtualAllocEx**
  Vyhradzuje, potvrdzuje alebo mení stav oblasti pamäte v rámci virtuálneho adresového priestoru špecifikovaného procesu. Funkcia inicializuje pamäť, ktorú nastaví na nulu.

- **VirtuallAlloc2**
  Vyhradzuje, potvrdzuje alebo mení stav oblasti pamäte v rámci virtuálneho adresového priestoru špecifikovaného procesu. Funkcia inicializuje pamäť, ktorú nastaví na nulu. Pomocou tejto funkcie môžete: pre nové pridelenia určiť rozsah virtuálneho adresového priestoru a obmedzenia zarovnania výkonu 2; špecifikovať ľubovoľný počet rozšírených parametrov; špecifikovať preferovaný uzol NUMA pre fyzickú pamäť ako rozšírený parameter.

- **WriteProcessMemory**
  Zapisuje údaje do oblasti pamäte v zadanom procese. Celá oblasť, do ktorej sa má zapísať, musí byť prístupná inak operácia zlyhá.

- **ReadProcessMemory**
  Číta udaje z pamäte v zadanom procese.

- **CopyMemory**
  Kopíruje pamäť do vyhradenej alokovanej pamäti.

- **SetThreadContext**
  Nastavuje kontext pre aktuálne vlákno.

- **ExitThread**
  Ukončí aktuálne vykonávané vlákno.

3 Existujúce riešenia na detekciu

Doposiaľ známe existujúce riešenia na detekciu techniky *Process Hollowing* využívanej niektorými malvérmy, sú určené na forenznú analýzu. Táto analýza prebieha už po infikovaní zariadenia malvérom a zistením, že sa škodlivý kód už v zariadení nachádza. Tieto riešenia neobsahujú spôsoby detekcie škodlivého kódu, ktoré by mohli odchytiť malvér už pri infikovaní ľubovoľného zariadenia.

https://github.com/m0n0ph1/Process-Hollowing

https://github.com/idan1288/PHDetection

https://www.andreafortuna.org/2017/10/09/understanding-process-hollowing/

https://cysinfo.com/detecting-deceptive-hollowing-techniques/

https://www.microsoft.com/security/blog/2017/07/12/detecting-stealthier-cross-process-injection-techniques-with-windows-defender-atp-process-hollowing-and-atom-bombing/

3.1 PHDetection

PHDetection hľadá moduly, od ktorých závisí pôvodný .exe program. *PHDetection* kontroluje či sú moduly načítané do procesnej pamäte programu. Ak nájde moduly, na ktorých závisí dotyčný program ale nenájde ich v pamäti procesu, čo naznačuje, že proces je prázny *PHDetection* detekuje, že sa jedná o Process Hollowing. Pôvodné moduly boli nahradené touto metodóu za iné ktoré obsahujú škodlivý kód. Existuje niekoľko súborov, ktoré nezávisia od mnohých modulov v IAT. Program analyzovuje aj tabuľku importu oneskoreného načítania volania modulov. Program bol implementovaný v jazyku C++. Program sa spúšťa spustením súboru podľa verzie systému Windows [**?**]

V každom riešení popis, akým spôsobom daný nástroj funguje, ako sa používa, v čom bol implementovaný + link na nástroj do literatúry.

3.2 HollowFind

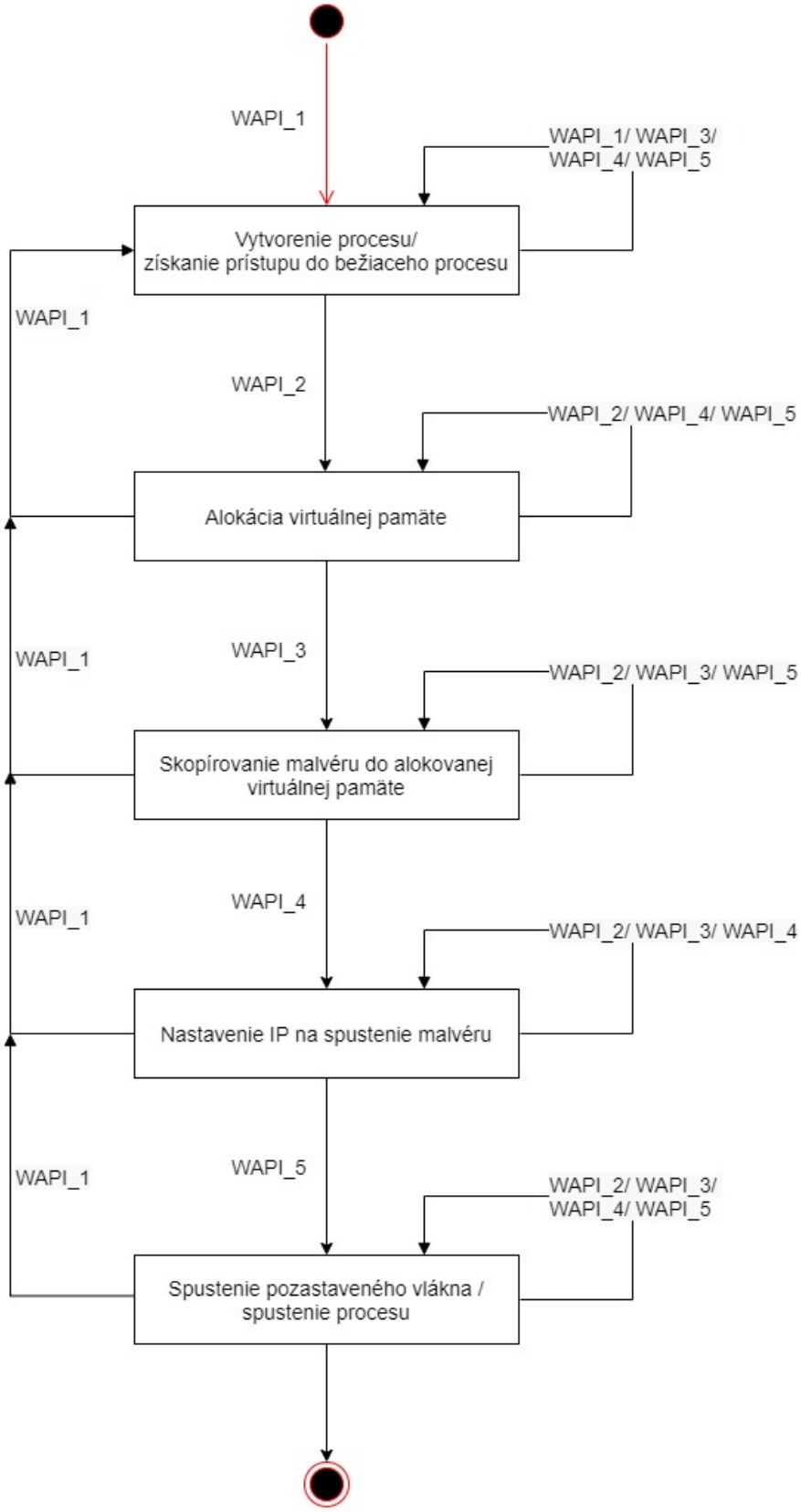
Hollowfind je plugin pre program Volatility na detekciu rôznych typov techník *Process Hollowing* používaných vo voľnej prírode na obídenie, zamenenie a odklonenie techník forenznej analýzy. Plugin detekuje takéto útoky zistením nezrovnalostí vo VAD a PEB, tiež rozoberie adresu vstupného bodu, aby zistil akékoľvek pokusy o presmerovanie, a tiež nahlási akékoľvek podozrivé pamäťové oblasti, ktoré by mali pomôcť pri detekcii akéhokoľvek škodlivého kódu. Program bol napísaný v jazyku Phyton. Plugin sa spúšta v Programe Volatility po nainštalovaní. [**?**]

4 Algoritmus na detekciu

Navrhnutý algoritmus na detegciu malvéru využívajúceho na svoje ukrytie v systéme *Process Hollowing*, tvorí 5 prechodných stavov. Cez tieto sa algoritmus dostáva vďaka vybraným Windows API funkciám. Tieto funkcie umožňujú sledovať volania jednotlivých procesov, kde by mohol ukryť malvér svoju činnosť. Jednotlivé API tak môžeme rozdeliť do 5 skupín.

- **WAPI1** Sú API funkcie, ktoré umožňujú vytvárať nové vlákna procesov alebo získať prístupy do už existujúcich vlákien.
- **WAPI2** Predstavujú API, ktoré slúžia na alokáciu pamäte pre dané vlákno alebo rozširujú virtuálnu pamäť existujúceho procesu.
- **WAPI3** Tieto funkcie umožňujú manipulovať s pamäťou. Ako je kopírovanie pamäte, zapisovanie do pamäte atď..
- **WAPI4** Tieto API funkcie nastavujú kontext daného vlákna a umožňujú nastaviť *Instruction Pointer*, ktorý pri volaní spustí aj funcionalitu malvéru.
- **WAPI5** Poslednou skupinou sú API, ktoré spúšťajú pozastavené vlákno alebo ukončiť funcionalitu aktuálneho vlákna.

Algoritmus na detegovanie funguje na jednoduchom princípe. Podozrivý proces získa prístup do bežiaceho vlákna legitímneho procesu. Vo vlákne daného procesu malvér alokuje virtuálnu pamäť čím následne umožní do tejto pamäte zapísať svoj škodlivý kód. Počas týchto výkonov operácií malvér volá jednotlivé API. Pri správnom odhade postupnosti volaných API, je algoritmus schopný vyhodnotiť v reálnom čase či v zariadení existuje hrozba v podobe malvéru a informovať o tom uživateľa.

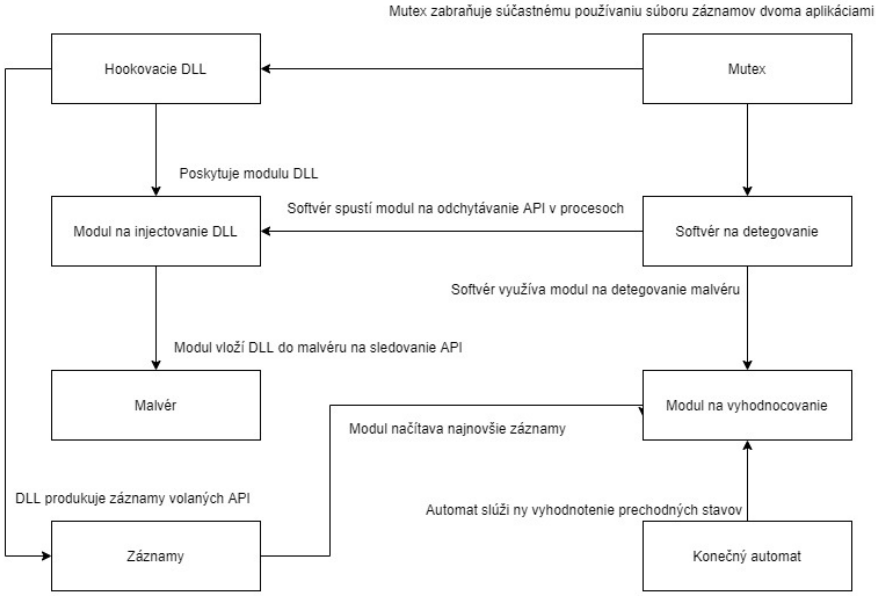


Obrázok 2: Konečný stavový automat.

5 Implementácia

Stručný úvod do implementácie, programovací jazyk, vývojové prostredie + jednoduchý diagram fungovania celého systému.

Pre implementáciu riešenia je zvolený programovaný jazyk C++, pretože primárnym testovavím prostredím budú najčastejšie používané systémy Windows. Na simuláciu testovanie prostredia sa použije **Virtual Box** v ktorom budeme simulovať bežné používanie systému. Zvolené vývojové prostredie je **Visual Studio 2019**, ktoré nám umožňuje pracovať s najnovšími verziami systému a aj uľahčiť jednuduchšiu implementáciu algoritmu.

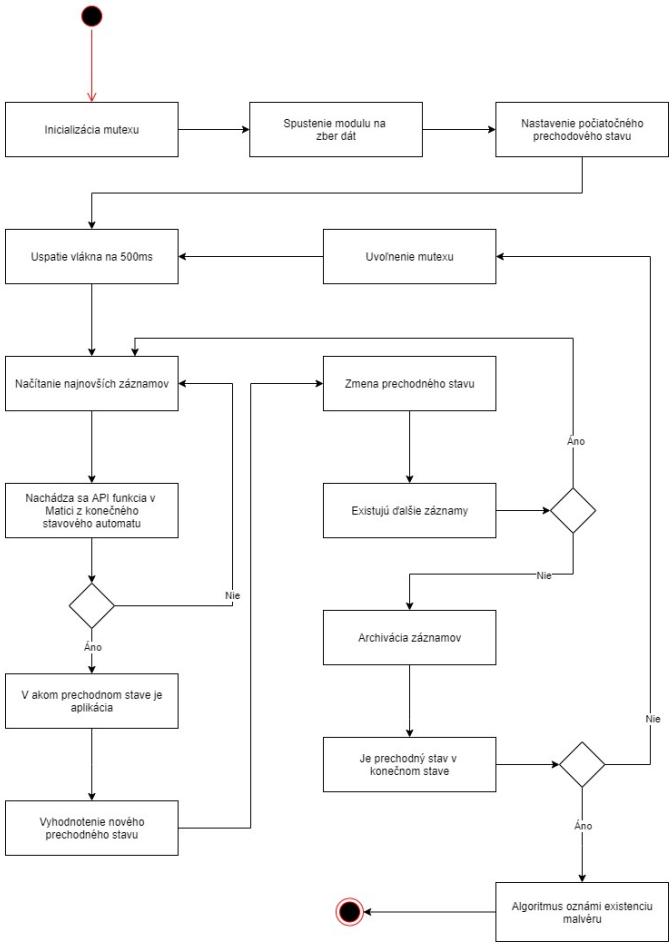


Obrázok 3: Moduly aplikácie

Práca sa zaoberá návrhom a implementáciou spôsobu detegovania malvéru, ktorý na ukrytie svojej činnosti v PC používa *Process Hollowing*. Teda ukrýva svojú činnosť za iné bežné programy, ktoré nie sú podozrivé. Aplikácia určená na detegovanie takéhoto spôsobu ukrývania malvéru bude bežať v reálnom čase, teda je schopná detegovať malvér počas jeho behu. Vstupné dáta do tejto aplikácie budú predstavovať volané API funkcie škodlivým softvérom a z týchto API funckií bude aplikácia vyhodnocovať a poskytovať výsledky či v systéme operuje alebo neoperuje malvér. Aplikácia bude pozostávať z nasledujúcich častí. Mutex, ktorý bude zabezpečovať plynulosť a bezpečnosť programu. Modul na injectovanie DLL, ktorý bude vkladať DLL knižnicu na odchytávanie volaných API. Modul na vyhodnocovanie prítomnosti mavéru, ktorý sa skladá z viacerých častí. Mod-

ulu na načítanie najnovších záznamov z DLL knižnice. Konečného stavového automatu reprezentovaného Maticou, oproti ktorej bude aplikácia porovnávať prechody medzi jednotlivými stavmi podľa volaných API. Archiváciou, ktorá zabezpečí archiváciu dát po prečítaní najnovších záznamov z DLL knižnice.

5.1 Algoritmus



Obrázok 4: UML diagram vyvýjaného algoritmu.

Aplikácia na začiatku inicializuje mutex, ktorý slúži na bezpečné zapisovanie a čítanie záznamov zo súboru. Následne aplikácia spustí modul na zber na dát, ktorý monitoruje aké API sa volajú v systéme Windows. Pre jednoduchšie monitorovanie využívame voľne dostupnú aplikáciu *Detours*. Aplikácia si nastaví počiatočnú hodnutu prechodného stavu, ktorý slúži na vyhodnocovanie prítomnosti malvéru. Po nastavení všetkých počiatočných hodnôt aplikácia prejde na samotnú detegciu malvéru. Vlákno sa uspí na stanovenú dobu a po nastavení mutexu začína aplikácia samotný proces detegovania. Aplikácia si načíta na-

jnovšie záznami z modulu na zber dát. Zo záznamu aplikácia vyberie volanú API funkciu a zisťuje či k danej API existuje hodnota v matici konečného automatu. Ak áno, aplikácia zisťuje aktuálny prechodný stav a následne aj pomocou konečného stavového automatu novú hodnotu do ktorej sa prechodný stav následne zmení. Po vykonaní zmeny nastane kontrola na existenciu ďalších záznamov v súbore ak súbor obsahuje ďalšie záznami, načíta si ďalší záznam a proces sa opakuje. Ak už aplikácia načítala všetky záznami spraví sa archivácia záznamov a aplikácia vyhodnocuje prechodný stav konečného stavového automatu. Ak prechodný stav nieje v konečnom stave mutex sa uvoľní a aplikácia čaká ďalšie záznami z modulu na zber dát a cyklus sa opakuje. Keď sa prechodný stav dostane do konecného stavu aplikácia oznámi existenciu malvéru a ukončí sa.

5.2 Modul na zber dát

Modul na zber dát predstavuje DLL knižnicu (**Hook.dll**) využívanú na vloženie do vybraného procesu, cez ktorú sa následne odchytávajú vybrané API funkcie zapísaním do textového súboru. Modul sa skladá z dvoch častí. Prvá časť predstavuje definície vybraných API funcií ako môžeme vydieť na obrázku, ku ktorým je následne definované aj ich volanie. Okrem volania konkrétnej API funkcie aj do pripraveného textového súboru zapíše čas a volanú API funkciu vo **writeFunctionToFile**, ktoré sú následne použité v aplikácii na detegovanie malvéru v bežiacom procese.

```
void writeFunctionToFile(std::string originalFuncion)
{
   DWORD ret = WaitForSingleObject(hMutex, INFINITE);

   if (ret == WAIT_OBJECT_0)
   {
      time_t now = time(NULL);
      tm* ltm = localtime(&now);
      std::ofstream myFile("api_data.txt", std::ofstream::app |
          std::ofstream::out);

      if (myFile.is_open())
      {
         myFile << ltm->tm_hour << ":" << ltm->tm_min << ":" << ltm->tm_sec <<
             ";" + originalFuncion << endl;
         myFile.close();
      }
```

```
      ReleaseMutex(hMutex);
   }
}
```

Výpis č. 1

Druhá časť aplikácie už slúži len na nahradenie pôvodnej volanej API funkcie, modifikovanou tou istou funckiou, ktorá je rozšírenia o zapisovanie volanej API do súboru. Pôvodnú funkciu *SetThreadContext* určenú na nastavenie *Instriction Pointer* v tomto module nahradíme funciou *HookSetThreadContext*, ktorá okrem volania pôvodnej funkcie volá aj funkciu **writeFunctionToFile** na zapísanie API funkcie do predpripraveného súboru.

```
static BOOL(__stdcall *RealSetThreadContext)(HANDLE, const CONTEXT*) =
    SetThreadContext;

BOOL WINAPI HookSetThreadContext(HANDLE hThread, const CONTEXT* lpContext)
{
   writeFunctionToFile("SetThreadContext");
   return RealSetThreadContext(hThread, lpContext);
}
```

Výpis č. 2

```
switch (ul_reason_for_call)
{
   case DLL_PROCESS_ATTACH:
   {
      DetourAttach(&(PVOID&)RealSetThreadContext, HookSetThreadContext);
      DetourTransactionCommit();
      break;
   }
   case DLL_PROCESS_DETACH:
   {
      DetourDetach(&(PVOID&)RealSetThreadContext, HookSetThreadContext);
      DetourTransactionCommit();
      break;
   }
```

```
}
```

Výpis č. 3

5.3 Detours

Detours je softvér určený na monitorovanie a inštruovanie API volaní v systéme Windows. Tento softvér podporuje rovnako 32 tak aj 64 bitovú verziu Windowsu. *Detours* uľahčuje prácu vývojárom, ktorý pracujú s rozhraním API volaní. Je k dispozícii na základe *Open source* licencie a je voľne dostupný pre komunitu.

5.4 Mutex

Mutex je synchronizačný objekt, ktorý slúži na signalizáciu používania nejakého objektu v procesnom vlákne. Mutex slúži na koordináciu viacerých aplikácií (dvoch vlákien), ktoré vyžadujú prístup k rovnakému objektu súčastne. Napríklad aby sa zabránilo zápisu dvoch vlákien súčastne do zdieľaného objektu alebo pamäti. Takže každá aplikácia alebo vlákno čaká na uvoľnonenie mutexu, ktoré signalizuje, že daná pamäť nieje využívaná a môže sa do nej bezpečne zapisovať bez toho aby došlo zápisu dvoch aplikácii súčastne. V tejto aplikácii mutex slúži na zabránenie zapisovaniu záznamov do súboru modulom na zber dát a čítaním záznamov z toho istého súboru detegčnou aplikáciou súčastne.

```
HANDLE mutexOnThreadSafe;
mutexOnThreadSafe = CreateMutex(NULL, FALSE, TEXT("MutexOnThreadSafe"));
```

Výpis č. 4

5.5 Konečný stavový automat

Konečný automat je teoretický výpočtový model používaný v informatike na štúdium rôznych formálych jazykoch. Popisuje veľmi jednoduchý počítač, ktorý môže byť v jednom z niekoľkých stavov, medzi ktorými prechádza na základe symbolov, ktoré číta zo vstupu. Množina stavov je konečná , konečný automat nemá žiadnu ďalšiu pamäť, okrem informácie o aktuálnom stave. V informatike sa rozlišuje okrem základného deterministického či nedeterministického automatu tiež Mealyho a Moorov automat.

Konečný automat je definovaný ako usporiadaná pätica (S, Σ , σ, s, F) kde:
S je konečná neprázdna množina stavov.
Σ je konečná neprázdna množina vstupných symbolov, nazývaná abeceda.
σ je prechodová funkcia respektíve prechodová tabuľka popisujúca prechod medzi jed-

notlivými stavmi.
s je počatočná stav patriaci do množiny stavov S.
F je množina finálych akceptujúcich stavov.

Na začiatku sa automat nachádza v definovanom počiatočnom stave. Ďalej v každom kroku prečíta jeden symbol zo vstupu a prejde do stavu, ktorý je daný hodnotou, ktorá v prechodovej tabuľke zodpovedá aktuálnemu stavu a prečítanému symbolu. Potom pokračuje čítaním ďalšieho symbolu zo vstupu, ďalším prechodom podľa prechodovej tabuľky atď.

Podľa toho, či automat skončí po prečítaní vstupe v stave, ktorý patrí do množiny prijímajúcich stavov, platí, že automat buď daný vstup prijal, alebo neprijal. Množina všetkých reťazcov, ktoré daný automat príjme, tvorí regulárny jazyk.

V našom prípade abecedu tvorí množina nami vybraných API windows funcií, ktoré by potenciálne mal najčastejšie využívať malvér využívajúci techniku *Process Hollowing* na ukrytie svojej činnosti.

5.6 Matica konečného stavového automatu

Matica reprezentuje prechody medzi jednotlivými stavmi konečného stavového automatu. tieto prechody sú definované ako volania vybraných API, ktoré posúvajú automat cez jednotlivé stavy podľa toho v akom stave je aktuálne automat a do akého nového stavu sa automat dostane. Konečná množina stavov je reprezentovaná stavmi "štart","vytvorenie vlákna", "alokácia pamäte", "kopírovanie malvéru", "nastavenie IP", "spustenie vlákna" . Počiatočný stav automatu je **štart** a konečný stav je **spustenie vlákna**, ktorý oznamuje existenciu malvéru v zariadení.

	Prechodové stavy					
Volané API funkcie	Štart	Vytvorenie procesu/ Získanie prístupu do procesu	Alokácia virtuálnej pamäte	Kopírovanie malvéru do alokovanej pamäte	Nastavenie IP na spustenie malvéru	Spustenie vlákna/ Spustenie pozastaveného procesu
Skratky stavov	S0	S1	S2	S3	S4	S5
CreateThread	S1	S1	S1	S1	S1	S1
CreateRemoteThread	S1	S1	S1	S1	S1	S1
CreateRemoteThreadEx	S1	S1	S1	S1	S1	S1
CreateProcessA	S1	S1	S1	S1	S1	S1
CreateProcessW	S1	S1	S1	S1	S1	S1
SwitchToThread	S1	S1	S1	S1	S1	S1
OpenThread	S1	S1	S1	S1	S1	S1
SuspendThread	S1	S1	S1	S1	S1	S1
VirtualAlloc	S0	S2	S2	S3	S4	S5
VirtualAllocEx	S0	S2	S2	S3	S4	S5
CopyMemory	S0	S1	S3	S3	S4	S5
WriteProcessMemory	S0	S1	S3	S3	S4	S5
ResumeThread	S0	S1	S2	S3	S5	S5
ExitThread	S0	S1	S2	S3	S5	S5
SetThreadPriority	S0	S1	S2	S4	S4	S5

Tabuľka 2: Matica prechodov medzi stavmi pri volaniach API.

6 Výsledky

Stručný úvod do testovania nami vytvoreného riešenia/riešení. Zoznam experimentov.

6.1 Experiment A

Detailný popis experimentu, výsledky, úspešnosť, graf, koľko trval čas detegovania od spustenia experimentu a pod. To navrhneme podľa výsledkov implementácie.

6.2 Experiment B

6.3 Porovnanie s existujúcimi riešeniami

Ak to bude možné. Ak nie spomenúť že existujúce riešenia využívajú metódy, ktoré neumožňujú priame porovnanie a pod.

Záver

Písať tiež až úplne na záver Odseky:

Všeobecne čo bolo cieľom práce a čím sme sa zaoberali.

Popísať ako sa podarilo splniť jednotlivé ciele, prípadne ak sa ich splniť nepodarilo tak prečo. Stručne zosumarizovať výsledky experimentov.

Popísať nejaké ciele do budúcnosti, čo by bolo vhodné ešte vylepšiť, pridať, upraviť.

Prílohy

A Štruktúra elektronického nosiča

pozor tu nema byt ziadny vypis jednotlivych suborov len zoznam s polozkami, napr. priecinok src obsahuje zdrojove kody, bp.pdf pracu atd