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

Práca sa zaoberá súčasnými technikami na ukrývanie činnosti malvéru v napadnutých systémoch. V úvodnej časti práce popisujeme malvér a jednotlivé techniky na ukrývanie, ktoré sú najčastejšie využívané v posledných rokoch. V tejto časti práce sme sa zaoberali najmä vzorkami, ktoré boli detegované počas posledného roka. Cieľom práce bolo navrhnúť detekčný algoritmus vybranej techniky. Metóda, ktorú sme sa rozhodli bližšie preskúmať je tzv. *process hollowing*. Navrhnutý algoritmus je založený na princípe sledovania zvolených Windows API volaní, pomocou ktorých sa dokáže samotný malvér ukryť do bežného procesu. Základom algoritmu je konečný stavový automat, ktorý vyhodnocuje prítomnosť *process hollowing* techniky sledovaním postupnosti typických API volaní, používaných na implementáciu danej metódy. V ďalšej časti popisujeme samotný algoritmus, jednotlivé moduly detekčnej aplikácie a spôsob, akým sme ich implementovali. V záverečnej časti uvádzame výsledky, ktoré sme dosiahli v rámci experimentov so skutočnými vzorkami malvéru.

Kľúčové slová: malvér, process hollowing, detekcia, konečný stavový automat

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: malware, process hollowing, detection, finite state machine

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

APC - Asynchronous Procedure Call
API - Application Programming Interface
BOT - RoBOT
CnC - Command and Control Center
DLL - Dynamic Link Library
EWM - Extra Windows Memory
FTP - File Transfer Protocol
PE - Portable Executable
PEB - Process Environment Block
VAD - Virtual Address Descriptor

Zoznam výpisov

Úvod

Malvér môžeme charakterizovať ako škodlivý kód, ktorého primárnym cieľom je uškodiť používateľovi napr. stratou peňazí alebo súkromia, zneužitím osobných údajov a pod. Malvér na svoju infiltráciu najčastejšie využíva bezpečnostné chyby v systéme, prípadne rôzne phishingové kampane. Dôležitým prvkom každého škodlivého kódu je taktiež spôsob, akým sa môže vyhnúť odhaleniu. Jedným z takýchto spôsobov je ukrytie sa v pamäti systému a vydávanie sa za legitímny proces. Kým v minulosti vznikalo približne zopár vzoriek ročne, dnes sú to tisícky jedinečných súborov denne. Z toho dôvodu je nutné sa zaoberať detekciou a odhaľovaním škodlivého kódu v systéme aby sme vedeli zabrániť rôznym bezpečnostným incidentom.

V práci sa zaoberáme technikami, pomocou ktorých dokáže malvér ukryť svoju existenciu pred antivírusovými riešieniami a forenznými inžiniermi. Medzi najznámejšie metódy patria napr. *DLL injection*, *process hollowing* alebo *portable executable injection*. V našej práci sme sa rozhodli bližšie preskúmať techniku *process hollowing*. Princíp spomínanej metódy je podobný ako pri klasickom injektovaní škodlivého kódu do cudzieho procesu. Rozdiel spočíva najmä v tom, že *process hollowing* taktiež odmapuje pamäť pôvodného procesu. V rámci práce sme navrhli detekčný algoritmus, ktorý je založený na konečnom stavovom automate. Aplikácia sleduje v reálnom čase dôležité API volania a na ich základe sa posúva v automate. Jednotlivé stavy konečného automatu, reprezentujú fázy v ktorých sa nachádza malvér, počas injektovania škodlivého kódu do cudzieho procesu pomocou techniky *process hollowing*.

Štruktúra našej práce je nasledovná. V úvodnej kapitole popisujeme najznámejšie techniky na ukrývanie činnosti malvéru. Taktiež uvádzame vzorky malvéru, ktoré boli detegované počas minulého roka spolu s identifikáciou techniky, ktorú používajú.

V druhej kapitole sa zaoberáme detailnou charakteristikou techniky *process hollowing*. V rámci kapitoly taktiež uvádzame zoznam Windows API volaní, ktoré sa používajú na implementáciu techniky *process hollowing*.

Kapitola 3 popisuje súčasné riešenia, ktoré sa zaoberajú detekciou techniky *process hollowing*.

V ďalšej kapitole uvádzame popis nami navrhnutého algoritmu. V rámci kapitoly prezentujeme navrhnutý konečný stavový automat a tabuľku prechodov.

Kapitola 5 sa zaoberá popisom implementácie jednotlivých modulov detekčnej aplikácie, použitých knižníc a technológií.

Záverenčná kapitola sa venuje výsledkom, ktoré sme dosiahli pomocou našej detekčnej aplikácie.

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.[5] 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 skriptoch alebo dynamických knižniciach, aby boli čo najmenej detegovateľné. [16]

- **DLL Injection / Reflective DLL Injection**
  DLL Injection je technika, pri ktorej malvér zapíše cestu ku škodlivému DLL súboru do virtuálneho adresného priestoru legitímneho procesu a následne zabezpečí aby daný proces túto dynamickú knižnicu načítal. Postup je nasledovný:

  – Škodlivý kód získa prístup ku legitímnemu procesu.
  – V rámci legitímneho procesu alokuje priestor dostatočne veľký na zapísanie cesty ku škodlivému DLL súboru.
  – Malvér zapíše cestu ku škodlivému DLL súboru do virtuálneho adresného priestoru legitímneho procesu.
  – Malvér na záver spustí vlákno v rámci legitímneho procesu, ktorého cieľom je načítať a spustiť DLL súbor.

  Hlavným cieľom *DLL Injection* je škodlivý kód ukryť do legitímneho procesu, kde je malvér ukrytý pred antivírovým softvérom, odkiaľ môže byť následne spustený [9].

- **Proces Hollowing**

  Využíva podobné princípy ukrývania škodlivého softvéru ako *DLL Injection*. Cieľom je ukryť škodlivý kód do existujúce procesu, z ktorého sa následne bude vykonávať[9]. Princíp je podobný ako pri technike *DLL Injection*:

  - Malvér spustí nový pozastavený legitímny proces, do ktorého sa plánuje ukryť.
  - Škodlivý kód odmapuje pamäť legitímneho procesu.
  - Do virtuálneho adresného priestoru legitímneho procesu nakopíruje svoj škodlivý kód.
  - Po dokončení kopírovania nastaví nový vstupný bod procesu a obnoví pozastavený proces.

  Ako môžeme vidieť, hlavný rozdiel spočíva najmä v odmapovaní existujúceho kódu. V prípade *Process Hollowing* techniky bude v rámci neškodného procesu bežať výhradne iba škodlivý kód, na rozdiel od *DLL injection*, kde v rámci jedného procesu sa nachádza škodlivý kód spolu s pôvodným. Výsledkom môže byť napr. spustený bežný proces *svchost.exe*, ktorý ale v skutočnosti vykonáva škodlivú aktivitu.

- **Thread Execution Hijacking**

  Pri tejto technike existujú určité podobnosti s metódou *Process Hollowing* a *DLL injection* [9]. Hlavný princíp spočíva v získaní prístupu už k existujúcemu vláknu legitímneho procesu, do ktorého chceme vložiť škodlivý kód. Po získaní prístupu k vláknu, malvér dané vlákno pozastaví. Samotné injektovanie môže uskutočniť podobne ako pri *DLL injection*, zapísaním cesty k DLL súboru a následnom načítaní. Nevýhodou takéhoto spustenia pozastaveného programu je, že môže spôsobiť pád systému v rámci systémového volania. Výhodou ostáva, že technika nepotrebuje vytvoriť nové vlákno alebo proces a použije už existujúce.

- **Portable Executable Injection**

  Výhodou tohoto spôsobu ukrytia malvéru je využitie nakopírovania celého PE súboru do existujúceho procesu [9]. Ako výhodu môžeme taktiež považovať, že malvér nepotrebuje uložiť žiadne súbory na disk ale len priamo prekopíruje dáta PE súboru do legitímneho procesu. Hlavnou nevýhodou je, že sa zmení bázová adresa procesu a z toho dôvodu je nutné prepočítať nové adresy v legitímnom procese tak, aby bolo zabezpečené korektné správanie novo vloženého PE súboru.

- **Hook injection**

  Hookovanie je všeobecne technika používaná na zachytávanie API volaní. *Hook injection* využíva túto techniku na načítanie škodlivého DLL, pri zachytení určitej udalosti v konkrétnom vlákne [9]. Pomocou prislušného API volania vieme nainštalovať hook na špecifickú udalosť v systéme (napr. stlačenie klávesy), pre konkrétne vlákno v systéme. Taktiež vieme špecifikovať smerník na funkciu, ktorá sa má zavolať ak daná udalosť nastane. V tejto funkcii následne vieme určiť že sa má načítať napr. škodlivé DLL. Môžeme si všimnúť že väščina techník sa samotné načítanie kódu využíva metódu načítania škodlivého DLL do regulérneho procesu a hlavné rozdiely spočívajú najmä v riešení ako donúniť cudzí proces túto funkcionalitu vyvolať.

- **APC Injection**

  Škodlivý softvér môže využívať výhody tzv. *Asynchronous Procedure Calls* (APC), aby prinútil iné vlákno spustiť svoj vlastný kód [9]. Túto funkcionalitu vie dosiahnúť tak, že vloží dané vlákno do APC fronty. Každé vlákno má vlastnú APC frontu, z ktorej vykonáva volania ak sa nachádza v pozastavenom stave a čaká na konkrétne udalosti. Podobne ako v predchádzajúcich prípadoch môže malvér do APC fronty vložiť smerník na funkciu, v ktorej sa načíta škodlivé DLL. Malvér zvyčajne hľadá ľubovoľné vlákno ktoré sa nachádza v špeciálnom stave, pri ktorom vykonáva postupne položky v APC fronte. Takýchto vláken je väčšinou v systéme veľa.

- **Extra windows memory injection**

  Tento spôsob schovávania softvéru sa spolieha na možnosť špecifikovať dodatočnú pamäť pri registrácii aplikačných okien v systéme. Pri registrácii nového okna aplikácie, softvér špecifikuje ďalšie bajty pamäte, ktoré rozšíria veľkosť alokovanej pamäte pre spustenú aplikáciu [9], nazývané aj *Extra Windows Memory*(EWM). V tejto časti ale nevzniká dostatok miesta na uloženie dát. Aby sa toto obmedzenie obišlo, škodlivý softvér zapíše kód do zdieľanej pamäte a do EWM vloží ukazovateľ na danú časť. Do tejto rozšírenej časti zieľanej pamäte ďalej softvér zapíše smerník na funkciu, ktorá obsahuje kód na načítanie malvéru. Malvér môže nakoniec vyvolať spustenie tohoto kódu pomocou konkrétnych Windows API volaní.

1.2 Súčasný malvér

Táto kapitola obsahuje opis jednotlivých vzoriek škodlivého kódu z roku 2019, ktoré boli detegované spoločnosťami ako Avast a McAfee. Tieto vzorky sú najčastejšie využívané v oblasti Európy. Kapitola obsahuje bližší opis jednotlivých vzoriek, ich využitie, použité spôsoby útokov a ukrytie malvéru v systéme.

- **Sodinokibi**
  Tento malvér bol detekovaný v období okolo apríla 2019. Patrí do rodiny ransomvéru, ktorých cieľom je zašifrovať dáta v zariadení a následne za dešifrovanie pýtať peniaze [6] (väčšinou v podobe kryptomeny). Názov bol objavený v heši, ktorý obsahoval názov *Sodinokibi.exe*. Vírus sa šíri sám zmeužívaním zraniteľnosti v serveroch, ktoré používajú *Oracle WebLogic*. Kód je navrhnutý tak, aby rýchlo vykonával šifrovanie špecifických súborov, definovaných v konfigurácii ransomvéru. Prvou akciou škodlivého kódu je načítať všetky externé funkcie potrebné počas behu programu. Technika využívaná na ukrytie malvéru je *Portable Executable Injection*. Analýza spoločnosti McAfee ukazuje podobnosť s iným starším malvérom GandCrab.

- **Emotet**
  Emotet je malvér, ktorý sa primárne šíri pomocou rôznych spam emailov [12]. Na infikovanie zariadenia používa rôzne skripty, makrá v dokumetoch alebo linky. Emotet sa teda spolieha najmä na techniky sociálneho inžinierstva. Prezentuje sa ako hodnoverný zástupca napr. banky, rôznych internetových obchodov, a pod. Emotet sa prvýkrát objavil v roku 2014 kedy využíval na infikovanie rôzne JavaScript súbory [2]. V roku 2019 sa tento vírus objavil znova tentokrát už v pokročilejšej verzii. V novej verzii je Emotet už polymorfným škodlivým kódom, čo mu umožňuje vyhnúť sa klasickej detekcii. Emotet môže navyše generovať falošné funkcionalitu, 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 botnetov na svete a postihol milióny počítačov [11]. Taktiež bolo vytvorerných množstvo variantov, ktoré boli založené na tomto malvéri. Po čase 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 webovej stránke banky, začne zaznamenávať stlačenia klávesov použité na prihlásenie. 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 v online bankovníctve. Tento typ škodlivého kódu sa objavil v roku 2014 a stále sa postupne vyvíja. Nový variant Dridex je schoplný vyhnúť sa detekcii tradičnými antivírusovými produktami. Tento malvér je v súčastnosti schopný detekovať približne 25 až 30 percent aktuálnych antivírusových softvérov. [14]

- **Mirai**

  Mirai je samošíriteľný typ škodlivého súboru na vytvorenie botnetu. Zdrojový kód pre *Mirai* bol autormi verejne sprístupnený po úspešnom a dobre propagovanom útoku na webovú stránku Krebbs. Kód botnetu Mirai infikuje zariadania pripojené k internetu, ktoré využívajú telnet protokol (sieťový komunikačný protokol založený na TCP) na nájdenie tých, ktoré stále používajú svoje predvolené uživateľské meno a heslo. Účinnosť málvéru *Mirai* je spôsobená jeho schopnosťou infikovať desiatky tisíc týchto nezabezpečených zariadení a koordinovať ich tak, aby začali útok DDoS proti vybranej obeti. [10]

  Mirai má dve hlavné zložky, samotný vírus a C&C server, 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 zariadení pomocou protokolu telnet (na porte TCP 23 alebo 2323)

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

- **Osiris**

  Osiris je odvodený od malvéru Kronos, ktorý sa zameriaval na bankovnívctvo. Podobne ako Kronos, je Osiris modernejšou verziou bankového trójskeho koňa [13].

Táto verzia malvéru využíva na skrývanie metódu *process hollowing*. Umožňuje mu vydávať sa za legitímne procesy. Malvér sa šíri vydávaním sa za legitímny spustiteľný súbor (útoky zaznamenné s malvérom Osiris boli dokumenty Microsoft Word). Vydávanie sa za iný oficiálny softvér značne sťažuje identifikáciu malvéru a obmedzuje možnosti na zastavenie útoku [17]. Malvér v dokumetoch Word obsahoval aj makrá, ktoré po spustení stiahli další škodlivý malvér, ktorý umožuje zahĺtiť zariadenia alebo sťažiť detekcu

- **Loki**

  Loki je ďalšou variáciou staršieho malvéru Kronos. Rovnako ako Osiris, aj Loki využíva na svoje ukrytie metódu *process hollowing*. Loki sa zameriava na krádeže osobných údajov ako napr. 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.

Celkový prehľad použítých techník v súčasným vzorkách škodlivého kódu môžeme vidieť v tabuľke č.1.

	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 sme sa v tejto práci zaoberali je *process hollowing*. Nasledujúca kapitola sa venuje spôsobu akým sa malvér môže ukryť pomocou spomínanej techniky. Taktiež obsahuje potenciálne API funkcie pomocou ktorých môže byť technika *process hollowing* implementovaná.

2.1 Princíp

Princíp ukrytia malvéru, ktorý využíva *process hollowing* je do istej miery podobný technike *DLL Injection*. Hlavný cieľ metódy spočíva v ukrytí škodlivého kódu do bežného procesu, ktorý v systéme Windows spôsobuje čo najmenšie podozrenie [15]. Takýmto procesom môže byť napríklad *svchost.exe*, ktorý je v systéme bežne spustený aj vo viacerých inštanciách. *Process hollowing* vytvorí pozastavený proces *svchost.exe* (prípadne získa prístup už k bežiacemu procesu) a odmapuje jeho pamäť. Po odmapovaní alokuje dostatok miesta vo virtuálnom adresnom priestore procesu. Následne, po alokácii nakopíruje škodlivý kód a nastaví nový vstupný bod programu. Po ukončení týchto krokov obnoví pozastavený proces. Výsledkom je navonok bežiaci štandardný proces *svchost.exe*, ktorý ale vo vnútri vykonáva škodlivú činnosť. Schematické znázornenie priebehu tejto techniky môžeme vidieť na obrázku č.1.

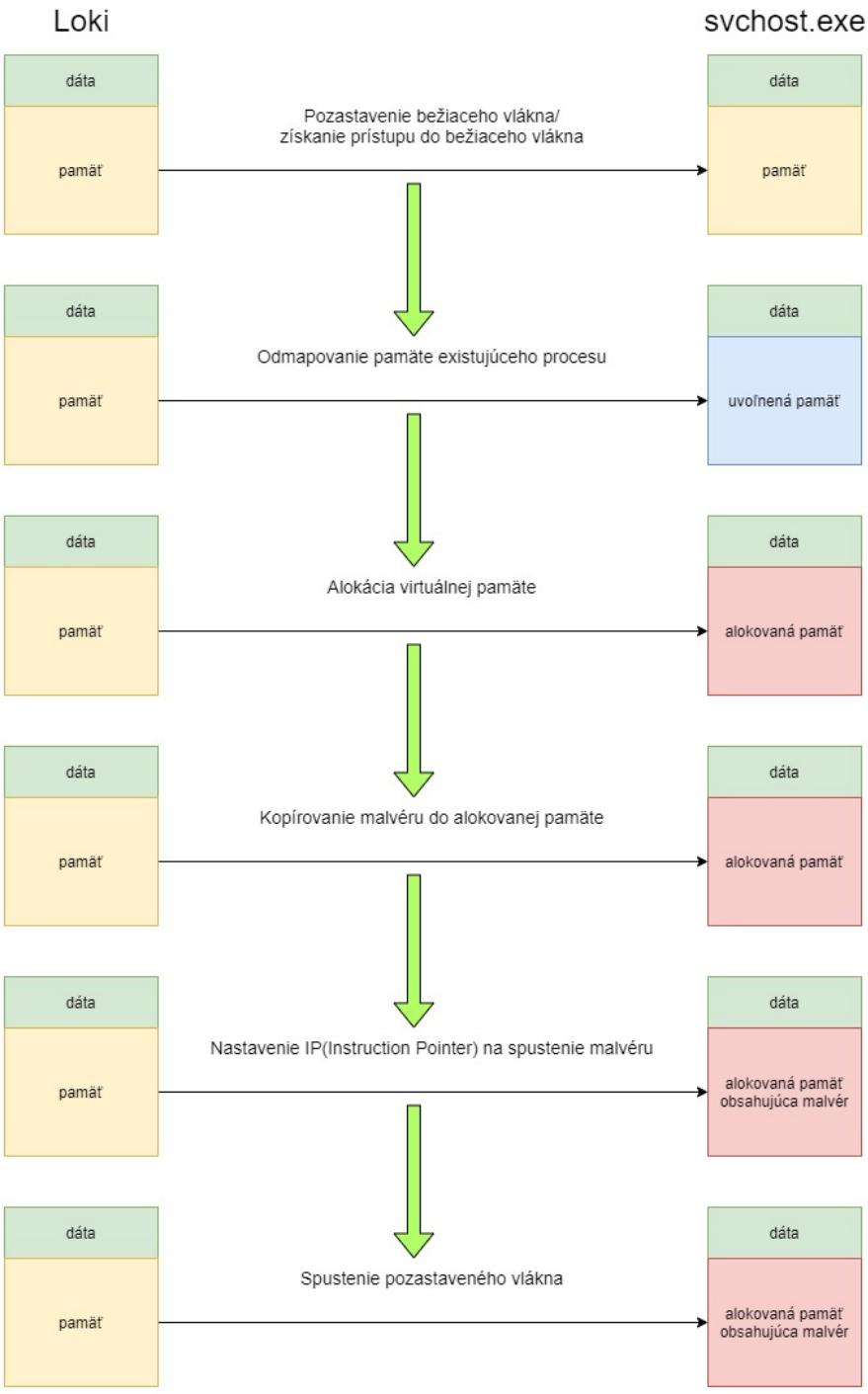
2.2 Využívané API funkcie

Process hollowing využíva na svoje fungovanie rôzne štandardné API volania. Nasledujúce nami vybrané API funkcie[8], môžu byť využité pri technike *rocess hollowing*:

- **CreateThread**
  Vytvorí nové vlákno vo virtuálnom adresnom priestore procesu, ktorý danú funkciu zavolal.

- **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, ako napr. nastavenie na ktorom procesore bude dané vlákno bežať.



Obrázok 1: Ukážka zmien adresného priestoru počas Hollowingu.

- **ResumeThread**
  Funkcia dekrementuje hodnotu, ktorá špecifikuje, koľkrát bolo vlákno pozastavané. V prípade ak sa hodnota dostane na nulu, vlakno je obnovené. V opačnom prípade ostáva pozastavené.

- **SuspendThread**
  Pozastavenie vykonávanie činnosti špecifikovaného vlákna.

- **SwitchToThread**
  Spôsobuje prepnutie aktuálneho vlákna, na vlákno, ktoré je pripravené bežať na procesore. Tento výber vykonáva operačný systém.

- **CreateProcessA** Vytvorí nový proces a jeho hlavné vlákno. Nový proces beží s rovnakými oprávneniami ako proces, ktorý danú funkciu zavolal.

- **VirtuallAlloc**
  Alokuje alebo mení oprávnenia stránok vo virtuálnom adresom priestore procesu, ktorý danú funkciu zavolal. Alokovaná pamäť je automaticky inicializovaná na nulu.

- **VirtualAllocEx**
  Funguje rovnako ako *VirtualAlloc*, s tým rozdielom, že alokuje pamäť v rámci virtuálneho adresného priestoru iného procesu.

- **WriteProcessMemory**
  Zapisuje údaje do pamäti v zadanom procese. Celá oblasť, do ktorej sa zapisuje, musí mať potrebné oprávnenia na zápis.

- **ReadProcessMemory**
  Číta údaje z virtuálneho adresného priestoru špecifikovaného procesu.

- **SetThreadContext**
  Nastavuje kontext (t.j. obsah registrov) pre špecifikované vlákno.

- **ExitThread**
  Keď sa táto funkcia zavolá, všetky udalosti aktuálneho vlákna sa zrušia. Podobne všetky čakajúce vstupno-výstupné udalosti iniciované vláknom sa zrušené a vlákno ukončí svoju činnosť.

- **NtUnmapViewOfSection**
  Odmapuje pamäť vybraného procesu.

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.

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. [7]

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. [3]

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.
  CreateThread, SuspendThread, CreateProcessA.

- **WAPI2** Predstavujú API primátrne určené na odmapovanie pamäte existujúceho procesu.
  NtUnmapViewOfSection.

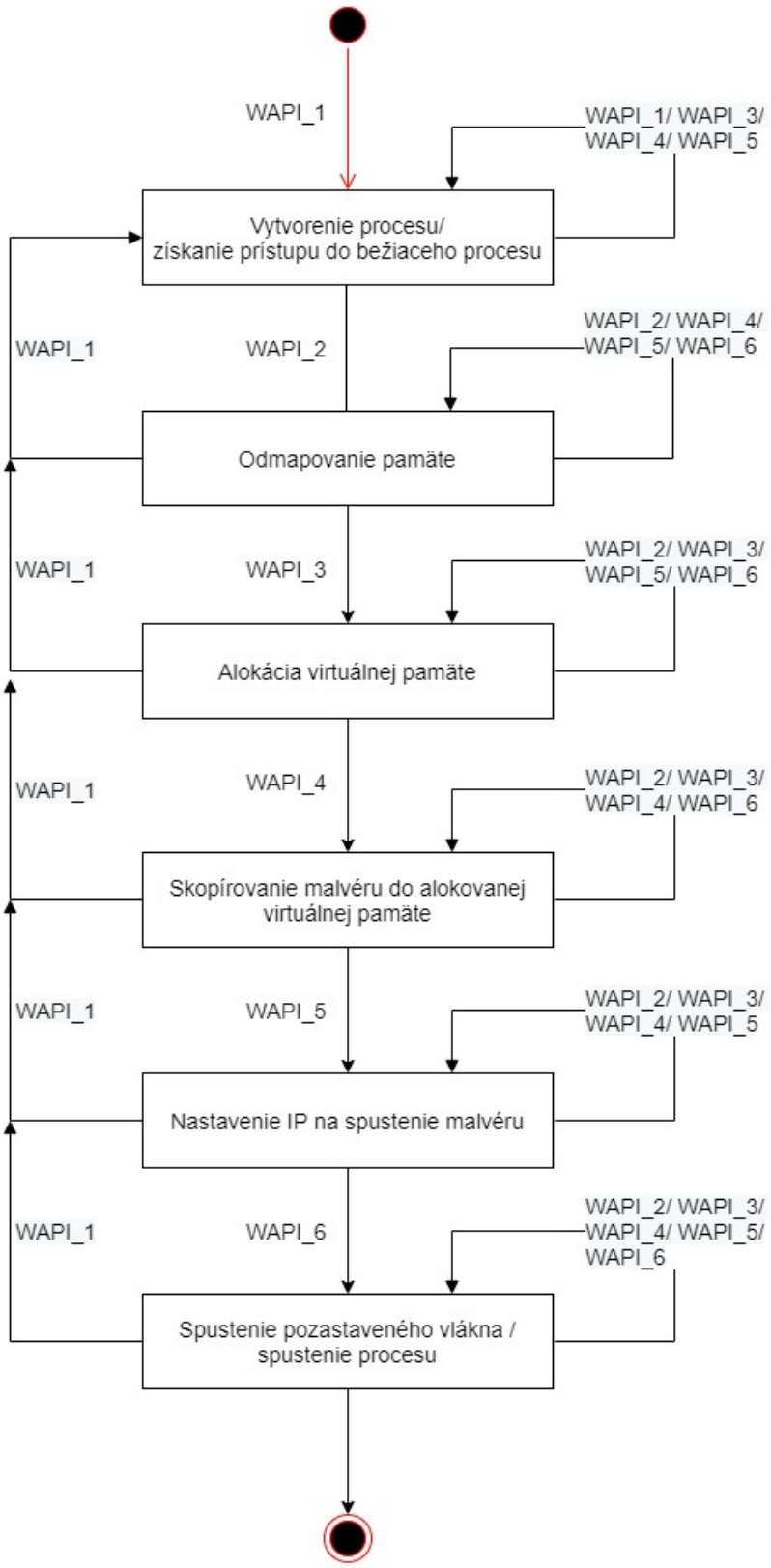
- **WAPI3** Predstavujú API, ktoré slúžia na alokáciu pamäte pre dané vlákno alebo rozširujú virtuálnu pamäť existujúceho procesu.
  VirtualAlloc, VirtualAllocEx.

- **WAPI4** Tieto funkcie umožňujú manipulovať s pamäťou. Ako je kopírovanie pamäte, zapisovanie do pamäte atď..
  WriteProcessMemory, CopyMemory.

- **WAPI5** Tieto API funkcie nastavujú kontext daného vlákna a umožňujú nastaviť *Instruction Pointer*, ktorý pri volaní spustí aj funkcionalitu malvéru.
  SetThreadContext

- **WAPI6** Poslednou skupinou sú API, ktoré spúšťajú pozastavené vlákno alebo ukončiť funkcionalitu aktuálneho vlákna.
  ResumeThread, ExitThread.

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.

4.1 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 jednotlivými 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 funkcií, ktoré by potenciálne mal najčastejšie využívať malvér využívajúci techniku *Process Hollowing* na ukrytie svojej činnosti.

4.2 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.

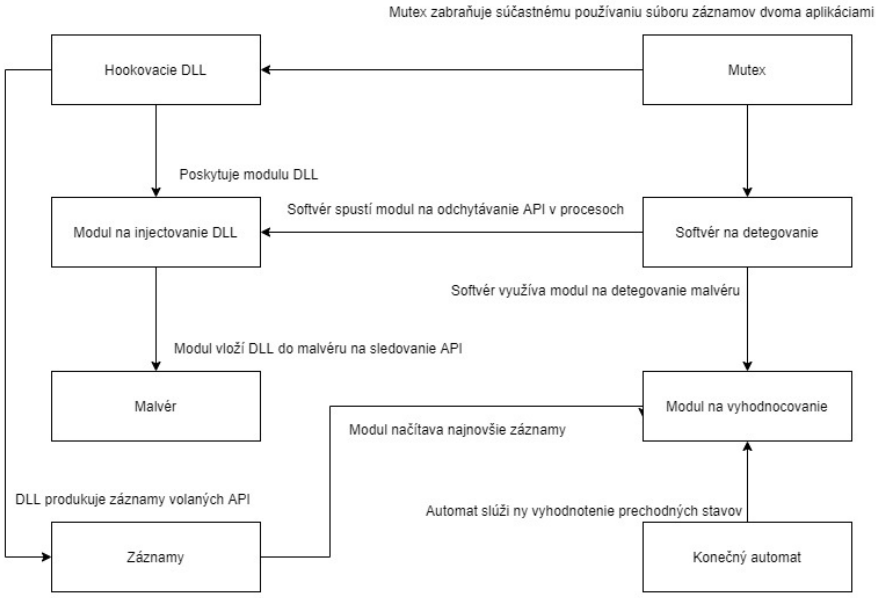
	Prechodové stavy						
Volané API funkcie	Štart	Vytvorenie procesu/ Získanie prístupu do procesu	Odmapovanie pamäte existujúceho procesu	Alokácia virtuálnej pamäte	Kopírovanie malvéru do alokovanej pamäte	Nastavenie IP na spustenie malvéru	Spustenie vlákna/ Spustenie pozastaveného procesu
Skratky stavov	S0	S1	S2	S3	S4	S5	S6
CreateThread	S1	S1	S1	S1	S1	S1	S1
CreateRemoteThread	S1	S1	S1	S1	S1	S1	S1
CreateRemoteThreadEx	S1	S1	S1	S1	S1	S1	S1
CreateProcessA	S1	S1	S1	S1	S1	S1	S1
CreateProcessW	S1	S1	S1	S1	S1	S1	S1
SwitchToThread	S1	S1	S1	S1	S1	S1	S1
OpenThread	S1	S1	S1	S1	S1	S1	S1
SuspendThread	S1	S1	S1	S1	S1	S1	S1
NtUnmapViewOfSection	S0	S2	S2	S3	S4	S5	S6
VirtualAlloc	S0	S1	S3	S3	S4	S5	S6
VirtualAllocEx	S0	S1	S3	S3	S4	S5	S6
CopyMemory	S0	S1	S2	S4	S4	S5	S6
WriteProcessMemory	S0	S1	S2	S4	S4	S5	S6
ResumeThread	S0	S1	S2	S3	S4	S6	S6
ExitThread	S0	S1	S2	S3	S4	S6	S6
SetThreadPriority	S0	S1	S2	S3	S5	S5	S6

Tabuľka 2: Matica prechodov medzi stavmi pri volaniach API.

5 Implementácia

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

Pre implementáciu riešenia sme zvolili programovaný jazyk C++, pretože primárnym testovavím prostredím budú najčastejšie používané systémy Windows. Na simuláciu testovanie prostredia použijeme **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.

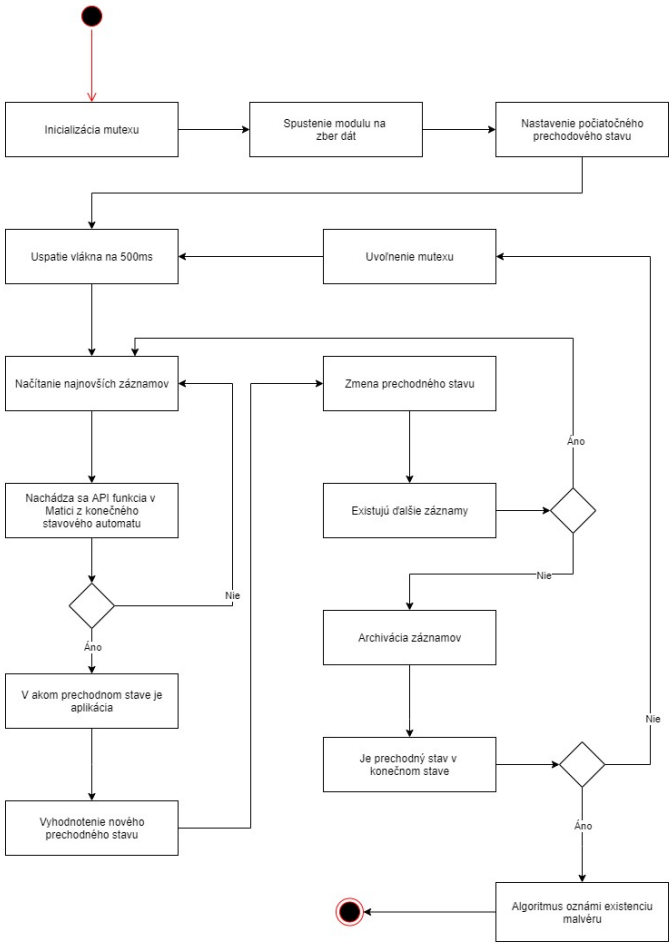


Obrázok 3: Moduly aplikácie

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 funkcií 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. Modulu 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 najnovšie 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 funkcií 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 originalFunkcion)
{
   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 <<
             ";" + originalFunkcion << endl;
         myFile.close();
      }
```

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

Výpis č. 1 Implementácia funkcie writeFunctionToFile.

Druhá časť aplikácie už slúži len na nahradenie pôvodnej volanej API funkcie, modifikovanou tou istou funkciou, ktorá je rozšírenia o zapisovanie volanej API do súboru. Pôvodnú funkciu *SetThreadContext* určenú na nastavenie *Instriction Pointer* v tomto module nahradíme funkciou *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 Definícia API volania SetThreadContext.

```
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 Nahradenie pôvodnej funkcie SetThreadContext.

5.3 Detours

Detours je knižnica určená na monitorovanie a inštruovanie API volaní v systéme Windows. Táto knižnica 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. Knižnica sa aplikuje dynamicky za behu programu. Detours nahrádza pokyny cieľovej funkcie skokom na funkciu zadanú používateľom. Kód cieľovej funkcie je modifikovaný v pamäti, nie na disku, čo umožňuje zachytávanie binárnych funkcií. [1] Po načítaní do procesu môže knižnica DLL obísť akúkoľvek funkciu v procese, či už v aplikácii alebo v systémových knižniciach, ako sú napríklad rozhrania Windows API.

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 Inicializácia mutexu.

6 Výsledky

Stručný úvod do testovania nami vytvoreného riešenia/riešení. Zoznam experimentov. Našu aplikáciu budeme testovať vo *Virtual Boxe*, na systéme Windows 10. Aplikáciu budeme testovať na reálych vzorkách malvéru, ktoré využívajú *Process Hollowing* na ukrytie svojej činnosti.

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.

Zoznam použitej literatúry

[1] Detours. https://github.com/microsoft/Detours. [Online; cit. 2020-05-15].

[2] Emotet. https://www.malwarebytes.com/emotet/. [Online; cit. 2020-04-01].

[3] HollowFind. https://github.com/monnappa22/HollowFind. [Online; cit. 2020-04-20].

[4] Loki Number Seven - Loki Malware Keeps Stealing Your Credentials. https://www.cyberark.com/resources/threat-research-blog/loki-number-seven-loki-malware-keeps-stealing-your-credentials/.

[5] Malvér. https://www.eset.com/sk/malver/. [Online; cit. 2020-04-01].

[6] McAfee ATR Analyzes Sodinokibi aka REvil Ransomware-as-a-Service - What The Code Tells Us. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-atr-analyzes-sodinokibi-aka-revil-ransomware-as-a-service-what-the-code-tells-us/. [Online; cit. 2020-04-01].

[7] PHDetection. https://github.com/idan1288/PHDetection. [Online; cit. 2020-04-20].

[8] Processthreadsapi.h header - Win32 apps. https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/. [Online; cit. 2020-05-15].

[9] Ten Process Injection Techniques: A Technical Survey Of Common And Trending Process Injection Techniques. https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process. [Online; cit. 2020-04-05].

[10] What is the Mirai Botnet? https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. [Online; cit. 2020-04-05].

[11] Zeus Virus. https://usa.kaspersky.com/resource-center/threats/zeus-virus. [Online; cit. 2020-04-01].

[12] Cimpanu, C. Emotet, today's most dangerous botnet, comes back to life. https://www.zdnet.com/article/emotet-todays-most-dangerous-botnet-comes-back-to-life/. [Online; cit. 2020-04-05].

[13] Lewis, N. What new technique does the Osiris banking Trojan use? https://searchsecurity.techtarget.com/answer/What-new-technique-does-the-Osiris-banking-Trojan-use. [Online; cit. 2020-04-01].

[14] Osborne, C. New Dridex malware strain avoids antivirus software. https://www.zdnet.com/article/new-dridex-malware-strain-avoids-antivirus-software-detection/. [Online; cit. 2020-04-01].

[15] Salem, E. Astaroth malware uses legitimate os and antivirus processes to steal passwords and personal data. https://www.cybereason.com/blog/information-stealing-malware-targeting-brazil-full-research/. [Online; cit. 2020-05-05].

[16] Sikorski, Michael and Honig, Andrew. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. USA. 1st. No Starch Press.

[17] Yaroslav Harakhavik, N. F. Osiris: An Enhanced Banking Trojan. https://research.checkpoint.com/2018/osiris-enhanced-banking-trojan/. [Online; cit. 2020-04-01].

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