



**GÁBOR
DÉNES
EGYETEM**

MÉRNÖKINFORMATIKUS

SZAKDOLGOZAT

**KÁRTÉKONY KÓDOK MŰKÖDÉSÉNEK
TANULMÁNYOZÁSA STATIKUS ÉS DINAMIKUS
MÓDSZEREKKEL**

Maklári Péter

53/2024

Tartalomjegyzék:

1.	Bevezetés	6
1.1.	A kártékony kód fogalma.....	6
1.2.	Terjedésük	6
1.2.1.	Social Engineering	7
1.2.2.	Sérülékenység	7
1.2.3.	Hibrid terjedés.....	8
1.3.	Kártékony szoftver fejlesztői és támogatói	8
1.4.	Védekezés a kártékony kódok ellen.....	8
1.4.1.	Vírusirtók- és védelmi szoftverek	9
1.4.2.	Tűzfalak és hálózati védelem	9
1.4.3.	Az IDS és IPS	9
1.4.4.	Biztonsági frissítések és sérülékenységkezelés	9
1.4.5.	Felhasználói oktatás és tudatosság növelése	10
1.4.6.	Kibervédelmi stratégia és protokoll.....	10
1.5.	Kártékony kód elemzést használó szerepkörök.....	10
1.5.1.	Malware analyst	11
1.5.2.	Cybersecurity analyst	11
1.5.3.	Digital forensics investigator	11
1.5.4.	Vulnerability Researcher	12
2.	Kártékony kódok története és fajtái.....	13
2.1.	Kártékony kódok alfajtái.....	13
2.1.1.	Vírus	13
2.1.2.	Férgek	13
2.1.3.	Trójai szoftver	14
2.1.4.	Zsarolószoftver.....	14
2.1.5.	Reklámszoftver	14
2.1.6.	Rootkit és bootkit	14
2.1.7.	Kémszoftver.....	15

2.1.8.	Fájl nélküli kártékony szoftver.....	15
2.1.9.	Bányászszoftver	15
2.1.10.	Hibrid kártékony szoftverek	15
2.2.	Kártékony kódok története.....	15
2.2.1.	Fogalom születése és fejlődés	15
2.2.2.	Az első kártékony kódok	16
2.2.3.	ILOVEYOU.....	16
2.2.4.	STUXNET	17
2.2.5.	WannaCry zsaroló szoftver.....	18
2.2.6.	NotPetya	19
2.2.7.	NSO Pegasus.....	19
3.	Elemzéshez használt eszközök bemutatása.....	21
3.1.	Virtuális gépek felépítése és konfigurációja	21
3.1.1.	Virtualizációs szoftver	21
3.1.2.	FlareVM.....	21
3.1.3.	REmnux.....	22
3.1.4.	Virtuális hálózat	22
3.2.	Elemzéshez használt szoftverek	24
3.2.1.	Detect It Easy	24
3.2.2.	Ghidra.....	24
3.2.3.	x64dbg/x32dbg.....	24
3.2.4.	Wireshark.....	24
3.2.5.	INetSim.....	25
3.2.6.	AnyRun.....	25
4.	Statikus és dinamikus elemzés	26
4.1.1.	Statikus elemzés bemutatása.....	26
4.1.2.	Dinamikus elemzés bemutatása	26
4.2.	PE fájl alapok az elemzéshez	27
4.2.1.	DOS fejléc	27

4.2.2.	Import szekció:	28
4.3.	Statikus elemzés	28
4.3.1.	Azonosítás	28
4.3.2.	Virus Total	29
4.3.3.	Saját elemző script:	29
4.3.4.	Detect It Easy elemzés	32
4.3.5.	Elemzés és visszafejtés Ghidra-val	33
4.4.	Dinamikus elemzés	37
4.4.1.	AnyRun	37
4.4.2.	x32dbg és Wireshark	37
5.	Kódolás és titkosítások elemzése	40
5.1.	Titkosítás	40
5.1.1.	Szimmetrikus titkosítás	40
5.1.2.	Aszimmetrikus titkosítás	41
5.1.3.	Titkosítás összegzés	41
5.2.	Kódolás	41
5.2.1.	Base64 kódolás	42
5.3.	Hashelés	42
5.4.	Obfuszkáció	42
5.4.1.	Fájlformátum hamisítás	42
5.4.2.	Zaj generálás	43
6.	Összegzés	44
6.1.	Elemzési eredmények	44
6.2.	Következő lépések	44
6.2.1.	A python script tovább fejlesztése	44
6.2.2.	Patch készítés a teljeskörű dinamikus elemzéshez	44
6.2.3.	YARA szabály készítése	44
6.3.	Záró gondolatok	45
7.	Hivatkozások	46

8.	Ábrajegyzék	51
9.	Mellékletek	52
	1. sz.melléklet „PrivateLoader.json” [49].....	52
	2. sz.melléklet „RisePro.json” [49]	58

1. Bevezetés

Amikor kitaláltam, hogy mérnökinformatikai szakot szeretnék elvégezni, akkor már a kezdeteknél eldöntöttem, hogy elsősorban főleg kiberbiztonsággal szeretnék foglalkozni. Amióta a főiskolára járok rendszeresen (heti szinten) követem a kiberbiztonsági szakmával kapcsolatos híreket és a témakörhöz kapcsolódó számos tartalmat fogyasztok. Ezeknek köszönhetem azt is, hogy jelenleg is van lehetőségem az iparágban tudok dolgozni. A kitűzött célom, hogy egy támadásokat és fenyegetéseket szimuláló cégnél helyezkedhessek el. Szeretnék a jövőben kártékony szoftverek fejlesztésével és biztonsági rések fejlesztésével foglalkozni azért, hogy az informatikai rendszerek biztonságosabbá tételéhez hozzá tudjak járulni.

A bevezetőben még szó lesz a kártékony kódok fogalmáról, a terjedésükről, a kiberbiztonság gazdasági jelentőségéről, valamint a kártékony kódok fejlesztőiről és ezen személyek támogatóiról. Említésre kerülnek védekezési rendszerek és módszerek. Ezek mellett említésre kerül az is, hogy a szakma mely területei tudhatják a kártékony kód elemzést az eszköztárunkban.

1.1. A kártékony kód fogalma

Kártékony kódnak nevezzük azokat a szoftvereket, amelyeknek a hatásai vagy felhasználóra, vagy a számítógépre nézve ártalmasak. Egy kártékony kód a működése folyamán megsérti a kiberbiztonság alapjait ezzel a lehetséges alanyra vagy hardverre negatív hatást gyakorol. A három alapelv a bizalmasság, sértetlenség és rendelkezésre állás. A bizalmasság megsértése valamilyen adatszivárgással jár, lehet például személyes adat vagy intellektuális tulajdon. A sértetlenség vagy integritás megsértése az információs rendszeren tárolt adatok megváltoztatása, törlése esetleg hamis adatokkal való megtöltése. Az rendelkezés állás megsértése pedig üzleteknél kimaradással járhat, ami anyagi károkat okoz. [1]

A veszteségeknek van egy fizikális és egy nem fizikális oldala is. Fizikális oldalról lehet például egy számítógép, szerver vagy hálózatban okozott kár, amely akár közvetlen a hardverre fejthet ki negatív esetleg maradandó hatását is. Másik oldalról a nem fizikális kár információ vesztése vagy eltulajdonítása, vagy megsemmisítése szerepében, illetve ide lehet sorolni egy rendszer elérhetetlensége miatt okozott reputációs károkat is.

1.2. Terjedésük

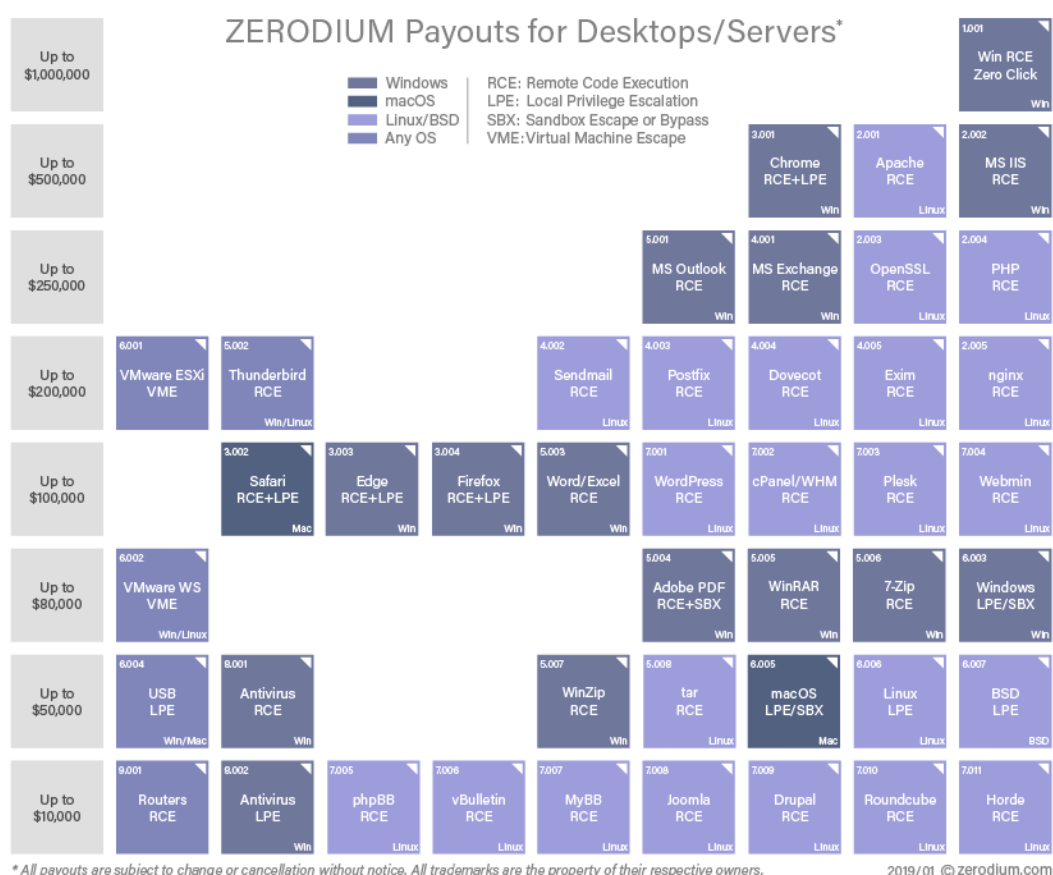
A kártékony kódok terjedésének két típusa van. Mindkét típus ugyan annyira veszélyes és feltételezhetően, hasonlóan nagy közönséget lehet velük elérni. A terjedés szempontjából is jelen lehet egy hibrid terjedés.

1.2.1. Social Engineering

Első és gyakoribb a felhasználó valamilyen úton való manipulálásával történik. Ezt angolul social engineering-nek nevezik. Az alkalmazáshoz nem szükséges semmiféle technológiai ismeret, a technika a pszichológiai manipuláción alapszik. Ezen típusú technikára példa egy a phishing, avagy adathalászat, ahol a felhasználót bele csalják egy fertőzött fájl letöltésébe egy emailen keresztül, de lehet oly módon tévesztik meg a felhasználót, hogy egy fakes and pretexting technikával elérik, hogy egy távoli asztali alkalmazással hozzáférést szerez a támadó és így lehetőségük van telepíteni kártékony kódokat. [2]

1.2.2. Sérülékenység

A másik típusa a kártékony kódok terjedésének a különböző sérülékeny szoftveken keresztül történik. Itt előfordulhat, hogy az áldozat nem telepítette fel a legújabb biztonsági frissítéseket, de az is lehetséges, hogy a fertőzést egy addig ismeretlen sérülékenységen, avagy nulla napos sérülékenységen (zero day vulnerability) keresztül kapta el. Ezeknek a sérülékenységeknek van egy piaca, ahol az exploit-nak a komolysága meghatározza annak árát. Az következő ábra megmutatja, hogy milyen átlagárral rendelkezik egy-egy ilyen sérülékenység a piacon. [3]



1. ábra Sérülékenységek ára [4]

1.2.3. Hibrid terjedés

Hibrid módon is terjedhetnek ilyenkor az áldozat a felhasználó egy linkre kattintás után egy kártékony weboldalra lesz irányítva. Itt a weboldal kihasználhat egy sérülékenységet, amivel képes lehet kitörni a böngésző által szolgáltatott „sandbox” környezetből.

Egy másik lehetőség lehet, hogy a szoftver letöltése és futtatása után, ami a felhasználói manipulálás része a támadásnak, a kód tartalmaz valamilyen sérülékenységet és ezáltal a hálózat többi számítógépét is megfertőzi.

1.3. Kártékony szoftver fejlesztői és támogatói

Káros kódokat lehet jó, illetve rossz indulattal fejleszteni, minden csak a végtermék felhasználásától és célpontjától függ. Jó felhasználása az ilyen kódoknak például kártékony kód elemszámok tanulási és gyakorlási anyagbiztosítására alkalmazható. Ezzel feltudják készíteni magukat egyes valós esetekre, ahol a szükséges módszereket gyakorolták már. Ezek mellett a szakmában vannak olyan szolgáltatások, ahol egyes cégek olyan fenyegetés szimulációt csinálnak, ahol egy ilyen kártékony kódot egy teszt környezetben szabadjára engednek, hogy a környezet kapacitását feltudják mérni. Rossz felhasználása pedig lehet az ilyen szoftvereknek, ahol vagy egy független hacker csapat vagy egy ország által támogatott csapat anyagi haszonszerzés céljából támadja meg áldozatait. Ilyen támadások hatalmas károkat okozhatnak akár kritikus infrastruktúrákon. Fontos megjegyezni egy érdekes trendet, ami szerint manapság egyes nemzetközi alvilági maffia szervezetek is elkezdtek pénzt fektetni a kiberháborúba és saját kiberbűnözéssel foglalkozó divíziókat üzemeltetni.

Mivel a mai világban a kiber bűnözés egy több mint 10 milliárd amerikai dolláros üzletág, ahol az átlagos adatszivárgásnak az ára a 4 millió amerikai dollárt meghaladja, ezért a cégeknek fontos, hogy a kiberbiztonságuk megerősítése. Ezzel könnyen eltudják kerülni az ismert támadásokból fakadó kártékony kód általi megfertőződést.

A világ vezető hatalmai is előszeretettel fektetnek be rengeteg pénzt magába a kiber fegyverkezésbe. A NATO a 2000 évek vége és a 2010 es évek eleje fele elfogadta a kibertér fontosságát és a kibertér hadviselését, mindezek mellett politikai álláspontot is foglalt ezzel kapcsolatban. Ezen államok közé tartozik az USA, Oroszország, Kína, Izrael, Irán és még más országok. Az előbb említett országok között folyamatos a kiberháború. [5] [6]

1.4. Védekezés a kártékony kódok ellen

A következő részben bemutatom, hogy milyen módszerekkel, lehet a kártékony szoftverek ellen védekezni. Ezek a bemutatások nem teljeskörűek, csak megemlítsre kerülnek.

1.4.1. Vírusírtók- és védelmi szoftverek

Az antivírus- és védelmi szoftverek olyan programok, amiknek a számítógép megvédése a feladata. Az ilyen programok folyamatosan figyelik a hálózati kommunikációkat a számítógépen és ellenőrzik az általunk használt fájlokat, gyanús folyamatokat, valamint a rendszerkomponenseket is. Érdemes itt megemlíteni, hogy a két fogalom az antivírus és a védelmi szoftverek között különbség, ami pedig, hogy az antivírus jelentéséből fakadóan csak vírusok ellen véd, amíg a védelmi szoftverek más kártékony kódok elleni is, viszont a mai világban a laikusok között is az elterjedt szóhasználat az antivírus, mivel a 90-es években a média kőbe véste a fogalmat. Ezzel az a probléma, hogy a napjainkban a vírusok, mivel nem fejlődtek az elmúlt évtizedekben, ezért nem igazán számot tevő fenyegetettség. Ezért az általunk antivírusnak nevezett szoftverek, nem csak vírusok ellen védenek, hanem más kártékony kódok ellen is. [7]

1.4.2. Tűzfalak és hálózati védelem

A tűzfal egy olyan a hálózat biztonságát elősegítő eszköz, ami a korlátozza a be- és kimenő forgalmat a szegmensen. Ez lehet szoftveres vagy hardveres is és elhelyezkedhetnek az OSI modell különböző rétegeiben is. A tűzfalak monitorozzák a forgalmat és szűrőszabályok szerint engedélyezik vagy blokkolják a csomagokat. Az ilyen szűrőszabályokat meg lehet fogalmazni például portokra, IP címekre, protokollokra és egyéb dolgokra is. [8]

1.4.3. Az IDS és IPS

Az IDS, avagy Intrusion Detection System egy olyan rendszer, ami a behatolásokat érzékeli azáltal, hogy a hálózaton belül történő az oda nem illő gyanús forgalmat keresi. Ezeket az anomáliákat ismert fenyegetettségekkel hasonlítja össze, hogy IoC-eket fedezzen fel. Két fajta IDS rendszer szeretnék megemlíteni, az egyik az HIDS és az NIDS. Az első, amit szeretnék röviden ismertetni az a host alapú IDS. Ez általában egy szoftver alapú védelmi rendszer, ami a hálózaton található eszközökre felvan telepítve. Ez az eszközökről érkező log-okat elemzi. Az NIDS egy hálózat alapú behatolás érzékelő rendszer ez egy hardver alapú modellel dolgozik, ahol egy külön eszköz elemzi a forgalmat. Az IPS rövidítés az Intrusion Prevention System egy olyan rendszer, ami kiegészíti az IDS funkcióit azzal, hogy nem csak passzívan megfigyeli a rendszert és értesítéseket generál egy szakembernek elemzésre, hanem aktívan reagál az egyes eseményekre. [9] [10] [11] [12]

1.4.4. Biztonsági frissítések és sérülékenységkezelés

A biztonsági frissítések és a sérülékenységkezelés fontos részét képezik a kiberbiztonsági stratégiának. A szoftverek és operációs rendszerek gyártói folyamatosan kiadnak frissítéseket és javításokat, hogy azonosítsák és kijavítsák a biztonsági réseket és sérülékenységeket. A

rendszergazdáknak és felhasználóknak fontos, hogy rendszeresen frissítsék a szoftvereket és alkalmazzák a legújabb biztonsági patch-eket a védelmi szint fenntartása érdekében.

1.4.5. Felhasználói oktatás és tudatosság növelése

A cégek számára a felhasználók oktatása és a tudatos internetezés egy kulcsfontosságú pontja a kártékony kódok elleni védekezésben hiszen az ilyen szoftverek túlnyomótöbbségben social engineering technikákkal terjednek. A felhasználók az elsőszámú célpontjai nagyon sok támadásnak, mivel sokkal könnyebben tud a támadó rávenni egy embert arra, hogy csináljon valamit amint nem kéne, mint egy számítástechnikai eszközt. Egy rossz kattintásnak rettenetesen súlyos következményei lehetnek és ezzel minden felhasználónak tisztába kéne lennie. Ezért is fontos, hogy felhasználóinkat oktassuk arra, hogy hogyan lehet felismeri az adathalászatot, illetve hogyan tudnak biztonságosan internetezni. Ez nem csak a cég szempontjából fontos, hanem mindenki számára, aki a világhálón tevékenykedik. [13]

1.4.6. Kibervédelmi stratégia és protokoll

A kibervédelmi stratégia és a hozzátartozó protokollok meghatározása fontos szerepet játszanak a kibertámadások sikeres elhárításában. A kiberbiztonságnak ezen része, ami a szabályzással és az ezeknek való megfeleléssel foglalkozó szakirányt Security Governance-nek nevezik.

A kibervédelmi stratégia magába foglalja a, hogy a vállalatnak milyen céljai, illetve folyamatai vannak ezen a területen. Az öt szintje egy stratégiának a következő. Először is képeseknek kell lenni észlelni és azonosítani a fenyegetéseket. Másodszor muszáj, valamilyen jól működő technikai védelmi rendszerrel rendelkeznie a vállalatnak, amivel ezek ellen a fenyegetések ellen fel tud lépni. Harmadszor az előzőkből származó kockázatokat kezelni kell. Szükséges még, hogy incidensekre képesek legyünk reagálni, hogy a vállalatra kifejtett hatást minimalizálni lehessen. Utoljára muszáj rendelkezni egy visszaállási folyamattal egy lehetséges adatszivárgás esetén.

A stratégia részei protokollok, amiket egyes esetekhez készítenek, hogy legyen egy útmutatás, hogy mik a megfelelő lépések a helyzetben. [14]

1.5. Kártékony kód elemzést használó szerepkörök

Ebben a részben arról fogok beszélni, hogy a kiberbiztonsági szakmában mely néhány olyan szerepkör van, ami közvetve találkozik kártékony szoftverekkel és vagy azok elemzésével, nem csak döntéshozási szempontból, hanem esetleg olyan rendszerekkel kerül kapcsolatba, amelyek fertőzöttek vagy potenciálisan fertőzöttek.

1.5.1. Malware analyst

Ez a szerepkör a nevéből adódóan nap mint nap különböző káros szoftvereket elemez és fejt vissza annak érdekében, hogy más kiberbiztonsági szakértők tudjanak védekezni a legújabb fenyegetettségekkel szemben.

Munkájuk során egyes kódmintákat kell elemezniük, ezt visszafejtéssel teszik, és a végső céljuk, hogy a kompromitálhatóságra mutató jeleket dokumentálják és az érintett személyeket értesítsék. Erre a visszafejtés során megtalált IP címek blokkolásával, valamint a kártékony szoftverekről készített szignatúrák letiltásával tudnak gondoskodni.

1.5.2. Cybersecurity analyst

A szakmában nagyon sok dolgozóra aggatják rá ezt a szerepkört, ezért egy kifejezetten sok feladattal rendelkezhetnek a kiberbiztonsági elemzők. Fő feladatuk, hogy a kibertámadásokat észleljék és elemezzék és az ebből keletkező incidenseket vizsgálják és reagáljanak rá. Ilyen helyzetekben szükségük van a digitális bizonyítékok megszerzésére és azok elemzésére. Némely incidenst kártékony kód vagy annak gyanúja okoz, ezért szükséges lehet ezen szoftverek tanulmányozása. Mivel ezek a szoftverek hatalmas pénzvesztést okozhatnak a cégnek, ezért nagy prioritást élvez a kezelésük és ezekből fontos leszűrni a tanulságokat, hogy képesek legyünk megerősíteni a védelmi stratégiájukat, valamint, ha szükséges akkor finomíthatnak az incidensek kezelésére alkalmazott folyamatokon.

1.5.3. Digital forensics investigator

A Digital forensics investigator egy olyan szakértő, aki jogi és informatikai szakterületen is szakértelemmel rendelkezik, egy a digitális térben nyomozást folytató személy. Egy független szakértő, akinek a feladata az ilyen kiberbűnözési helyszíneken bizonyítékokat gyűjtsön némely esetben helyre is állítson, ha az ilyen evidenciák károsítása történt. Ezeket bármilyen számítástechnikai eszközről kinyerheti lehetnek ezek szimpla személyi számítógépek, okostelefonok, szerverek vagy IoT eszközök.

Mindezek mellett, amit talált azt tanulmányoznia is kell például log-ok elemzésével, hogy utána azt a megbízónak prezentálhassa. Ilyen szakembereknek a tudását a privát-, illetve a közszférában is. A privátszférában olyan cégek alkalmazhatják, akik súlyos kibervédelmi incidenseket szenvednek és tanácsadásra szorulnak. Más esetekben ügyvédi irodák is alkalmazhatják ezen szakembereket. A közszférában, pedig a rendőrségi nyomozásokban vehet részt, hogy vagy kiberbűnözőket utáni nyomozásban vagy más bűnözők a kibertéren keresztül történő megtalálásában nyújtson segítsen. Ezek mellett az igazságszolgáltatásban is jelen lehet törvényszéki szakértőként, akinek feladata, hogy könnyen fogyasztható formában elmagyarázza egy nem szakembernek azokat a tényeket, amik segítik az eset lezárását.

1.5.4. Vulnerability Researcher

Ez a szerepkör egy igen fontos, hiszen ezek az emberek a hétköznapiakban több módszerrel keresnek a gyártók által ismeretlen sérülékenységeket. Az ehhez alkalmazott egyik eszközt „Fuzzing” -nak nevezik, ahol a programnak a bemeneti értékeit tesztelik véletlenszerűen generált értékekkel. Ezek az értékek nem megfelelően vannak megformázva és ezáltal próbálnak valamilyen program meghibásodást előidézni.

Egy másik módszer lehet, hogy nagyon friss káros kódok elemzésével, mivel ezek tartalmazhatnak ilyen kódolási vagy tervezési hibákat, ami kiberbiztonsági szempontból kockázattal rendelkezhet. Ilyen helyzetekben gyakorta együtt dolgoznak olyan személyekkel, akik vagy többet foglalkoznak kártékony kódok elemzésével, vagy olyanokkal emberekkel, akiknek nagyobb tapasztalatuk van a visszafejtésben.

A kutatók a szakmában egy nagyon fontos szerepet töltenek be és nagyon magas technikai tudással kell rendelkezni, ahhoz, hogy ilyen munkát végezhessünk.

2. Kártékony kódok története és fajtái

A fejezet a kártékony szoftverek fajtáit és történetét fogja bemutatni, hogy egy átfogó nézőpontot kapjunk róluk.

2.1. Kártékony kódok alfajtái

Először is le kell fektetni azt, hogy ezen alfajták közül az utolsó alfajta az, ami igazán megmutatja, hogy milyenek a kártékony kódok a mai világban az internet dzsungelében. A többi alfaj ezen kódoknak egyes tulajdonságainak alkategóriája. Ezen alkategóriák definiálhatják azt, hogy hogyan terjed a kód például mire van szüksége, hogy ezt megtehesse, vagy leírhatja azt, hogy milyen módon fertőződött meg a rendszer. Azt a jellemzőjét is megszabhatja, hogy mit csinál a rendszeren, avagy rendszerrel és azt is, hogy ott mit támad.

Egy másik szempont, ami alapján fel lehet osztani kártékony kódokat, amit a fejezet nem tárgyal külön fajtaként az pedig a következő, hogy milyen operációs rendszert vagy platformot támad. Nem kevés hírt hallunk manapság arról, hogy a kártékony kódok nem csak a hétköznapi számítástechnikai eszközöket támadják, hanem IoT eszközöket is, mivel ezen eszközöknél könnyen előfordulhat, hogy egy ma már nem támogatott rendszer komponensét használnak, ami sérülékeny ezáltal könnyen beépíthetik egy botnet hálózatba.

2.1.1. Vírus

A vírus egy olyan káros kód, ami saját magát képes replikálni de ehhez szüksége van egy másik fájlra, hogy megtegye. Ezzel apránként elhelyezkedve a fertőzött rendszer minden pontján feltehetőleg a boot szektorban is. Fontos megérteni, hogy a vírusok csak egy típusa a kártékony kódoknak, és mint ilyenek, részei egy szélesebb körű fenyegetéseknek a kiberbiztonság területén.

2.1.2. Férgek

Számítógépes férgeknek nevezzük azokat a kártékony kódokat, amelyek tudják magukat replikálni és nincs szükségük ehhez egy másik fájlra, ahova beinjektálódnak. Ezen típus gyakorta sérülékenységeket használ ki és nem csak egyes számítógépeket fertőz meg, hanem teljes hálózati rendszereket.

Nem minden esetben okoznak kárt egy fertőzésnél. Egy féreg képes megfertőzni egy gépet és onnan tovább fertőzni a hálózatot, de ha az első fertőzött gép nem rendelkezik a funkcionalitással vagy a féreg által keresett adatokkal, akkor szimplán csak arra használja, hogy önmagát tovább terjessze.

2.1.3. Trójai szoftver

Trójai vírusnak nevezzük azokat a kártékony kódokat, amik valami legitim szoftvernek adják ki magukat azért, hogy feltelepítsük és ezzel megfertőzzük gépünket. Miután ezt engedélyeztük neki ő képes változtatni mind az operációs rendszer-, mind az felhasználó által birtokol fájlokon.

2.1.4. Zsarolószoftver

Zsaroló szoftvernek nevezzük azon kártékony szoftvereket, amelyek a megtámadott rendszeren tartott fájlokat titkosítja, és ennek feloldásáért pénzt kér. Az ilyen szoftverek általában úgy vannak megírva, hogy ne csak az áldozat számítógépén tárolt fájlokat károsítsa, hanem a hálózaton található összes ilyen eszközt is rabul ejthesse. Jelenleg ez a fajta kártékony kód a legnépszerűbb és a leggyakoribb. Ezt lehet egy csomagban szolgáltatás ként is megvásárolni, ahol nem szükséges se a kódot elkészíteni, se a hozzá szükséges infrastruktúrát felépíteni vagy üzemeltetni. Ilyenkor kapunk egy vezérlő felületet, ahonnan minden támadást egyszerűen tudunk irányítani.

Ha ilyen támadás ér minket szinte lehetetlen az adat mentés, mivel általában aszimmetrikus titkosítással lesznek titkosítva az adataink. Az egyetlen remény ilyen esetben, hogyha a szoftver elemzés közben kiderül, hogy hibát ejtett a fejlesztő és ezáltal valamilyen ilyen esetben lehetséges, hogy még a memóriából ki lehet olvasni a titkosításhoz használt kulcsot.

2.1.5. Reklámszoftver

Az olyan vírusokat nevezzük reklám vírusnak, amik kényszerűen termékeket reklámoznak nekünk. Gyakran felugró ablakkal teszik a felhasználó életét kellemetlenné. Ezen vírus gyakran a fejfájás okozása mellett nem igazán okoz kárt, ha nem társul mellé valami más funkcionalitás.

2.1.6. Rootkit és bootkit

A rootkit egy olyan típusú kártékony kód, amit a támadók azért hagynak egy számítógépen, hogy később képesek legyenek visszatérni. Ezen szoftverek gyakorta a számítógépünk legmagasabb jogosultságát célozzák meg a kernel szintet. Ezzel a privilégiummal képesek arra, hogy minden újraindításkor elinduljon és meg nyissa a csatornát az eszközön. Ilyen típusú szoftverek fejlesztőinek a főcélja, hogy egy rendszer adminisztrátor ne találjon bizonyítékot vagy fertőzésre utaló nyomot egy lehetségesen kompromittált rendszerben. [15]

A bootkit annyiban különbözik az előbb említett típustól, hogy a kernel helyett eggyel mélyebbre az operációs rendszert futtató szoftverbe a BIOS vagy UEFI rendszerbe injektálja magát. Ilyen esetekben nem elég az operációs rendszer újra telepíteni hiszen az új rendszer is fertőzött lesz. Ezen esetekben két megoldás lehetséges, vagy újra telepítjük az alaplapunk firmware-ét vagy be kell szerezniünk egy teljesen új hardvert. [16]

2.1.7. Kémszoftver

Ezek a szoftverek arra specializálódtak, hogy információt gyűjtsenek az áldozatról. Ezen információk lehetnek például a belépési jelszavak, banki adatok, az interneten folytatott tevékenységek vagy személyes információk. Napjainkban gyakran engedünk be szoftvereket mi a környezetünkbe és ők legálisan vagy fél legálisan gyűjtenek adatokat rólunk. Ez nem összekeverendő a kém szoftverekkel, mivel ilyenkor engedélyt adunk erre, ellentétben a kém szoftvereknél, ahol ez jogtalanul történik.

Más esetekben ezek a szoftvereket katonai céllal fejlesztik akár ipari kémkedés vagy teljes államok-, kormányzatokról való információ szerzés céljából.

2.1.8. Fájl nélküli kártékony szoftver

Ezen típusú kártékony kód az arra épül, hogy a rendszer legitim eszközeit kihasználva érje el destruktív céljait bármilyen fájl telepítése nélkül. Egyik gyakori technikája a Living of the Land (LotL). Az ehhez gyakran használt szoftver például a PowerShell. [17]

2.1.9. Bányászszoftver

Napjainkban, mivel a kriptovaluták ára megnövekedett, ezért a kártékony kódot piacán megjelent egy fajta szoftver, amivel a támadónak lehetősége van arra, hogy jogtalanul bányásson az áldozata számítógépén. Ezzel károsíthatja a videokártyát és effektív bevételhez juthat. Ezen vírusok elterjedése a kriptovaluták népszerűségének növekedésével nőtt.

2.1.10. Hibrid kártékony szoftverek

Egy hibrid kártékony szoftver gyakorta a fentiekben említett alfajoknak az összekeveréséből áll. Például egy terjedő zsaroló szoftver terjedhet egy férgeként és feltelepíthet egy rootkit-et is, hogy megbizonyosodjon arról, hogy mindenképp a támadó tudja csak visszafordítani a vírus által okozott károkat vagy adatvesztés során egy teljes telepítés legyen szükséges a káros szoftvertől való megszabaduláshoz.

2.2. Kártékony kódok története

Ebben a részben a kártékony szoftverek történetét fogom röviden bemutatni. Említésre kerül nagy vonalakban a kezdetek és a legelső ilyen típusú kódok, valamint több az ezredforduló utáni káros szoftverek külön-külön alfejezetebe fogom tárgyalni. [18] [19]

2.2.1. Fogalom születése és fejlődés

Az 1940-es évek végén Neumann János a modern számítógépek architektúrájának megalkotója már felvetette a gondolatot, miszerint lehetséges olyan számítógépes programot készíteni, ami képes arra, hogy saját magát replikálni. Ezt a tanulmányát poszthumusz 1966-ban publikálták.

2.2.2. Az első kártékony kódok

1971-ben a Bob Thomas készítette el a Creeper férget. A féreg az ARPANET-en terjedt és fertőzte meg a hálózaton lévő számítógépeket. Ez a program egy kísérlet volt Neumann János elméletének bebizonyítására, így károkat nem okozott semmiben. Egyedüli funkciója, hogy a kimenetre kiírta: „Én vagyok a Creeper: Kapj el, ha tudsz”. 1972-ben Ray Tomlinson készítette el az ehhez tartozó Reaper névre keresztelt antivírus szoftvert, ami eltakarította a Creeper-t.

Az első káros féreg, amiről szeretnék beszélni a Morris. 1988-ban a Cornell Egyetem hallgatója Robert Tappan Morris Jr. fejlesztette. A szoftver a BSD Unix operációs rendszerben található sérülékenységeknek köszönhetően terjedt el. Mivel ez akkoriban már bűncselekménynek számított, ezért a bíróság három év felfüggesztett börtönbüntetést, 10 ezer dolláros pénzbírságot és közmunkát szabott ki rá. [20]

2.2.3. ILOVEYOU

2000 májusában bukkant fel az ILOVEYOU féreg, amit az akkor 24 éves Fülöp szigeteki Onel de Guzman fejlesztett ki. A szoftver Windows számítógépeket támadott és tökéletesen manipulálta a felhasználót a célja elérése érdekében. Egy email mellékletben érkezett egy „LOVE-LETTER-FOR-YOU.TXT.vbs” névvel ellátott fájl. A .vbs kiterjesztés a Visual Basic Script-et takarja. Ez egy általában automatizálásra használt értelmezett nyelv Microsoft Office termékekhez, de még a mai napig rengeteg kártékony kód készül így gyakorta Word fájlokhoz hozzacsatolva. A levél tartalma arra ösztönözte a felhasználót, hogy nyissa meg a mellékletet. Itt adathalász által használt technika az emberek társasági vágyára és a titokzatosság által nyújtott izgalomra játszik, hogy elérje célját. Ezt tetőzve azzal, hogy a fájl kiterjesztését elrejtette az operációs rendszer, ezért nem tudott gyanút fogni az áldozat. Ezek mellett a sikeres támadást még az is támogatta még 2000-es években az adathalászat nem volt annyira elterjedt és kifinomult, mint napjainkban.

A script lefuttatása után semmi látványos dolog nem történt, de a háttérben annál több. Az áldozat tudta nélkül tovább küldte a káros email-t az Outlook cím listákon szereplőknek, a féreg jelszavakat lopott és fájlokat törölt le a rendszerből. A szoftver úgy terjedt, mint a pandémia.



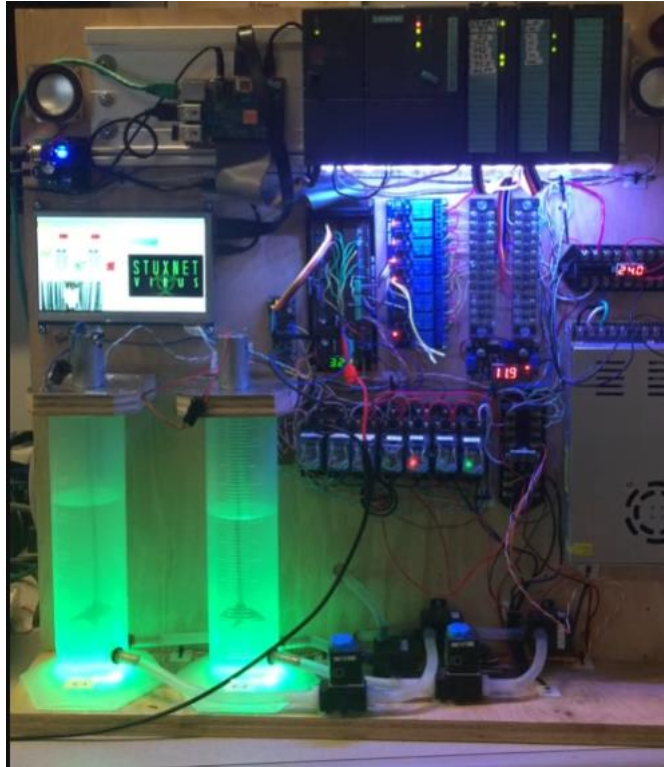
2. ábra ILOVEYOU email [21]

2.2.4. STUXNET

Ezt a kártékony szoftvert az egyik legszofisztikáltabb, amiről egyelőre tudunk és ezért a legtöbbet tanulmányozott is. Ezt a férget feltételezhetőleg az USA és Izrael közösen fejlesztette ki, Irán ellen. Az első variánsa 2009-ben kezdett el terjedni. Egy átlagos káros kóddal ellentétben itt nem a hétköznapi felhasználó volt a célpont, hanem a helyi nukleáris programhoz tartozó uránium dúsító üzem SCADA rendszere volt a célpont. Azon belül is az ipari PLC-k, amivel a folyamatot irányították. Ezen a Siemens által gyártott eszközöket és az ezekkel kommunikáló Windows gépeket kereste a féreg. Itt a két eszköz közötti kommunikációt szabotálta és hamisította. Vezérlési oldalról a Windows-os gépek nem mutattak semmit, de a PLC-k hibásan működtek.

A szoftvert úgy készítették, hogy az csak iráni dúsítógépeken fejtse ki káros hatását így, ha más olyan környezetbe került, ami nem volt a célja a kódnak akkor ott nem csinált semmit. Emellett a kód tartalmazott egy sérülékenységi csomagot, ami négy nullnapos sérülékenységből állt, ezzel biztosítva azt, hogy az eszköz bejárhat minden számítógépet a célpont megtalálása érdekében.

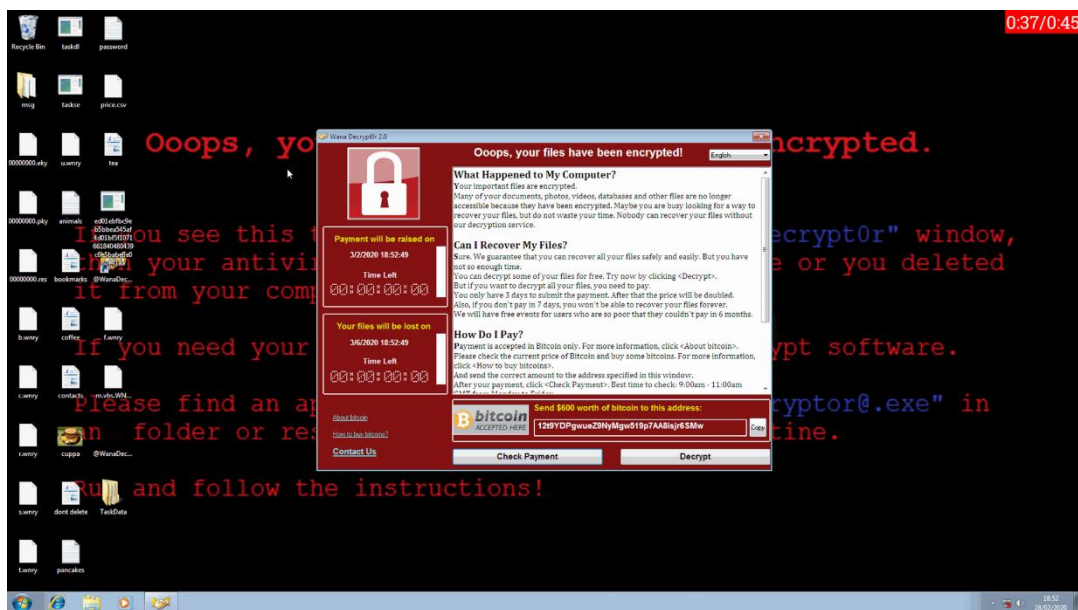
Ezt a fajta kártékony kódot „Stealthware”, avagy rejtőzködő szoftvernek szokták hívni. Ennek az elsődleges célja, hogy minél szofisztikáltabb és nehezen észrevehetőbb legyen. Erre az egyik módszer, hogy valamilyen módon megszerzett digitális aláírással biztosítja, hogy az antivírussal szemben. A következő ábrán egy demó rendszernek a felépítése látható. [22] [23]



3. ábra STUXNET demo [24]

2.2.5. WannaCry zsaroló szoftver

A WannaCry egy zsaroló szoftver, ami 2017 májusában szabadult el. A káros kód egy fűreg volt, ami a titkosított minden fájlt a rendszerünkön és mivel egy nulla napos sérülékenységet használt ki ezért képes volt a hálózaton található összes Windows operációs rendszerrel rendelkező gépet megfertőzni. A nulla napos sérülékenységet az NSA fejlesztette ki és EternalBlue névre hallgatott. Ezt egy The Shadow Brokers nevű csapat lopta el az Amerikai Nemzet Biztonsági Ügynökségtől. Ez a kritikus sérülékenység a Microsoft által az MS17-010 kódnév alatt található meg. A hiba a Microsoft Server Message Block-ban volt megtalálható, ami távoli kód futtatásra adott lehetőséget. A következő képen látható, hogy hogyan nézett ki egy lezárolt számítógép asztala. [25]



4. ábra WannaCry fertőzött asztal [26]

2.2.6. NotPetya

2016 márciusában fedezték fel először a Petya nevű zsarolóvírust ez hasonlóan a WannaCry-hoz titkosítás után Bitcoin-t követelt az áldozattól. Ez a vírus a partíciós szektort fertőzte meg és a futása után újraindította a fertőzött eszközt. Egy nagy különbség volt a kettő között ez pedig a terjedésük módjával volt. A Petya social engineering-el terjedt, lényegében egy trójai vírus volt, ellentétben az utódjával. 2017-ben a WannaCry támadás után pár hónappal került elő a Notpetya. Neki a terjedési módja már sérülékenységen keresztül történt és ugyanúgy az EternalBlue-t használta ki.

A támadók Ukrajna függetlenségi napjának éjszakáján szabadították el a szoftvert, ezért sokan gondolják, hogy a támadás mögött az orosz kormány állhat. Habár ezt sosem vállalták magukra, a szoftver komplexitásából arra lehet következtetni, hogy valószínűtlen, hogy ezt valaki csak szórakozásból készítette volna. Az oroszok azzal védekeztek, hogy ők is érintettek voltak a támadásban, mint például az egyik olajcégük. [27] [28]

A dolgozat nem tárgyalja az elkövetkezendő évtizedet a terjedelmi határok miatt.

2.2.7. NSO Pegasus

Az NSO csoport egy izraeli kiber hírszerzési cég. Olyan szoftvereket fejlesztenek és adnak el, aminek segítségével képesek távolról megfigyelni felhasználókat. Ezeket az eszközöket csak kormányok tudták megvásárolni.

A Pegasus szoftver az egyik ilyen terméke a csoportnak. Ezzel képesek az Apple által fejlesztett IOS telefonos operációs rendszerben egy hibát kihasználni. Ez a sérülékenység szintén

egy nulla napos és a kihasználásához nincs szükség semmiféle felhasználói interakcióra, azaz zero click exploit, ahhoz, hogy a telefont kompromittálja.

Ez egy kémsoftver. Képes a felhasználóról mindent megmondani. Folyamatosan lekérdezhető a telefon helyzete, mindenféle kommunikáció, ami a felhasználásával történik üzenetek, hívások és ezek mellett minden letöltött applikációt is.

Magát a sérülékenységet a telefonra egy ismeretlen hívással vagy egy üzenettel lehet feltelepíteni. Ez nagyjából egy pár másodperc alatt történik. Rengeteg ország vásárolta meg és használta ezt a szoftvert, egyes helyeket gyakran nem jogtalanul. Itt fontos még megjegyezni, hogy az esetek túlnyomó többségében ezt az eszközt nemzetbiztonsági ügyek megoldása érdekében használták jogosan, hogy terrorista szervezetek, illegális narkotikumokat terjesztő csoportok kommunikációs hálózatát tudják kompromittálni. [29]

3. Elemzéshez használt eszközök bemutatása

A kártékony szoftvereket egyértelműen a saját, illetve a környezetünk érdekében szigorúan tilos a fő operációs rendszerünkön vizsgálni, mivel ezzel nem csak sajátmagunkat hanem a velünk együtt élő személyeket vagy akár egy teljes vállalatot is veszélybe sodorhatunk. Ezért érdemes egy erre felépített „digitális mikroszkópot” építenünk, amivel képesek vagyunk a szoftver mintának a viselkedését tanulmányozni és ellenanyagot készíteni. Ha erre nincs lehetőségünk, akkor léteznek cégek, akik ezt szolgáltatásként adják.

3.1. Virtuális gépek felépítése és konfigurációja

3.1.1. Virtualizációs szoftver

Az elemzéshez a következőkben mind a statikus és a dinamikus elemzéshez virtuális gépeket fogok használni. A szoftver a VMware Workstation Pro, mivel az Oracle által fejlesztett VirtualBox véleményem szerint nem annyira megbízható, gyors és rendszeresen karbantartott, igaz az általam használt szoftver licenz köteles, de sokkal stabilabb egyszerűbben használható és természetesen mindenre képes, amire a VirtualBox.

A VMware Inc. vállalatot 1998-ban a Kaliforniai Palo Alto-ban alapította Diane Greene és Mendel Rosenblum. Az évek során végül a Broadcom vállalathoz került, napjainkban felhő alapú megoldásokkal foglalkoznak. [30]

Amiért érdemes virtualizációt használni az pedig az úgy nevezett snapshot funkció. Ezzel képesek vagyunk egy mentést készíteni a virtuális gépünkről, ami kifejezetten hasznos olyan értelemben, hogy nem kell minden egyes kártékony kód futtatása után újból telepíteni a teljes rendszerünket. Egy gomb megnyomásával visszatudjuk állítani a detonáció előtti állapotunkat és érdemes gyakorta snapshot-okat készíteni.

3.1.2. FlareVM

Az analízishez választott virtuális környezetem a FlareVM. Ez egy olyan speciális virtuális gép, ami arra lett kifejlesztve, hogy kód visszafejtést és kártékony kód elemzést segítse elő Windows-os környezetben, főleg a PE futtatható fájlok elemzéséhez hasznos. Ahhoz, hogy használhassuk egy friss Windows 10-et kell feltelepíteni. A minimum feltételek között van még, hogy rendelkezni kell a PowerShell 5-ös verziójával, 2 gigabájt memóriával, valamint 60 gigabájt háttértárral. Mindezek mellett konfigurálni kell magát az operációs rendszert is. Először is ki kell kapcsolni a frissítéseket majd az összes vírus elleni védelmet le kell tiltani, amit két helyen is meg kell tennünk, hogy maga a környezet ne állítsa meg a detonációt. [31]

Egy érdekes információ a FlareVM-mel kapcsolatban, hogy abban az időszakban, amikor telepítettem még nem volt ismert a CVE-2024-3094 sérülékenység, ami az XZ tömörítő

könyvtárban a liblzma-ban volt megtalálható. Ez eredetileg egy Linux sérülékenység volt, de ha valaki a cygwin szoftvert használja a számítógépén és ha az idei év tavaszának eleje és közepe között lett feltelepítve, akkor nagy valószínűséggel rendelkezik ezzel a hibával. Én is ide tartozok, ahogy az alábbi ábra mutatja, 5.6.x-es verziók mind sérülékenyek, de kihasználtság csak akkor történik, ha az használjuk az ssh-t, amit szerencsére se eddig, se ezután nem fogok, mivel ez nem egy olyan környezet.

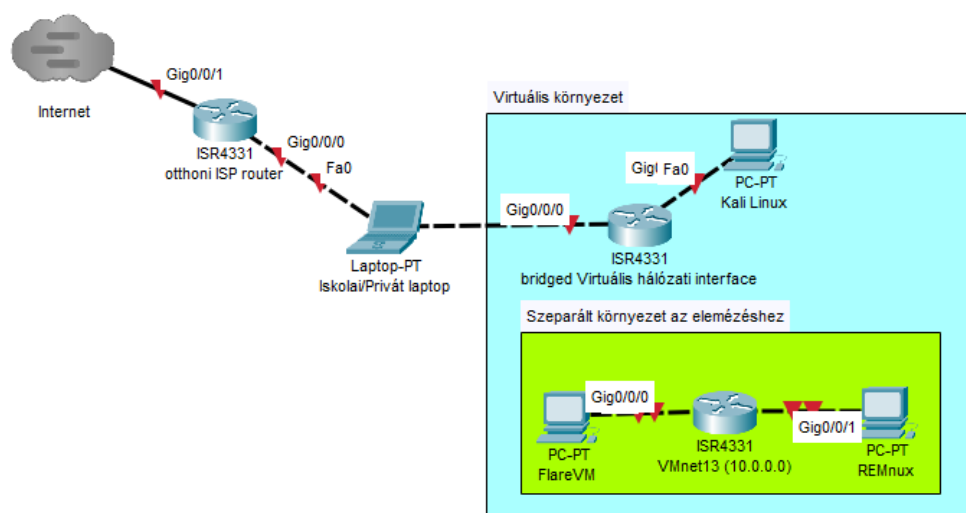
3.1.3. REmnux

A REmnux egy olyan Linux alapú reverse engineering-re, avagy szoftver visszafejtéshez használt operációs rendszer, amit az Ubuntu 20.04-hez készítettek. Telepíteni két módszerrel lehet. Az egyik az előbb említett rendszerre letölteni egy egyszerű wget paranccsal. A másik lehetőség, hogy ha virtuális gépet használunk az, hogy a REmnux virtuális háttértárat csatoljuk a telepítéshez. [32]

Jelen esetben a csomagok közötti egyik alkalmazással szeretném, majd szimulálni az internetet az előbb említett virtuális gépnek. Ez akkor fontos, hogyha egy kártékony szoftver több stádiumból áll. Ilyenkor képesek vagyunk megakadályozni azt, hogy a letöltött kód egyből kiértékelődjön mielőtt elemezni tudnánk.

3.1.4. Virtuális hálózat

Ezt a hálózatot a virtualizációs szoftverünk szolgáltatja. Itt képesek vagyunk arra, hogy az helyi hálózatunk egyik eszközén létrehozunk egy szegmenst, ahol két számítógép közötti kapcsolat kihasználásával tudjuk modellezni a támadó, illetve az áldozat környezetét. Ezen eszközöknek nincs semmilyen kommunikációs csatornája a külvilággal, avagy az internettel csak is egymáshoz van hozzáférésük. A következő ábrán ezt egy bemutató jelleggel lehet megtekinteni.



5. ábra A környezet felépítése

Ez a hálózat a környezetben a VMnet13-as alhálózat és mivel egy /24-es hálózatról van szó, ezért a maszk 255.255.255.0 és a DHCP az első címet 10.0.0.3-tól tudja kiosztani. A következő ábrákon lehet látni, hogy milyen címeket kaptak virtuálisgépek. [33]

```
C:\Users\flare
λ ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix  . : localdomain
    Link-local IPv6 Address . . . . . : fe80::9c40:861b:e748:ed19%7
    IPv4 Address. . . . . : 10.0.0.3
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :
```

6. ábra FlareVM ipconfig

```
remnux@remnux:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:0e:22:ac brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.4/24 brd 10.0.0.255 scope global dynamic ens33
        valid_lft 1522sec preferred_lft 1522sec
    inet6 fe80::20c:29ff:fe0e:22ac/64 scope link
        valid_lft forever preferred_lft forever
```

7. ábra remnux ip a

3.2. Elemzéshez használt szoftverek

Az elemzéshez használt szoftverek bemutatása a statikus elemzésre alkalmas eszközök bemutatásával fog kezdődni és a dinamikus-al fog folytatódni.

3.2.1. Detect It Easy

Detect It Easy egy Windows, macOS és Linux platformról is elérhető ingyenes nyíltforráskódú bináris fájlanalízishez használt eszköz, ami rendelkezik hexadecimális szerkesztővel is. Ez a szoftver egy fájl hexadecimális értékeit mutatja meg és ezen keresztül lehet elemezni például egy PE fájl karakterláncait. Ezek mellett képesek vagyunk a szoftverrel a fájl egyes szekciójának entrópiáját is megvizsgálni.

3.2.2. Ghidra

A Ghidra az NSA által Java-ban és C++-ban fejlesztett visszafejtéshez használt ingyenes és nyílt forráskódú szoftver. Ez lehetőséget kínál nekünk C- vagy C++ -ban fejlesztett szoftvereket elemzéséhez. Az eszköztárában megtalálható több más dolog mellett egy disassembler, amivel a fájl processzor műveleteit tudjuk megvizsgálni, valamint megtalálható egy decompiler, amivel meg az assembly kimenetéből készít a program egy pszeudó kódot. [34]

3.2.3. x64dbg/x32dbg

Az x64dbg és az x32dbg egy olyan nyílt forráskódú debugger, amivel Windows-os futtatható fájlokat tudunk elemezni futásközben. A szoftvert kártékony szoftverek elemzői használják .exe és .dll fájlok elemzéséhez.

3.2.4. Wireshark

A Wireshark egy Gerald Combs által készített szoftver, amit C/C++-ban fejlesztettek és először 1998-ban adtak ki. Egy ingyenes, nyíltforráskóddal rendelkező eszköz, amivel a hálózati csomagokat tudjuk elemezni. Szinte minden platformon elérhető de jelen esetben a REMnux-os virtuális gépről fogjuk elemezni az érkező hálózati forgalmat. [35]

3.2.5. INetSim

Az INetSim nevű parancssori alkalmazás egy beépített része a REMnux operációs rendszernek. Ennek a szoftvernek az a célja a nevéből adódóan, hogy az internetet szimuláljon egy virtuális hálózaton egy dinamikus elemzéshez használt környezetnek. Ez hasznos lehet olyan esetekben az elemzett kártékony kód több stádiumból épül fel és az internet felé nyúl ki a következő kódrészletért. Egy ilyen esetben a nem csak választ ad neki az INetSim, hanem ha egy fájlt kér akkor generál egy megegyező nevű, de hatástalanított állományt. Mivel nem egy éles állomány lesz kiértékelve, két fontos dolgot is le tudunk szűrni ennek a programnak a segítségével az egyik egy IOC a másik pedig egy következő stádium, amit újból elemezhetünk.

3.2.6. AnyRun

Az AnyRun egy internetes szolgáltatás, ahol egy sandbox környezetben tudunk tesztelni fájlok viselkedését. Ez hasznos, hogy kártékony kódokat nem szeretnénk egy saját rendszerben dinamikusan elemezni. Egy nagyon egyszerű és jól értelmezhető felülettel rendelkezik, ahol egy Windows 7-es operációs rendszerben lehet lefuttatni az elemezni kívánt szoftver mintát. A feltöltött fájl lefuttatásakor az AnyRun figyeli a fájl által létrehozott folyamatokat és megmutatja a kifelé küldött kapcsolati kísérleteket. A szolgáltatást korlátozottan lehet ingyenesen is használni, de ehhez mindenképpen szükségünk van egy vállalkozói e-mail címre, hogy hozzáférésünk lehessen az eszközhöz.

4. Statikus és dinamikus elemzés

A kártékony kódok elemzése azért fontos, mert a mind statikus, illetve dinamikus módszerrel kulcsfontosságú információkat tudunk szerezni azzal kapcsolatban, hogy hogyan és mi ellen kell védekeznünk egy esetleges támadási kampány időszaka alatt.

Ebben a fejezetben egy kártékony kód mintát fogok elemezni mindkét módszerrel és az elemzés eredményeit fogom dokumentálni. Prynt Stealer káros szoftvert fogok elemezni. Ezt a kód mintát Malware bazaar weboldáról szereztembe mivel nem szerettem volna egy olyan szoftvert elemezni, amit már sokan elemeztek és készítettek róla tartalmat. Ezt a szoftvert C/C++-ban fejlesztették.

4.1.1. Statikus elemzés bemutatása

Statikus elemzésnek nevezzük azt amikor a szoftverünket nem futtatjuk, hanem kódjának valamilyen formáját elemezzük. Ezt végezhetjük egy disassembler-el, ami a PE futtatható fájl formátumból készít assembly kódot, decompiler-el ez gyakorta az előző eszköz kimenetéből generál egy pszeudó kódot vagy egy hexadecimális szerkesztővel, ahol a fájlban a hexadecimális értékeit tudjuk megnézni és megváltoztatni. Más esetekben elég, hogyha egy szövegszerkesztővel nyitjuk meg ilyenek a script alapú rosszindulatú alkalmazások, mivel ezeknél a nyelveknél nem egy fordító, hanem egy értelmező végzi a munkát.

Ezen elemzésnél a használt szoftver nagyban függ attól, hogy a szoftver készítője milyen programozási nyelvet használt, mivel a, ha egy értelmezett nyelvnél vagy script nyelvnél a kód maga általában tiszta szöveggént olvasható, amíg más fordított nyelveknél a fájl egy fordított formátumból például PE-ből kell visszafejteni. Például egy C#-ban készített káros kódot nem tudunk Ghidra-val elemezni, hanem ilyenkor a dnSpy vagy a dotPeek szoftvert érdemes használni. [36]

4.1.2. Dinamikus elemzés bemutatása

Dinamikus elemzésnek azt nevezzük amikor egy kártékony szoftvert futtatunk és tanulmányozzuk annak működését, avagy a viselkedését és a hálózati kommunikációját elemezzük, mivel egyes kártékony kódok, amelyeknek a célja információ lopás vagy egy több stádiumból álló káros szoftver, rendelkezik egy mögötte neki parancsoló és őt irányító szerverrel, amit a szaknyelvben C2 (Command and Control) szervernek nevez. Ezen szerverek a feladata egy előző stádium kiszolgálása vagy lopott információ továbbítása a támadónak és ezzel a kiletét fedni. Ehhez a fajta elemzéshez érdemes használni 3.fejezetben említett eszközöket. [36]

4.2. PE fájl alapok az elemzéshez

Ebben az alfejezetben a PE fájl struktúrájának két részét szeretném kiemelni. Az első részben a DOS fejléccet a másodikban, pedig az „imports” szekciót. Ezen fájlformátum ismerete elengedhetetlen a kártékony szoftver elemzés világában, ugyan úgy, mint a C/C++ ismerete hiszen ezen a nyelven készítenek a nagyon sok ilyen kódot, mivel a Win32 API segítségével, jól manipulálható a Windows operációs rendszer.

A PE azaz portable executable a nevéből adódóan egy hordozható és futtatható fájlformátum, amit más Windows operációs rendszerrel rendelkező számítógépeken is futtathatunk probléma mentesen. Ezen struktúrát használják például a .exe és .dll fájl kiterjesztéssel rendelkező állományok mind 32 és 64 biten is. A következő ábrán lehet látni, hogy milyen felépítése van ezeknek a fájloknak.

PE¹⁰¹ a windows executable walkthrough
Ange Albertini
corkami.com

Dissected PE

simple.exe

header
technical details about the executable

sections
contents of the executable

DOS header
simple

PE header
simple

optional header
simple

data directories
simple

sections table
simple

code
simple

imports
simple

data
simple

Sections table

x86 assembly

Equivalent C code

Imports structures

Consequences

Loading process

- 1 Headers**
The DOS header is parsed.
The PE header is parsed.
The optional header is parsed.
The data directories are parsed.
The sections are parsed.
- 2 Sections table**
Sections table is parsed.
The sections are mapped to memory.
The sections are loaded into memory.
- 3 Mapping**
The file is mapped in memory according to the ImageBase.
The sections are mapped to memory.
- 4 Imports**
Data directories are parsed.
The ImportTable is parsed.
The ImportTable is mapped to memory.
The ImportTable is loaded into memory.
- 5 Execution**
Code is called at the EntryPoint.
The code is executed.

Notes

- PE IMAGE** aka **DOS_HEADER**
Starts with 'MZ' (Mark Zbikowski MS-DOS developer)
- PE IMAGE** aka **IMAGE_FILE_HEADER** / **COFF** file header
Starts with 'PE' (Portable Executable)
- OPTIONAL_HEADER** aka **IMAGE_OPTIONAL_HEADER**
Optional only for non-standard PE but required for executables
- RVA** Relative Virtual Address
Address relative to ImageBase (at ImageBase, RVA = 0)
Almost all addresses of the headers are RVAs
In code, addresses are not relative
- DATA** Import Name Table
Null-terminated list of pointers to Hint, Name structures
- DATA** Import Address Table
Null-terminated list of pointers
On file it is a copy of the DATA
After loading it points to the imported APIs
- HINT**
Index in the exports table of a DLL to be imported
Not required but provides a speed-up by reducing look-up

8. ábra PE fájl felépítése [37]

4.2.1. DOS fejléc

A PE fájlok DOS fejléccel kezdődnek, ami a régi DOS alapú operációs rendszerekhez a kompatibilitást támogatja. Ezen rész hossza 64 bájt és az első két bájt mindig 4D 5A. Ennek a háttérét az adja, hogy a két bájt ASCII karakterekként MZ, ami Mark Zbikowski az MS-DOS egyik vezető fejlesztőjének a két kezdő betűje. Ezt a kód struktúrájában *e_magic* avagy a mágikus szám mezője, ezáltal tudja az operációs rendszer, hogy PE fájlal kell dolgoznia. Ez után

következik a DOS Stub. Ez a rész tartalmazza a „This program cannot be run in DOS mode.” karakterláncot, ami olyan esetben jelenik meg, ha a fájl nem megfelelő kiterjesztésben van jelen.

Emellett tartalmazza még az *e_lfanew* mezőt is, ami pedig megmutatja, hogy hol található a következő fejléc. [38]

4.2.2. Import szekció:

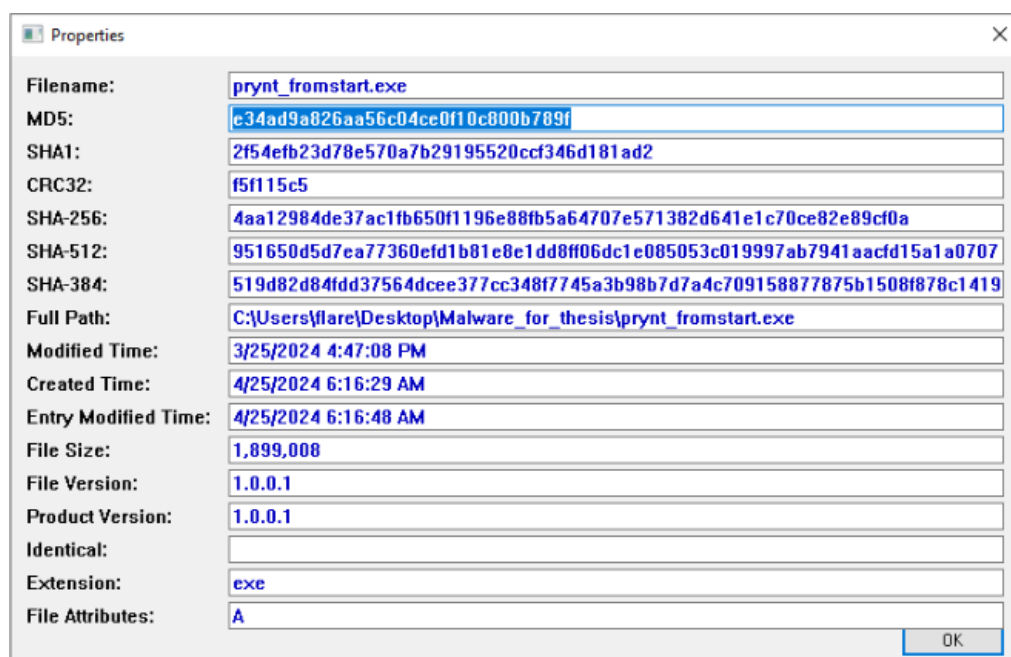
Az import szekció azért fontos számunkra, mivel a futtatható fájl ezen területén vannak feljegyezve azok függvények a dinamikus könyvtárak szerint csoportosítva, amelyeket a szoftver használ. Ezen függvények mellett megtalálható a hozzátartozó hexadecimális cím, amin keresztül a program referál rá. [39]

4.3. Statikus elemzés

Statikus elemzésünket azzal kezdjük, hogy információt gyűjtünk az adott fájlról az internet segítségével, ezzel a kódot jól azonosítani tudjuk. Ezenkívül egy saját Python script-el is fogunk információt gyűjteni a szoftverről.

4.3.1. Azonosítás

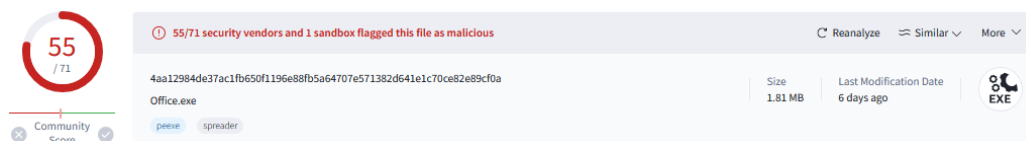
Az azonosítást először is azzal kezdjük, hogy megnézzük a fájl mind az MD5 és az SHA1-es hashét és ledokumentáljuk ezeket. Ezt a HashMyFiles programjával teszem, ami egy beépített alkalmazása a FlareVM-nek. Az eredményeket a következő ábrán lehet látni.



9. ábra Prynt hashek

4.3.2. Virus Total

Ezek után a Virus Total weboldalnak fogom megnézni a fájl értékelését. Itt van lehetőség arra, hogy a hash alapján tudjunk lekérdezni az adatbázisból.



10. ábra Virus Total értékelés

Ahogy a fenti ábrán is jól látható 73 security vendoból 55 kártékonynak jelölte meg ezt a szoftvert, ezáltal biztosan állíthatjuk, hogy olyan műveleteket, hajt végre, amelyek a felhasználó szempontjából megsértik a biztonsági háromszöget. Mivel a viselkedési részét én szeretném végezni, ezért a „Behavior” részét a Virus Totalnak kihagyom. [40]

4.3.3. Saját elemző script:

A statikus analízishez készítettem egy rövid python scriptet, hogy ezzel ellenőrizni lehessen, hogy milyen Win32 API függvényeket használ a kártékony kód. A scriptben a található egy lista, ahol ezek a gyakran használt függvénynevek összevannak gyűjtve és ezeket ellenőrzi le az PE futtatható fájlban a pefile könyvtár segítségével. Ezeket a függvény neveket a következő forrásban találtam. [41]

```
import pefile
import sys

known_function_names = [ #ws2_32.dll
    "socket",
    "WSAStartup",
    "bind",
    "listen",
    "accept",
    "connect",
    "read",
    "recv",
    "write",
    "send",
    "shutdown",
    "WSACleanup",
    #Persistence
    "RegCreateKeyEx",
    "GetTempPath",
    "OpenSCManager",
    "RegOpenKeyEx",
    "CopyFile",
    "CreateService",
    "RegSetValueEx",
```

```

"CreateFile",
"StartServiceCtrlDispatcher",
"RegDeleteKeyEx",
"WriteFile",
"RegGetValue",
"ReadFile",
#Encryption
"WinCrypt",
"CryptAcquireContext",
"CryptGenKey",
"CryptDeriveKey",
"CryptDecrypt",
"CryptReleaseContext",
#Anti-Analysis/VM
"IsDebuggerPresent",
"GetSystemInfo",
"GlobalMemoryStatusEx",
"GetVersion",
"CreateToolhelp32Snapshot",
"CreateFileW",
"CreateFileA",
#Stealth
"VirtualAlloc",
"VirtualProtect",
"ReadProcessMemory",
"WriteProcessMemoryA",
"WriteProcessMemoryW",
"NtWriteVirtualMemory",
"CreateRemoteThread",
"NtUnmapViewOfSection",
"QueueUserAPC",
"CreateProcessInternalW",
"CreateProcessInternalA",
#Execution
"CreateProcessA",
"CreateProcessW",
"ShellExecute",
"ShellExecuteA",
"ShellExecuteW",
"ShellExecuteExW",
"ShellExecuteExA",
"WinExec",
"ResumeThread",
"NtResumeThread",
]

def list_compare(known_function_names_list, input_function_names_list):
    list_size = len(known_function_names_list)
    found_funtions_number = 0
    for input_function in input_function_names_list:
        for known_function in known_function_names_list:

```

```

        if input_function == known_function:
            found_funtions_number += 1
            print(input_function)
        print("\nChecked functions number: " + str(list_size) + "\nFound functions number: " + str(found_funtions_number))

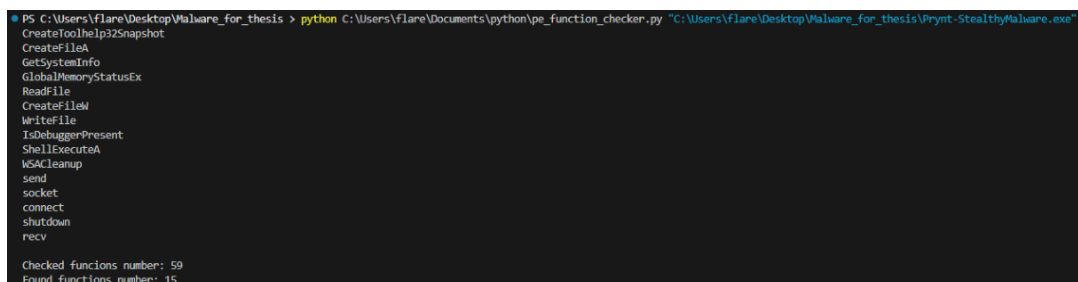
def main():
    fileName = sys.argv[1]
    isMalware = pefile.PE(fileName)

    isMalware_s_win32_api_functions = []
    for item in isMalware.DIRECTORY_ENTRY_IMPORT:
        for import_fn in item.imports:
            isMalware_s_win32_api_functions.append(import_fn.name.decode("utf-8"))
    list_compare(known_function_names, isMalware_s_win32_api_functions)

if __name__ == "__main__":
    main()

```

A következőben ezzel fogom analízálni a kártékony kódot és a következő ábrán lesz látható a Prynt – Malware-en lefuttatott végeredmény. A scriptről annyi fontos információt el kell még mondani, hogy a nem tökéletes, mivel minden kártékony kód más és más ezért nem lehet rá támaszkodni száz százalékban, viszont kaphatunk egy átfogó képet arról, hogy a szoftverben melyek azok import függvények, amelyekből arra lehet következtetni, hogy milyen káros tevékenységet folytat ez a kifejezett szoftver minta. A script olyan esetekben nem működik, ha a fájl valamilyen módszerrel bevan csomagolva például a UPX-szel.



```

PS C:\Users\Flane\Desktop\Malware_for_thesis > python C:\Users\Flane\Documents\python\pe_function_checker.py "C:\Users\Flane\Desktop\Malware_for_thesis\Prynt-Stealthy\Malware.exe"
CreateToolhelp32Snapshot
CreateFileA
GetSystemInfo
GlobalMemoryStatusEx
ReadFile
CreateFileW
WriteFile
IsDebuggerPresent
ShellExecuteA
WSACleanup
send
socket
connect
shutdown
recv
Checked functions number: 59
Found functions number: 15

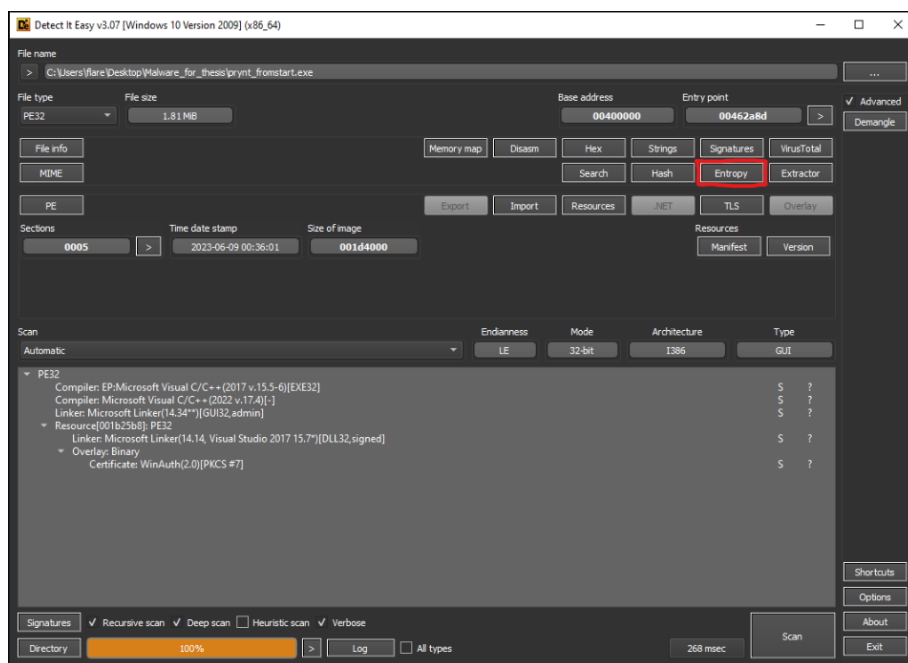
```

11. ábra python script eredménye

A fenti eredményből már jól látható, hogy a szoftver rendelkezik anti-debugging funkcionalitással erre az „IsDebuggerPresent” függvényből tudunk következtetni. Azt is tudjuk, hogy a ws2_32.dll-ből használja a felsorolt függvényeket, ezért bebizonyosodik az is számunkra, hogy valamilyen C2 szerverrel kommunikál a szoftver feltételezhetőleg információ lopás céljával.

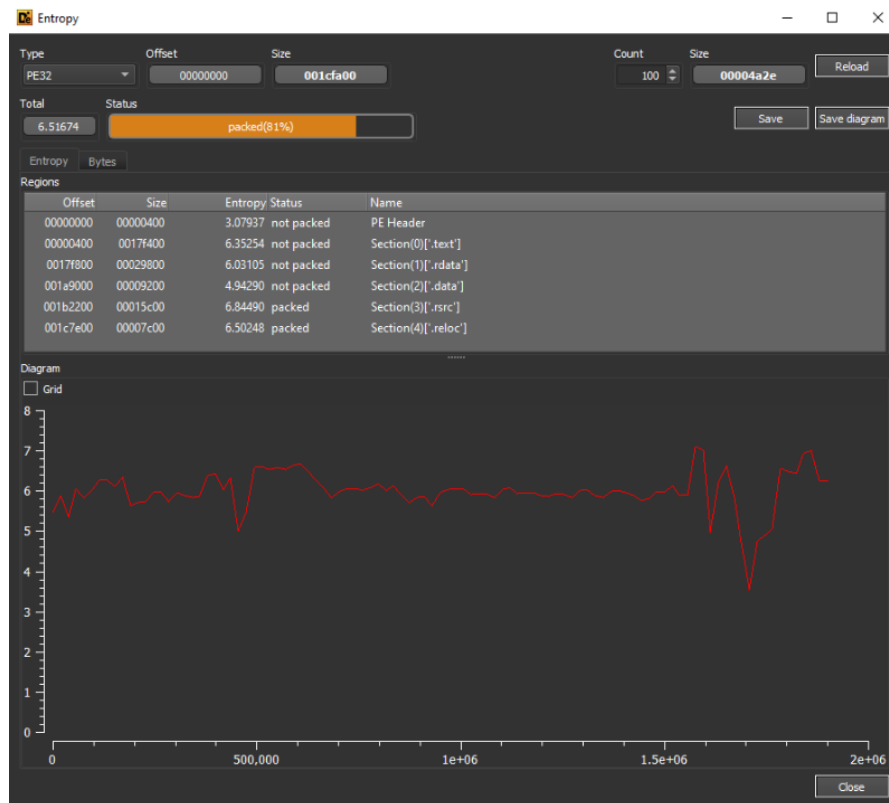
4.3.4. Detect It Easy elemzés

Következőben Detect It Easy szoftverrel fogjuk a fájlunk entrópiáját vizsgálni. Ez egy egyszerű feladat először is jobb kattintás a fájlra majd a lenyíló menüből ki kell választani a „detect it easy(DIE)” menüpontot. A következő ábrán lehet látni, hogy hogyan néz ki a grafikus felülete a szoftvernek és pirossal bejelölve a funkció, amit használni fogunk.



12. ábra detect it easy felülete

A következő ábra a DIE entrópia vizsgálatának az eredményét fogja megmutatni, amiben azt lehet majd látni, hogy mind a .reloc és a .rsrc szekció be van csomagolva.



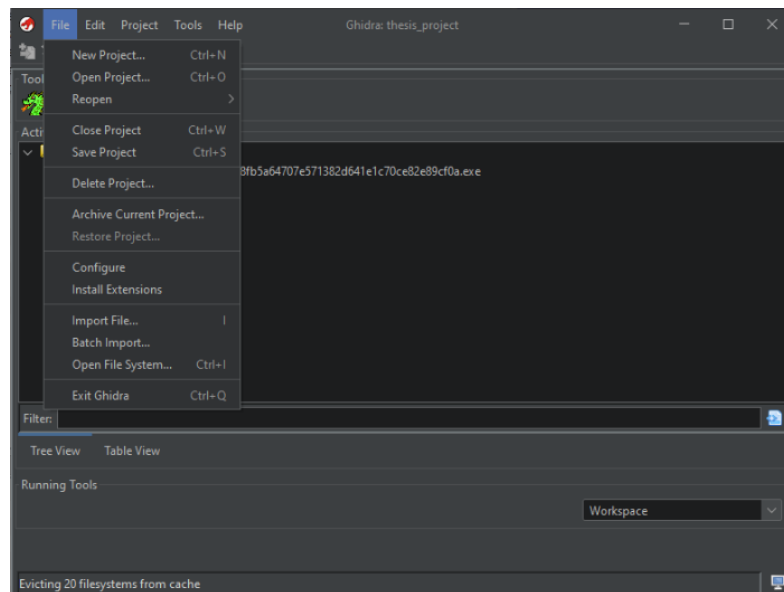
13. ábra Entrópia vizsgálat eredménye

Sajnos nem találtam semmilyen forrást sehol, aminek a segítségével ezeket kitudtam volna nyitni és elemezni a tartalmát.

4.3.5. Elemzés és visszafejtés Ghidra-val

Az elemzés előtt fontos még egy dolgot megjegyezni, ami pedig, hogy ha a fájl egésze be van csomagolva, akkor a nem leszünk képesek visszafejteni. Ilyenkor először is ki kell csomagolni, hogy utána folytatni tudjuk az elemzést, viszont jelen esetben ez nem releváns, mivel csak két szekciónál fedeztünk fel magas entrópiát.

Első dolgunk miután megnyitottuk a Ghidra-t, hogy csináljunk egy projektet, hogy legyen hova betöltenünk a fájlt, amit majd elemezni szeretnénk. A következő ábrán lehet látni, hogy vagy az „I” gomb megnyomással vagy a „File>Import File” útvonalon tudjuk hozzáadni az elemezni kívánt fájlunkat.



14. ábra Ghidra főképernyő

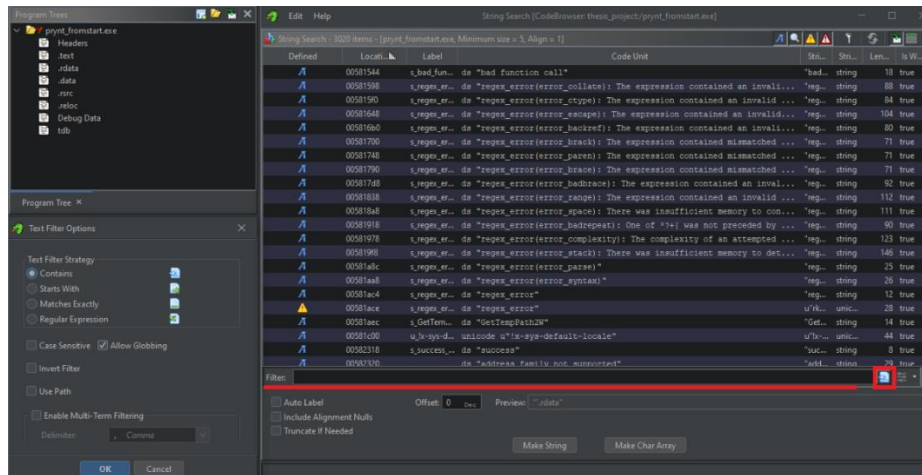
Ha először töltjük be a fájlt, akkor a Ghidra megkérdezi, hogy szeretnénk-e azt analizálni, amit mi természetesen fogunk. Jelen esetben az összes nem pirossal jelzett pontot kipipálom a, hogy az alábbi ábra mutatja, hogy minden információt leellenőrizzen a program, majd rákattintok az „Analyze” gombra és megvárom, hogy befejezze a folyamatot.



15. ábra Ghidra automatikus elemzési lista

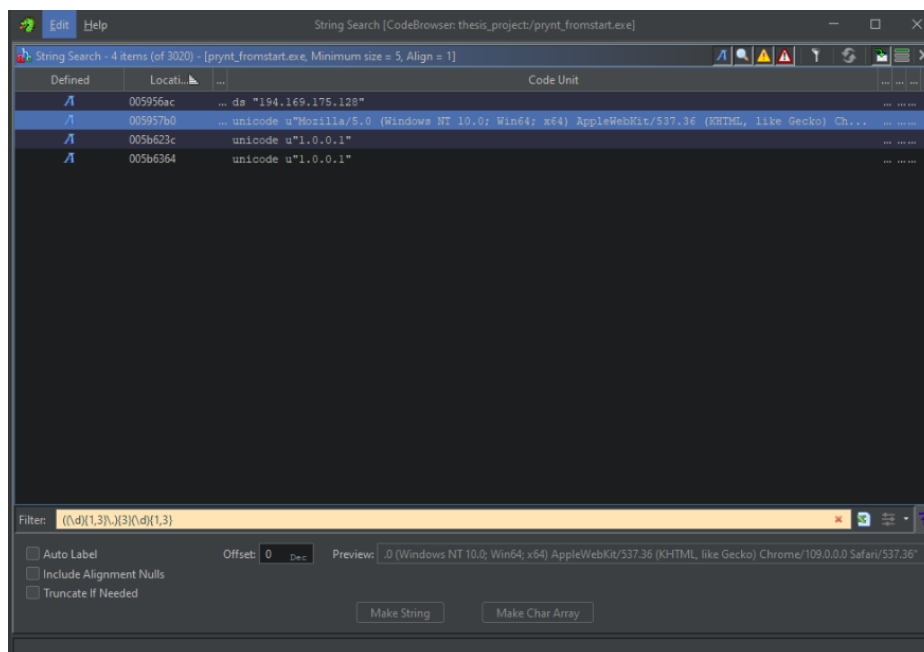
Az analízis után az első fontos lépés, hogy megkeressük a Ghidra alapján felismert karakterláncokat ezáltal könnyebben átláthatjuk, hogy milyen értékeket vesznek fel változók és azt is, hogy pontosan milyen win32api függvényeket használ a szoftver. Ezt úgy tudjuk megtenni, hogy a balfelső sarokban található „Search>For strings...” opciót kiválasztjuk és megadjuk a karakterláncunk minimum hosszát. Miután megnyílt az ablak a következő ábrán látható pirossal

vonallal jelölt szövegdobozból tudjuk megadni a szűrőt szűrni és a négyzetel jelölt gombra való kattintással be tudjuk állítani, hogy milyen módon szeretnénk szűrni.



16. ábra Ghidra karakterláncok szűrése

Itt a baloldali „Text Filter Options” ablakban az alapértelmezett „Contains”-t átállítom „Regular Expression”-re és az „OK” gombra kattintok, hogy könnyebben tudjak keresni minták alapján. Mivel olyan jelzőket keresünk, amivel egyes támadókat megszeretnénk akadályozni így érdemes IP címekre keresni. Ehhez a nem teljesen megfelelő, de működő reguláris kifejezés, amit használni fogok az a következő: `((\d){1,3}\.){3}(\d){1,3}`. Ezzel az a probléma, hogy olyan címeket is felismerünk, ami nem érvényes címek esetleg verziószámok, de már leszűkíti a táblában fellelhető rekordok számát. A következő ábrán látható ennek az eredménye.



17. ábra Karakterlánc szűrési eredmény

004629fc	56	PUSH	ESI	
004629fd	50	PUSH	EAX	
004629fe	57	PUSH	EDI	
004629ff	68 00 00	PUSH	IMAGE_DOS_HEADER_00400000	
	40 00			
00462a04	e8 d7 c7	CALL	FUN_0049f1e0	undefin
	03 00			
00462a09	8b f0	MOV	ESI,EAX	
00462a0b	e8 05 06	CALL	FUN_00463015	uint FU
	00 00			
00462a10	84 c0	TEST	AL,AL	
00462a12	74 6a	JZ	LAB_00462a7e	
00462a14	84 db	TEST	BL,BL	
00462a16	75 05	JNZ	LAB_00462a1d	
00462a18	e8 bb 7c	CALL	__cexit	void __
	02 00			

20. ábra Ghidra WinMain

4.4. Dinamikus elemzés

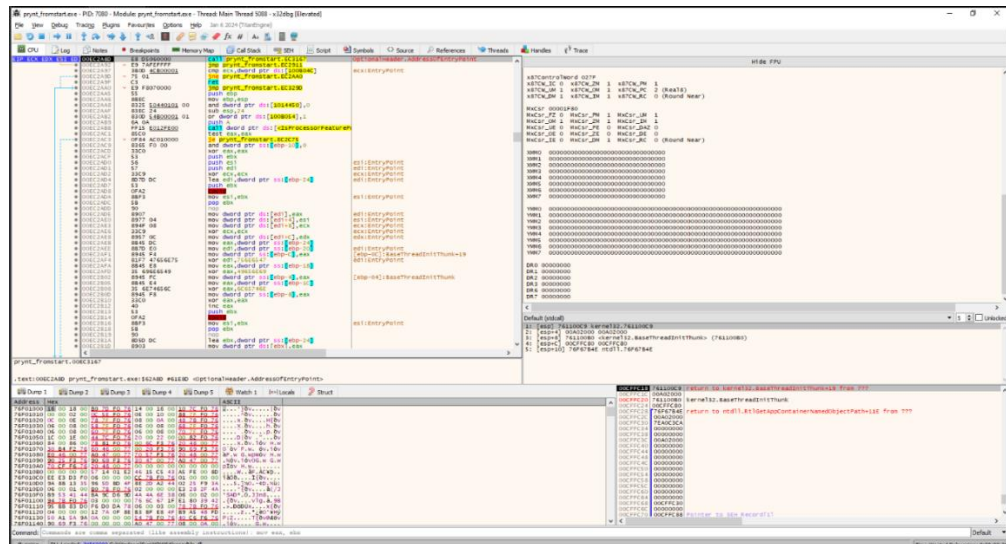
A dinamikus analízist (sandboxing) először az AnyRun platformon fogjuk elkezdni, hogy ne a saját rendszerünkön futtassuk először a kártékony kód mintánkat. Majd az x32dbg szoftverrel próbáluk meg elemezni.

4.4.1. AnyRun

Miután lefuttattuk a weboldalon a kártékony kódunkat, a hozzá tartozó konfigurációt is letudjuk tölteni. Itt könnyen lehet keresni C2 szervereket vagy más hálózati kommunikációra utaló nyomokat. A futtatás után a szoftver két konfigurációt talált. Ezeket a konfigurációs fájlokat az 1. és 2. számú melléklet tartalmazza. A listából jól látható, hogy mindenféle belépési információkat gyűjtenek java részt kriptovaluta tárcákhoz, bankkártya adatokat és más szolgáltatások fiókjának felhasználó nevét és jelszavát.

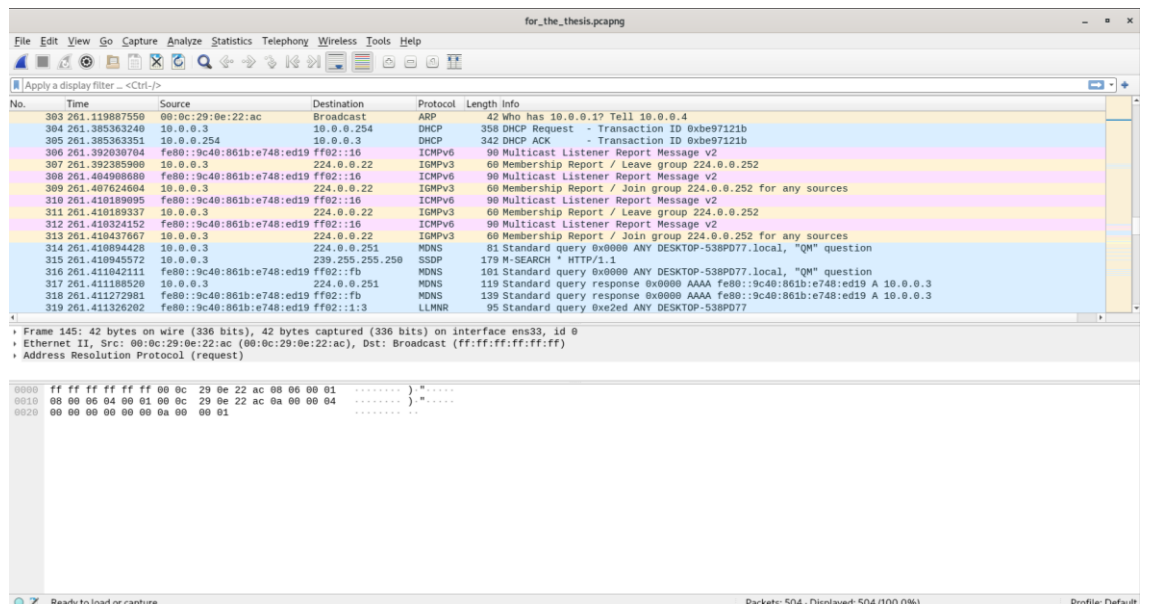
4.4.2. x32dbg és Wireshark

A következő részben szeretném futtatni a szoftvert a saját virtuális környezetemben és megfigyelni, hogy milyen hálózati kommunikációk fedezhetők fel a szoftver futásakor. Az alábbi ábrán látható a főképernyője a fájlnk betöltése után a debugger-nek.



21. ábra x32dbg grafikus felülete

A futtatással az a probléma, hogy pár lépés után kilép a program mivel figyeli, hogy van-e hozzákötve egy ilyen szoftver. Ezért mivel nincsen egy patchünk, amivel a szoftver ellenőrzéseit meghamisíthatjuk, debugger nélkül fogom lefuttatni. A következő lépésben a REMnux virtuális gépen elindítom az inetsim alkalmazást majd a Wireshark megnyitása után rácsatlakozok az ens33-as hálózati interfészre. Ezekután a FlareVM-ből elindítom a kártékony kódunkat. A következő ábrán láthatjuk, hogy milyen hálózati kommunikációk történtek a szoftver futtatása után.



22. ábra Wireshark hálózati kommunikáció

Ahhoz, hogy magabiztosan állíthassuk, hogy ezen kommunikációk a kártékony szoftverünk által történtek, viszont ami miatt feltételezhetjük, hogy a 224.0.0.252 az AnyRun-ban

jól látható, hogy ezt a hálózati kommunikációt az svchost.exe folyamat indította, amelyet támadók gyakran kihasználnak, hogy magasabb jogosultsági szintet szerezzenek.

A dinamikus elemzést a fenti eredményekkel szeretném lezárni, mivel az eddigi analízis során kiderült, hogy egy komplex kártékony kód és nem szeretném se a saját se a családtagjaim adatait kockáztatni, mivel a jelenlegi információink szerint nem tudjuk biztosan megmondani, hogy a szoftver nem próbálkozik azzal, hogy a virtuális környezetből és megpróbálja a host operációs rendszert megtámadni.

5. Kódolás és titkosítások elemzése

Ebben a fejezetben a kriptográfiával fogok foglalkozni és azon belül is a titkosítás, a kódolás, a hashelés és az obfuszkáció fogalmát bemutatni és ezáltal rávilágítani a különbségekre, mivel mindegyik fontos eleme a kártékony kódok elemzésében.

A titkosításnál bemutatom az ókori Cézár titkosítást és a modern RSA titkosítást, a kódolás részben említés lesz a base64 kódolásról, valamint az obfuszkációnál a fájlformátum hamisításról és a zaj generálást fogom bemutatni.

5.1. Titkosítás

Titkosításnak nevezzük azt a folyamatot, ahol egy üzenetet oly módon módosítunk, hogy csak az a személy olvashassa el, aki rendelkezik a megfelelő feloldókulccsal. Kétféle titkosítás létezik a szimmetrikus és az aszimmetrikus. A kettő között a különbség, hogy míg a szimmetrikusnál az adat titkosítása és annak feloldása ugyan azzal a kulccsal történik az aszimmetrikusnál ez két különböző kulccsal történik a publikus és a privát kulccsal történik. [43]

5.1.1. Szimmetrikus titkosítás

Szimmetrikus titkosításnak nevezzük az olyan titkosításokat, ahol csak egy kulcsot használunk mind a nyílt szöveg titkosításához mind a titkosított szöveg visszaalakításához.

Ilyen titkosításnak számít például a Cézár titkosítás. Itt a titkosítás arra alapszik, hogy a szöveget a latin ábécé szerint eltoljuk egy megadott számmal, ami a kulcsunk. Ez egy eltolással algoritmussal titkosítja az üzenetet. Tegyük fel, hogy a nyílt szövegünk, amit titkosítani szeretnénk a következő: „Szeretem a levest”. Miután megvan a titkosítani kívánt szövegünk ki kell választani a kulcsot, ami jelen esetben legyen a három, hogy Cézár hármastitkosítást csináljunk. Mivel két irányba tudjuk eltolni, ezért a két megoldásunk lesz, az egyik „Vchuhwhp d ohyhvw” a másik pedig „Pwbobqbj x ibsbpq”.

Ennek a titkosításnak több hátránya is van. Az egyik, hogy az úgy nevezett gyakoriság vizsgálattal könnyen megtalálható a kulcs. A példa jól szemlélteti ezt azzal, hogy mivel tudjuk, hogy magyar a titkosított üzenet és tudjuk, hogy a leggyakrabban használt betű az „e”, ezért kitudjuk következtetni, hogy az első megoldásban a „h” a másodikban „b” a legtöbbet használt betű, így feltételezhetjük, hogy ez az „e”.

A másik, hogy bármilyen ábécét használunk nagyon könnyen brute force-olható, tehát találgatással könnyen meg lehet találni a megoldást, mivel nincs sok változatosság a titkosításban. [44]

5.1.2. Aszimmetrikus titkosítás

Az aszimmetrikus titkosítás egy olyan rendszer, ahol két különböző kulcsot használunk ahhoz, a titkosítás folyamatához és annak feloldásához.

Ilyen a jelenleg a legjobban működő titkosítási algoritmust 1976-ban Ron Rivest, Adi Shamir és Len Adleman fejlesztette ki. A három MIT professzor vezetéknévének a kezdőbetűje lett a titkosítás neve, RSA. Ez a titkosítás kulcs párokat használ, egy publikus, illetve egy privát kulcsokat használ. Ezek természetesen egy szoros kapcsolatban vannak egymással. A titkosításnak a folyamata a következő. Vesszünk egy n számot, ami p és q prímszámoknak a szorzata majd a titkosítani kívánt üzenetet(m), amit, ha szöveg számokká alakítunk és ezt a publikus kulccsal lezárjuk ezzel megkapjuk a titkosított üzenetünket(c):

$$m^e \bmod n \equiv c \quad \text{Publikus kulcs} = e, n$$

Ha a küldés megtörtént a következő algoritmussal tudjuk megfejteni a titkos üzenetet:

$$c^d \bmod n \equiv m \quad \text{Privát kulcs} = d$$

Ezek alapján csak az a személy tudja meg az üzenetünk tartalmát, aki rendelkezik a privát kulccsal. A hagyományos számítógép nagyon nehezen számít prímfaktorizációt, mert sok számot kell ellenőriznie, hogy el tudja-e vele osztani. Ezeket a problémákat rejtett részcsoport problémáknak nevezik. Az RSA titkosítás alapja ebben nyugszik hiszen a p és q szükséges ahhoz, hogy kiszámoljuk a privát kulcsot.

Ezen titkosítás feltöréséhez Peter Shor amerikai matematikus 1994-ben megalkotott egy olyan algoritmust, amellyel az RSA- titkosítást egy kvantumszámítógép segítségével gyorsan fel lehet törni. Shor algoritmusával képesek vagyunk egy számot prímtényezőire bontani a hagyományos számítógépeknél sokkal gyorsabban. Az Simon algoritmusban Shor kicserélte a Hadamard szűrőt egy kvantum Fourier transzformációra, így megmutatta, hogy a kvantumszámítógép négyzetesen gyorsabban tudja megoldani a prímfaktorizációs feladatokat. [45] [46]

5.1.3. Titkosítás összegzés

Az elmúlt évszázadokban a titkosítási algoritmusok nagyon megváltoztak. Eleinte magát a titkosítási algoritmusokat is titokban kellett tartani, hogy ne lehessen visszafejteni a titkot. Ez napjainkba már pont az ellenkezője, mivel az interneten használt titkosítások azon alapszanak, hogy akkor se lehessen feltörni a titkosítást, ha ismerjük a titkosítási folyamatot.

5.2. Kódolás

Kódolásnak nevezzük azt a folyamatot, ahol információt valamilyen módon reprezentálunk. A folyamat résztvevői az adat, a kódoló, és a dekódoló. Ebben az esetben nem

fontos, hogy az új formátumból ne lehessen visszafejteni, tehát a biztonság nem a főszempontja a folyamatnak. Kódolást használunk például a következő esetben: ha analóg jelet digitálisra vagy digitális jelet analógra konvertálunk.

5.2.1. Base64 kódolás

A base64 kódolás egy olyan kódolási folyamat, ami során egy bináris adatot úgy módosítja, hogy egy másik bináris adattá változtatja át. Ilyenkor az adat degradálódik, mivel a kiinduló karakter halmaz nagyobb, mint a karakter halmaz, amivel reprezentálva lesz a folyamat után. A base64 kódolást könnyen fel lehet ismerni, mivel a latin ábécé nagy és kisbetűit, nullától kilencig a számokat és a + és a / két speciális karaktert. Ezek mellett, amiről még könnye felismerhetjük, hogy gyakran egy vagy két egyenlőségjel található a karakterlánc végén. [47]

5.3. Hashelés

Hashelésnek nevezzük azt a folyamatot, ahol egy változó hosszúsággal rendelkező bemeneti értékből egy algoritmussal egy állandó méretű kimeneti értéket állítunk elő. Fontos megjegyezni, hogy különböző bemeneti értékek, nem adhatnak megegyező kimeneti értéket, ezért használják a hashelést integritás ellenőrzésre.

5.4. Obfuszkáció

„Az obfuszkáció egy olyan program-transzformáció, mely a program eredeti funkcionalitásának megőrzésével (esetleg tolerálható teljesítménycsökkenés árán) a program leírását (forráskódját) vagy akár a működését változtatja meg, azzal a céllal, hogy a programban információt rejtessen el (pl. a program működéséről vagy a futás során használt változók értékeiről). A pontos céltól és a felhasznált eszközöktől függően az obfuszkáció két megközelítése ismert. A szoftverfejlesztés területén elterjedt kód-obfuszkáció kifejezés olyan obfuszkációs technikákra utal, melyek célja megnehezíteni a program működésének megértését (deobfuszkációját), statikus vagy dinamikus elemzését, ismeretlen forráskód esetén a kód visszafejtését (reverse-engineering). Ezen heurisztikus eljárásokkal szemben az ún. kriptográfiai obfuszkáció célja programok „titkosítása” (az üzenet titkosítás analógiájára) pontosan definiált biztonsági garanciák teljesítése érdekében.” [48] Az obfuszkációnak a célja a „Clean Coding”-nak a teljes ellentétje, hogy a kód ne legyen jól olvasható.

5.4.1. Fájlformátum hamisítás

Ennek a technikának a célja, hogy egy káros fájlt, egy más formátumként álcázzon, hogy ezzel bírja az áldozatot kattintásra. Ilyenkor két dolgot kell meg figyelembe vennünk. Se a fájl

ikonja, se a kiterjesztése nem utalhat arra, hogy valójában, milyen fájl formátummal rendelkezik. A fájlformátum hamisításához például használhatnak Resource Hacker-t, hogy megváltoztassák az ikont, vagy WinRar-t is, hogy egy sfx archívumba csomagolják a különböző forrásokat.

5.4.2. Zaj generálás

Ez egy olyan technika, amelyet az obfuszkálást végző fejlesztők részben két dolog miatt alkalmaznak. Először is, hogy a visszafejtés folyamatát megnehezítsék, másodszor, hogy a különböző védelmi rendszereket megpróbálják kijátszani.

Ilyen technika lehet az, több olyan komment marad a kódban, ami csak értelmetlen karakterláncokat tartalmaz, hogy a visszafejtési folyamatot végző személy idejét pocsékolják azzal, hogy a kódot egy olvasható formátumra hozzák.

Egy másik módszer, amit obfuszkációhoz alkalmaznak, hogy a definiált függvények és változók neveit véletlenszerűen adják meg. Deklarálnak olyan változókat és létrehoznak olyan függvényeket, amelyek nem a használnak, vagy csak figyelem elterelésként nem relevánsak a szoftver működésében.

6. Összegzés

6.1. Elemzési eredmények

Az elemzésünk során sikerült a szoftvert beazonosítanunk, a hash-ek feljegyzésével és megállapítanunk azt a tényt, hogy egy kifejezetten fejlett kártékony kóddal van dolgunk, ami rendkívül érzékeny az elemzési környezetre.

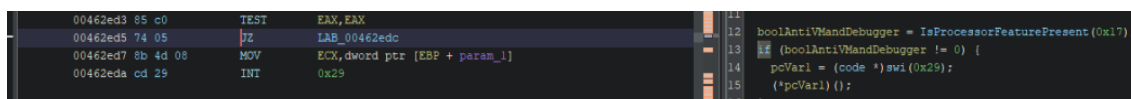
6.2. Következő lépések

6.2.1. A python script tovább fejlesztése

Itt érdemes az interneten információt keresni még, hogy milyen függvények azok, amelyeket a kártékony szoftverek használnak. Esetleg érdemes lehet még hozzáadni egy adatbázist, hogy az elemzett fájloknak a hash-eit és a bennük tárolt függvény neveket összehasonlítsuk.

6.2.2. Patch készítés a teljeskörű dinamikus elemzéshez

Ha szeretnénk mélyebben dinamikusán elemezni a kártékony szoftverünket, akkor érdemes egy patchet készíteni, hozzá, ahol az összes anti-vm és anti-debugging függvényhívásokat lefegyverezni.



23. ábra Ghidra Anti-Debugging Bypass

Ahogy a fenti ábrán látható, hogy hogyan néz ki egy ilyen módszer assemblyben és a Ghidra által visszafordított pszeudokódban. Itt jól látható, hogy a „ha” vezérlési szerkezet hardver szinten egy TEST és egy JZ instrukcióból áll. Ha itt a JZ instrukciót megváltoztatjuk JMP-re, akkor a TEST lefut viszont nem az eredmény szerint halad tovább a program. Ezt a módszert használva ki kell javítanunk ezeket az ellenőrzési függvényeket, hogy ne legyen hatása a szoftver működésére, hogy eredményesen tudjunk dinamikusán elemezni a szoftvert.

6.2.3. YARA szabály készítése

A fentebb feltárt információkból képesek vagyunk létrehozni egy YARA szabályt, amivel azonosítani tudunk bináris fájlokat, ezáltal képesek vagyunk szűrni a kártékony szoftvereket. A következőben látható egy ilyen YARA szabály, amivel az általunk vizsgált szoftvert vagy annak karakterisztikáját tudjuk felismerni.

```

rule prynt_malware
{
    meta:
        author = "Maklari Peter"
        date = "2024.05.01"
        malware = "Prynt_Malware"

    strings:
        $is_pe_file = { 4D 5A }
        $ip_address = "194.169.175.128"
        $SQL_String =
"REINDEXEDESCAPEACHECKKEYBEFOREIGNOREGEXPLAINSTEADDDATABASELECTABLEFTHENDEFERRABLELSEXC
LUDELETEMPORARYISNULLSAVEPOINTINTERSECTIESNOTNULLIKEXCEPTTRANSACTIONATURALTERAISEXCLUSIVE
XISTSCONSTRAINTOFFSETRIGGERREFERENCESUNIQUEQUERYWITHOUTERELEASEATTACHAVINGLOBEGINNERANGE
ETWEENOTHINGROUPSCASCADETACHCASECOLLATECREATECURRENT_DATEIMMEDIATEJOININSERTMATCHPLANAL
YZEPRAGMABORTUPDATEVALUESVIRTUALASTWHENWHERECURSIVEAFTERRENAMEANDEFAULTAUTOINCREMENTCA
STCOLUMNCOMMITCONFLICTCROSSCURRENT_TIMESTAMPARTITIONDEFERREDISTINCTDROPCECEDINGFAILIM
ITFILTERREPLACEFIRSTFOLLOWINGFROMFULLIFORDERESTRICTOTHERSOVERIGHTROLLBACKROWSUNBOUNDED
UNIONUSINGVACUUMVIEWWINDOWBYINITIALLYPRIMARY"

    condition:
        ($is_pe_file and $ip_address and $SQL_String) or ($ip_address and
$SQL_String)
}

```

6.3. Záró gondolatok

A dolgozatban a Windows operációs rendszer volt a célpont, de nem szabad elfelejteni, hogy a hiedelmek ellenére a Linux-os környezetben is vannak kártékony kódok, mivel minden nyílt forráskódú, ezért relatív jobban átlátható a rendszer. A választás azért esett a Windows-os kártékony kódokra, mivel ez az elsődleges operációs rendszer, amit majdnem minden felhasználó nap mint nap használatba vesz.

A statikus elemzésünk hasznos volt azzal kapcsolatban, hogy sikeresen tudtunk annyi információt gyűjteni a szoftverrel kapcsolatban, hogy jól betudjuk azonosítani.

Az dinamikus elemzés során a saját környezetünkben nem volt jó lehetőségünk, elemezni a szoftvert, mivel egy patch készítése túl sok időt venne igénybe és a dolgozat terjedelmi határán túl nyúlna.

7. Hivatkozások

- [1] „What is the CIA Triad and Why is it important? | Fortinet,” Fortinet, [Online]. Available: www.fortinet.com/resources/cyberglossary/cia-triad. [Hozzáférés dátuma: 15 2024].
- [2] „Social Engineering - mi az és hogyan működik?,” ESET, [Online]. Available: www.eset.com/hu/it-biztonsagi-temak-cegeknek/social-engineering/. [Hozzáférés dátuma: 21 4 2024].
- [3] „What is a Zero-Day Exploit?,” IBM, [Online]. Available: <https://www.ibm.com/topics/zero-day>.
- [4] „How to Sell Your Zero-Day (0day) Exploit to ZERODIUM,” ZERODIUM, [Online]. Available: zerodium.com/program.html. [Hozzáférés dátuma: 15 12 2023].
- [5] „NATO - Cyber defence,” NATO, 14 9 2023. [Online]. Available: www.nato.int/cps/en/natohq/topics_78170.htm. [Hozzáférés dátuma: 16 04 2024].
- [6] C. Griffiths, „The Latest Cyber Crime Statistics (updated May 2024),” AAG IT Support, 5 2024. [Online]. Available: <https://aag-it.com/the-latest-cyber-crime-statistics/>.
- [7] W. Zamora, „What's the difference between antivirus and anti-malware?,” malwarebytes Labs, 11 9 2015. [Online]. Available: www.malwarebytes.com/blog/news/2015/09/whats-the-difference-between-antivirus-and-anti-malware. [Hozzáférés dátuma: 29 4 2024].
- [8] „What is a Firewall? How Firewalls Work & Types of Firewalls,” Kaspersky, [Online]. Available: www.kaspersky.com/resource-center/definitions/firewall. [Hozzáférés dátuma: 26 3 2024].
- [9] „What is an Intrusion Detection System (IDS)? | IBM,” IBM, [Online]. Available: www.ibm.com/topics/intrusion-detection-system. [Hozzáférés dátuma: 14 4 2024].
- [10] „What is HIDS (Host-Based Intrusion Detection System)?,” Sysdig, [Online]. Available: <https://sysdig.com/learn-cloud-native/detection-and-response/what-is-hids/>. [Hozzáférés dátuma: 20 4 2024].

- [11] „Difference between HIDs and NIDs,” Geeks for Geeks, 12 7 2022. [Online]. Available: www.geeksforgeeks.org/difference-between-hids-and-nids/. [Hozzáféréstuma: 6 4 2024].
- [12] „Mi a különbség az IDS (Intrusion Detection System) és az IPS (Intrusion evention System) között?,” Makay Kiberbiztonsági Kft., [Online]. Available: makay.net/ids-ips-kulonbseg-intrusion-detection-prevention-system. [Hozzáféréstuma: 6 4 2024].
- [13] M. K. P. Kinza Yasar, „What is Security Awareness Training?,” TechTarget, 10 23. [Online]. Available: www.techtarget.com/searchsecurity/definition/security-areness-training. [Hozzáférés dátuma: 25 4 2024].
- [14] „The 5 Stages of an Effective Cyber Defence Strategy,” IT Governance Blog En, 9 2022. [Online]. Available: www.itgovernance.eu/blog/en/the-5-stages-of-an-ffective-cyber-defence-strategy. [Hozzáférés dátuma: 21 4 2024].
- [15] „Rootkit | ESET Glossary,” Eset, 19 4 2023. [Online]. Available: lp.eset.com/glossary/hu-HU/rootkits.html. [Hozzáférés dátuma: 24 4 2023].
- [16] E. R. S. B. Alex Matrosov, Rootkits and Bootkits, Reversing Modern Malware and ext Generation Threats, 245 8th Street, San Francisco, CA 94103: No Starch Press Inc., 19.
- [17] „What Is Fileless Malware?,” Trellix, [Online]. Available: www.trellix.com/security-awareness/ransomware/what-is-fileless-malware/. [Hozzáférés dátuma: 23 3 2024].
- [18] „A Brief History of Malware,” Lifewire, 9 2 2023. [Online]. Available: <https://www.lifewire.com/brief-history-of-malware-153616>. [Hozzáférés dátuma: 15 4 24].
- [19] J. Schneider, „The history of malware: A primer on the evolution of cyber threats,” IBM, 6 11 2023. [Online]. Available: www.ibm.com/blog/malware-history/. [Hozzáféréstuma: 12 4 2024].
- [20] „Malware Development - A History,” Radware Application and Network Security, [online]. Available: https://www.radware.com/resources/malware_timeline.aspx/.

- [21] N. Bene, „ILOVEYOU – 20 years ago – to the day!,” Eugene Kaspersky’s Official Blog, 5 5 2020. [Online]. Available: eugene.kaspersky.com/2020/05/05/iloveyou-20-years-ago-to-the-day/. [Hozzáférés dátuma: 1 4 2024].
- [22] C. András, „Cserhádi András: A STUXNET vírus,” *Fizikai Szemle*, %1. kötet5, p. 0, 2011.
- [23] J. Rhysider, „Stuxnet,” Darknet Diaries, 2 1 2019. [Online]. Available: darknetdiaries.com/transcript/29/. [Hozzáférés dátuma: 24 12 2023].
- [24] T. Menduk, „Stuxnet Simulator made from Pi and Siemens S7-300 PLC,” 7 9 2023. [Online]. Available: www.youtube.com/watch?v=0SXvWONAGk8. [Hozzáférés dátuma: 9 4 2024].
- [25] J. Rhysider, „WannaCry,” Darknet Diaries, 1 9 2020. [Online]. Available: <https://darknetdiaries.com/episode/73/>.
- [26] J. Walker, „Running WannaCry in a Virtual Machine,” 28 2 2020. [Online]. Available: jakew.me/wannacry-vm/. [Hozzáférés dátuma: 6 5 2024].
- [27] „What are Petya and NotPetya Ransomware?,” Malwarebytes, [Online]. Available: www.malwarebytes.com/petya-and-notpetya. [Hozzáférés dátuma: 12 4 2024].
- [28] J. Rhysider, „NotPetya,” Darknet Diaries, 24 12 2019. [Online]. Available: www.darknetdiaries.com/episode/54/. [Hozzáférés dátuma: 19 12 2023].
- [29] J. Rhysider, „NSO,” Darknet Diaries, 31 8 2021. [Online]. Available: darknetdiaries.com/episode/100/. [Hozzáférés dátuma: 31 12 2023].
- [30] „VMware, Inc. - Company Profile,” California Explore, [Online]. Available: californiaexplore.com/company/2069963/vmware-inc.
- [31] A. M. M. Gómez, „mandiant/flare-vm: A collection of software installations scripts for Windows systems that allows you to easily setup and maintain a reverse engineering environment on a VM.,” Mandiant, [Online]. Available: github.com/mandiant/flare-vm.
- [32] „Get the Virtual Appliance,” REMnux, [Online]. Available: cs.remnux.org/install-distro/get-virtual-appliance. [Hozzáférés dátuma: 25 12 2023].
- [33] HuskyHacks, „Malware Analysis In 5+ Hours - Full Course - Learn Practical Malware Analysis!,” 22 8 2022. [Online]. Available:

- www.youtube.com/watch?v=qA0YcYMRWyI&t=3235s. [Hozzáférés dátuma: 24 12 2023].
- [34] NSA, „NationalSecurityAgency/ghidra: Ghidra is a software reverse engineering RE) framework,” NSA, [Online]. Available: github.com/NationalSecurityAgency/ghidra.
- [35] „Wireshark · About,” Wireshark, [Online]. Available: <https://www.wireshark.org/about.html>.
- [36] 0xpat, 30 March 2020. [Online]. Available: [pat.github.io/Malware_development_part_1/](https://github.com/0xpat/Malware_development_part_1). [Hozzáférés dátuma: 8 6 2023].
- [37] Corkami, „GitHub,” [Online]. Available: github.com/corkami/pics/blob/master/binary/pe101/pe101.png.
- [38] 0xrick, „A dive into the PE file format - PE file structure - Part 2: DOS Header, OS Stub and Rich Header,” 22 10 2021. [Online]. Available: [0xrick.github.io/win-internals/pe3/](https://github.com/0xrick/win-internals/pe3/). [Hozzáférés dátuma: 16 4 2024].
- [39] 0xrick, „A dive into the PE file format - PE file structure - Part 5: PE Imports Import Directory Table, ILT, IAT),” 28 10 2021. [Online]. Available: [rick.github.io/win-internals/pe6/](https://github.com/0xrick/win-internals/pe6/). [Hozzáférés dátuma: 16 4 2024].
- [40] „VirusTotal - File - 4aa12984de37ac1fb650f1196e88fb5a64707e571382d641e1c70ce82e89cf0a,” VirusTotal, [Online]. Available: <https://www.virustotal.com/gui/file/4aa12984de37ac1fb650f1196e88fb5a64707e571382d641e1c70ce82e89cf0a/detection>.
- [41] „Common API used in Malware,” HackTricks, [Online]. Available: book.hacktricks.xyz/reversing/common-api-used-in-malware.
- [42] „194.169.175.128 IP Reputation Lookup,” IP Address Reputation Check, [Online]. Available: www.ipqualityscore.com/ip-reputation-check/lookup/194.169.175.128.
- [43] R. Miles, „Public Key Cryptography - Computerphile - YouTube,” Computerphile, 7 2014. [Online]. Available: https://www.youtube.com/watch?v=GSIDS_1vRv4. [Hozzáférés dátuma: 18 10 2022].
- [44] „Caesar Cipher (Shift) - Online Decoder, Encoder, Solver, Translator,” [Online]. Available: www.dcode.fr/caesar-cipher.

- [45] A. Vance, „How the RSA algorithm works, including how to select d , e , n , p , q , d φ (phi),” 14 10 2014. [Online]. Available: www.youtube.com/watch?v=Z8M2BTscoD4.
- [46] D. N. Benedek, „Új számítási paradigmák 13.fejezet 2.alfejezet 1.rész. - /antumalgoritmusok,” Debreceni Egyetem, 2013. [Online]. Available: <https://gyires.inf.unideb.hu/GyBITT/28/ch13s02.html>.
- [47] „Base64 Encode and Decode - Online,” [Online]. Available: www.base64encode.org/.
- [48] „Obfuszkáció - Wikik - Fogalomtár,” HTE, [Online]. Available: www.fogalomtar.hte.hu/wiki/-iki/HTE+Infokommunikacios+Fogalomtar/Obfuszk%C3%A1ci%C3%B3/pop_up. [hozzáférés dátuma: 15 2024].
- [49] „ANY.RUN - Interactive Online Malware Sandbox,” [Online]. Available: any.run.

8. Ábrajegyzék

1. ábra Sérülékenységek ára [4]	7
2. ábra ILOVEYOU email [21]	17
3. ábra STUXNET demo [24]	18
4. ábra WannaCry fertőzött asztal [26]	19
5. ábra A környezet felépítése	23
6. ábra FlareVM ipconfig	23
7. ábra remnux ip a	24
8. ábra PE fájl felépítése [37]	27
9. ábra Prynt hashek	28
10. ábra Virus Total értékelés	29
11. ábra python script eredménye	31
12. ábra detect it easy felülete	32
13. ábra Entrópia vizsgálat eredménye	33
14. ábra Ghidra főképernyő	34
15. ábra Ghidra automatikus elemzési lista	34
16. ábra Ghidra karakterláncok szűrése	35
17. ábra Karakterlánc szűrés eredmény	35
18. ábra Ghidra SQL karakterláncok	36
19. ábra IDA Freeware WinMain	36
20. ábra Ghidra WinMain	37
21. ábra x32dbg grafikus felülete	38
22. ábra Wireshark hálózati kommunikáció	38
23. ábra Ghidra Anti-Debugging Bypass	44

9. Mellékletek

1. sz.melléklet „PrivateLoader.json” [49]

```
{
  "C2": [
    "discord.com/api/v9/users/@me",
    "api64.ipify.org/?format=json",
    "ipinfo.io/widget/demo/",
    "db-ip.com/demo/home.php?s="
  ],
  "Strings": [
    "winhttp.dll",
    "wininet.dll",
    "LocalSimbl",
    "LocalSimba",
    "grab_screen",
    "grab_tg",
    "grab_ds",
    "grab_wallets",
    "grab_ihistory",
    "logins",
    "Vault_IE",
    "WindowsCredentials",
    "\\screenshot.png",
    "\\Files",
    "\\FileZilla",
    "\\Plugins",
    "IndexedDB",
    "Local",
    "\\Wallets",
    "%s\\t%llu\\r\\n",
    "nickname",
    "name_on_card",
    "card_number",
    "last_four",
    "**** *",
    "billing_address_id",
    "exp_month",
    "exp_year",
    "expiration_month",
    "\\Autofill",
    "value",
    "%s\\t%s\\r\\n",
    "\\Downloads",
    "%s\\n%s\\n\\n",
    "domain",
    "expirationDate",
    "secure",
```

```

"FALSE",
"httpOnly",
"%s\t%s\t%s\t%s\t%llu\t%s\t%s\r\n",
"\\passwords.txt",
"login",
"password",
"profile",
"\nStorage: %s [%s]\nURL: %s\nLogin: %s\nPassword: %s\n\n",
"\\discord.txt",
"\nStorage: %s\nUserName: %s\nE-MAIL: %s\nToken: %s\n\n",
"nss3.dll",
"autofill",
"download_history",
"cookies",
"history",
"Authenticator",
"bhghoamapcdpbohphigooaddinpkbai",
"MetaMask",
"nkbihfbeogaeaoehlefnkodbefgpgknn",
"Jaxx Liberty Extension",
"cjelfplplebdjjenllpjcbmljkfcffne",
"iWallet",
"kncchdigobghenbbaddojjnaogfppfj",
"BitAppWallet",
"fihkakfobkmkjojpchpfgcmhfjnmnmpi",
"SaturnWallet",
"nkddgncdjgjfcdamfgcmfnlhccnimig",
"GuildWallet",
"nanjmdknhkinifnkgdcggcfnhdaammj",
"MewCx",
"nlbmnnijcnlegkjpcfjclmcfggfefdm",
"Wombat",
"amkmjmmflddogmhpjloimipbofnfjih",
"CloverWallet",
"nhnkbkgjikgcigadomkphalanndcapjk",
"NeoLine",
"cphhlgmgameodnhkjdmkpanlelnlohao",
"RoninWallet",
"fnjhmkhmkbjkkabndcnogagobneec",
"LiqualityWallet",
"kpfpkelmapcoipemfendmdcghnegimn",
"EQUALWallet",
"blnieiiffboillknjnepogjhkgnoapac",
"Guarda",
"hpglfhgfnhbpgjdenjgmdgoeiappafln",
"Coinbase",
"Local State",
>Login Data",
>Login Data For Account",
"Web Data",
"History",

```

```

"cards",
"Cookies",
"An uncaught exception occurred1: ",
"An uncaught exception occurred1. The type was unknown so no information was
available.",
"\\Mozilla\\Firefox",
"Firefox",
"\\Waterfox",
"Waterfox",
"\\K-Meleon",
"K-Meleon",
"\\Thunderbird",
"Thunderbird",
"\\Comodo\\IceDragon",
"IceDragon",
"\\8pecxstudios\\Cyberfox",
"Cyberfox",
"\\NETGATE Technologies\\BlackHaw",
"BlackHaw",
"\\Moonchild Productions\\Pale Moon",
"Pale Moon",
"\\Discord",
"Discord",
"\\discordcanary",
"DiscordCanary",
"\\discordptb",
"DiscordPTB",
"\\discorddevelopment",
"DiscordDevelopment",
"\\Opera Software",
"Opera",
"\\Google\\Chrome\\User Data",
"Chrome",
"\\Microsoft\\Edge\\User Data",
"\\BraveSoftware\\Brave-Browser\\User Data",
"Brave",
"\\CryptoTab Browser\\User Data",
"CryptoTab",
"\\Battle.net",
"Battle.net",
"\\Chromium\\User Data",
"Chromium",
"\\Google(x86)\\Chrome\\User Data",
"Chrome (x86)",
"\\Yandex\\YandexBrowser\\User Data",
"Yandex",
"\\NVIDIA Corporation\\NVIDIA GeForce Experience",
"NVIDIA",
"\\Steam",
"Steam",
"\\Amigo\\User\\User Data",
"Amigo",

```

```

"\\Iridium\\User Data",
"Iridium",
"\\MapleStudio\\ChromePlus\\User Data",
"ChromePlus",
"\\7Star\\7Star\\User Data",
"rule_exceptions",
"rule_files",
"rule_folder",
"rule_size_kb",
"rule_collect_recursv",
"%DESKTOP%",
"%DOCUMENTS%",
"%USERPROFILE%",
"%APPDATA%",
"%LOCALAPPDATA%",
"%RECENT%",
"ld_marks",
"ld_geo",
"ld_url",
"mark_check_cookies",
"mark_check_passwords",
"mark_check_history",
"mark_domains",
"\\Telegram Desktop",
"\\tdata",
"\\key_dats",
"\\maps",
"\\Telegram",
"VaultEnumerateItems",
"VaultEnumerateVaults",
"VaultGetItem",
"\\profiles.ini",
"Profile",
"\\places.sqlite",
"SELECT place_id, visit_date FROM(SELECT place_id, visit_date, id FROM
moz_historyvisits ORDER BY id DESC LIMIT 2500) ORDER BY id ASC",
"SELECT url FROM moz_places WHERE (`id` = ",
"\\cookies.sqlite",
"SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies",
"SELECT place_id, content FROM moz_annos WHERE (`anno_attribute_id` = 1)",
"file:/// ",
"\\formhistory.sqlite",
"SELECT fieldname, value FROM moz_formhistory",
"\\logins.json",
"\\signons.sqlite",
"SELECT encryptedUsername, encryptedPassword, formSubmitURL FROM moz_logins",
"formSubmitURL",
"encryptedUsername",
"encryptedPassword",
"PK11_Authenticat",
"\\Local State",
"os_crypt",

```

```

"encrypted_key",
"Local Storage\\leveldb",
"dQw4w9WgXcQ:[^.*\\['(.*')\\].*$][^\\"]*",
"dQw4w9WgXcQ:",
"discord.com/api/v9/users/@me",
"username",
"email",
"\\CURRENT",
"\\Sync Extension Settings\\",
"\\Local Extension Settings\\",
"_0.indexeddb.leveldb\\CURRENT",
"\\IndexedDB\\chrome-extension_",
"_0.indexeddb.leveldb",
"SELECT url, last_visit_time FROM(SELECT url, last_visit_time, id FROM urls ORDER
BY id DESC LIMIT 2500) ORDER BY id ASC",
"\\Network",
"Network\\",
"SELECT host_key, is_httponly, path, is_secure, expires_utc, name, value,
encrypted_value FROM cookies",
"SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted,
origin, billing_address_id, nickname FROM credit_cards",
"expiration_year",
"origin",
"SELECT name_on_card, exp_month, exp_year, last_four, nickname, bank_name,
card_art_url, status, network FROM masked_credit_cards",
"SELECT tab_url, target_path FROM downloads",
"SELECT name, value FROM autofill",
"SELECT action_url, origin_url, username_value, password_value FROM logins",
"softkn3.dll",
"msvcpl140.dll",
"vcruntime140.dll",
"An uncaught exception occurred3: ",
"An uncaught exception occurred3. The type was unknown so no information was
available.",
"api.myip.com",
"An uncaught exception occurred_ip0_1: ",
"An uncaught exception occurred_ip0_1. The type was unknown so no information was
available.",
"api64.ipify.org/?format=json",
"An uncaught exception occurred_ip0_2: ",
"An uncaught exception occurred_ip0_2. The type was unknown so no information was
available.",
"ipinfo.io/widget/demo/",
"country",
"An uncaught exception occurred_ip1: ",
"An uncaught exception occurred_ip1. The type was unknown so no information was
available.",
"db-ip.com/demo/home.php?s=",
"demoInfo",
"countryCode",
"freebl3.dll",
"mozglue.dll",
"\\atomic\\Local Storage",

```



```

"\\Atomic",
"\\Electrum\\wallets",
"\\Electrum",
"\\Exodus\\exodus.wallet",
"\\Exodus",
"\\Electrum-LTC\\wallets",
"\\ElectrumLTC",
"\\Monero\\wallets",
"\\Monero",
"\\com.liberty.jaxx",
"\\Jaxx Liberty",
"\\IndexedDB",
"\\Local Storage",
"\\Session Storage",
"\\Jaxx\\Local Storage",
"\\Jaxx",
"\\Coinomi\\Coinomi\\wallets",
"\\Coinomi",
"\\Armory",
"\\WalletWasabi\\Client\\Wallets",
"\\Wasabi",
"\\Bither\\bither.db",
"\\Bither",
"\\bither.db",
"\\ElectronCash\\wallets",
"\\ElectronCash",
"\\Binance\\app-store.json",
"\\wallet.dat",
"\\wallets",
"\\Authy Desktop",
"\\Authy",
"Version: %s\\n\\n",
"Date: %s",
"Unknown",
"SOFTWARE\\Microsoft\\Cryptography",
"MachineGuid",
"MachineID: %s\\n",
"GUID: %s\\n",
"HWID: %s\\n",
"\\nPath: %s\\n",
"Work Dir: %s\\n\\n",
"IP: %s\\n",
"Location: %s, %s\\n",
"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
"ProductName",
"Windows: %s [%s]\\n",
"Computer Name: %s\\n",
"User Name: %s\\n",
"Display Resolution: %dx%d\\n",
"Display Language: %ws\\n",
"Display Language: Unknown\\n",

```

```

"Keyboard Languages: ",
" / %s",
"\nLocal Time: %d/%d/%d %d:%d:%d\n",
"TimeZone: UTC%d\n",
"\n[Hardware]\n",
"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
"ProcessorNameString",
"http://",
"WinHttpRequestDataAvailable",
"WinHttpRequest",
"WinHttpRequestReceiveResponse",
"WinHttpRequestCloseHandle",
"WinHttpRequestSetTimeouts",
"InternetQueryOptionA",
"HttpRequestA",
"InternetReadFile",
"InternetCloseHandle"
]
}

```

2. sz.melléklet „RisePro.json” [49]

```

{
  "C2": "194.169.175.128",
  "Strings": [
    ".zip",
    "WindowsCredentials",
    "Vault_IE",
    "grab_ihistory",
    "grab_wallets",
    "grab_ds",
    "grab_tg",
    "grab_screen",
    "LocalSimba",
    "LocalSimbl",
    "wininet.dll",
    "winhttp.dll",
    "\\Files",
    "\\screenshot.png",
    "\\Wallets",
    "Local",
    "Sync",
    "IndexedDB",
    "\\Plugins",
    "\\FileZilla",
    "Name: %s\r\nNickname: %s\r\nMonth: %s\r\nYear: %s\r\nCard: %s\r\nAddress: %s\r\n\r\n",
    "-",
    "**** *",
    "\\CC",
    "%s\t%llu\r\n",

```

```

"\\History",
"%s\n%s\n\n",
"\\Downloads",
"%s\t%s\r\n",
"\\Autofill",
"FALSE",
"TRUE",
"%s\t%s\t%s\t%s\t%llu\t%s\t%s\r\n",
"ab",
".txt",
"\\Cookies",
"\nStorage: %s\nUserName: %s\nE-MAIL: %s\nToken: %s\n\n",
"\\discord.txt",
"\nStorage: %s [%s]\nURL: %s\nLogin: %s\nPassword: %s\n\n",
"\\passwords.txt",
"Trezor Password Manager",
"imloifkgjagghnncjkhggdhalmcnfklk",
"GAAuth Authenticator",
"ilgcnhelpchnceei pipijaljkblbcobl",
"EOS Authenticator",
"oeljdldpnmdbchoni elidgobddffflal",
"Trust Wallet",
"egjidjbpglichdcondbcdbnbeppgdph",
"Leap Terra Wallet",
"aijcbedoi jmgnlmjeegjaglmepbmkpi",
"Finnie",
"cjmkn djhnagcfbpiemnkdpomccnjblmj",
"EMartian Aptos Wallet",
"efbglgofoippbgcjepnhiblaibcnclgk",
"Opera Wallet",
"gojhcdgcbbpfigcaejpfhfegekdgiblk",
"Petra Aptos Wallet",
"ejjladinnckdgjemekebdpeokbikhfci",
"Pontem Aptos Wallet",
"phkbamefinggmakgklpkljmgibohnba",
"Gerowallet",
"bgpipimickeadkjlklgciifhnalhdjhe",
"Eternl",
"kmhcihpebfmpgmihbkipmjlmioameka",
"Hashpack",
"gjagmgiddbbciopjhllkdnddhcglnemk",
"Sender Wallet",
"epapihdplajcdnnkdeiahlgigofloibg",
"OKX Wallet",
"mcohilncbfahbmgdjkbpemcciolgcge",
"Eth and Polk Web3 Wallet",
"kkp1lkodjeloidieedojo gacfhpai hoh",
"Braavos wallet",
"jnlgamecbpmbajjfhhmmmlhejkemejdma",
"Goby",
"jnkelfanjkeadonecabe halmbgpfodjm",

```

"Temple",
"ookjlbkiijinhpnmjffcofjonbfbgaoc",
"TezBox",
"mnfifekajgofkckjemidiaecocnkjeh",
"KHC",
"hcflpincpppdclinealmandijcmnkbgn",
"CyanWallet",
"dkdedlpgdmmkffjabffeganieamfklm",
"Solflare",
"bhhhlbepdkbapadjdnnojkbgioiodbic",
"WavesKeeper",
"lpilbniabackdjcionkobglmddfbcjo",
"BraveWallet",
"odbfpeeihdkbihmopkbjmoonfanlbfc1",
"Rabby",
"acmacodkjbdgmoleebolmdjonilkdbch",
"EVER Wallet",
"cgeeodpfagjceefiefldmfphplkenlfk",
"ICONex",
"flpiciilemghbmfalicajoolhikkenfel",
"PolymeshWallet",
"jojhfeoedkpglbfimdfabpdfjaoolah",
"AuroWallet",
"cnmamaachppnkjgnildpdmkaakejnhae",
"Sollet",
"fhmfendgdocmbmfikdcogofphimnkno",
"Keplr",
"dmkamcknogkgcdfhbbddcghachkejeap",
"afbcbjpbpfadlkmhmc1hkeeodmamcflc",
"Exodus_E",
"aholpfdialjggfhomihkjbmgidldcno",
"Nami",
"lpfcbjknijpeeillifnkikgncikgfhd",
"Harmony",
"fnnegphlobjdpkhecapkijjdkgcjhkib",
"Terra",
"aiifbnbfbopmeekipheeijimdpnlpgpp",
"coin98",
"aeachknmefpheapccionboohckonoeemg",
"KardiaChain",
"pdadjkfkcgafgbceimcpbkalfnepbnk",
"Maiar DeFi Wallet",
"dngmlblcodfobpdpecaadgfbcggfjfnm",
"XDEFI Wallet",
"hmeobnfnfcmkdcmlblgagmfpfboieaf",
"ForboleX",
"fmb1appgoiilbgafhjklehffibdocee",
"Bolt X",
"aodkkagnadcbobfpggfnjeongembjbca",
"PaliWallet",
"mgffkfbidihjpoaomajlbghddlicgpn",

```

"Oxygen",
"fhilaheimglignddkjgofkcbgekhenbh",
"Phantom",
"bfnaelmomeimhlpmgjnjophhpkkoljpa",
"TronLink",
"ibnejdfjmmkpcnlpebklmnkoeiohofec",
"BinanceChainWallet",
"fhbohimaebobhpjbbldcngcnapndodjp",
"Yoroi",
"ffnbelfdoeiohenkjibnmadjiehjhajb",
"NiftyWallet",
"MathWallet",
"jbdacneiiniimbjlgalhcelgbejmnid",
"Coinbase",
"hnfanknocfeofbddgcijnmhnfnkdnaad",
"Guarda",
"hpglfhgfnhbpgjdenjgmdgoeiappafln",
"EQUALWallet",
"blnieiiffboillknjnepogjhgknoapac",
"LiqualityWallet",
"kpfpokelmapcoipemfendmdcghnegimn",
"RoninWallet",
"fnjhmkhmkbjkkabndcnogagobneec",
"NeoLine",
"cphhlmgameodnhkjdmkpanlelnlohao",
"CloverWallet",
"nhnkbkgjikgcigadomkphalanndcapjk",
"Wombat",
"amkmjmmflldogmhpjloimipbofnfjih",
"MewCx",
"nlbmnnijcnlegkjpcfjclmcfggfefdm",
"GuildWallet",
"nanjmdknhkinifnkgdgcgcfnhdaammj",
"SaturnWallet",
"nkddgncdjgjfcdamfgcmfnlhccnimig",
"BitAppWallet",
"fihkakfobkmkjojpchpfgcmhfjnmnfpi",
"iWallet",
"kncchdigobghenbbaddojjnaogfppfj",
"Jaxx Liberty Extension",
"cjelfplplebdjjenllpjcbmljkcfcffne",
"MetaMask",
"nkbihfbeogaeaoehlefnkodbefgpgknn",
"Authenticator",
"bhghoamapcdpbohphigoooaddinpkbai",
"An uncaught exception occurred1. The type was unknown so no information was
available.",
"An uncaught exception occurred1: ",
>Login Data For Account",
"\\Moonchild Productions\\Pale Moon",
"Pale Moon",
"\\NETGATE Technologies\\BlackHaw",

```

```
"BlackHaw",
"\\8pecxstudios\\Cyberfox",
"Cyberfox",
"\\Comodo\\IceDragon",
"IceDragon",
"\\Thunderbird",
"Thunderbird",
"\\K-Meleon",
"K-Meleon",
"\\Waterfox",
"Waterfox",
"\\Mozilla\\Firefox",
"Firefox",
"\\NetboxBrowser\\User Data",
"NetboxBrowser",
"\\Mail.Ru\\Atom\\User Data",
"Atom",
"\\Chromodo\\User Data",
"Chromodo",
"\\Uran\\User Data",
"Uran",
"\\CocCoc\\Browser\\User Data",
"CocCoc",
"\\Nichrome\\User Data",
"Nichrome",
"\\Sputnik\\Sputnik\\User Data",
"Sputnik",
"\\K-Melon\\User Data",
"K-Melon",
"\\Maxthon3\\User Data",
"Maxthon3",
"\\360Browser\\Browser\\User Data",
"360Browser",
"\\Comodo\\User Data",
"Comodo",
"\\Torch\\User Data",
"Torch",
"\\Comodo\\Dragon\\User Data",
"Dragon",
"\\Orbitum\\User Data",
"Orbitum",
"\\QIP Surf\\User Data",
"QIP Surf",
"\\liebao\\User Data",
"liebao",
"\\Coowon\\Coowon\\User Data",
"Coowon",
"\\CatalinaGroup\\Citrio\\User Data",
"Citrio",
"\\Fenrir Inc\\Sleipnir5\\setting\\modules\\ChromiumViewer",
"ChromiumViewer",
```

```
"\\uCozMedia\\Uran\\User Data",
"uCozMedia",
"\\Epic Privacy Browser\\User Data",
"Epic Privacy Browser",
"\\Elements Browser\\User Data",
"Elements Browser",
"\\Kometa\\User Data",
"Kometa",
"\\Vivaldi\\User Data",
"Vivaldi",
"\\Chedot\\User Data",
"Chedot",
"\\CentBrowser\\User Data",
"CentBrowser",
"\\7Star\\7Star\\User Data",
"7Star",
"\\MapleStudio\\ChromePlus\\User Data",
"ChromePlus",
"\\Iridium\\User Data",
"Iridium",
"\\Amigo\\User\\User Data",
"Amigo",
"\\Steam",
"Steam",
"\\NVIDIA Corporation\\NVIDIA GeForce Experience",
"NVIDIA",
"\\Yandex\\YandexBrowser\\User Data",
"Yandex",
"\\Google(x86)\\Chrome\\User Data",
"Chrome (x86)",
"\\Chromium\\User Data",
"Chromium",
"\\Battle.net",
"Battle.net",
"\\CryptoTab Browser\\User Data",
"CryptoTab",
"\\BraveSoftware\\Brave-Browser\\User Data",
"Brave",
"\\Microsoft\\Edge\\User Data",
"Edge",
"\\Google\\Chrome\\User Data",
"Chrome",
"\\Opera Software",
"Opera",
"\\discorddevelopment",
"DiscordDevelopment",
"\\discordptb",
"DiscordPTB",
"\\discordcanary",
"DiscordCanary",
"\\Discord",
```

```
"Discord",
"(.*)",
"(. -)",
"*",
"%RECENT%",
"%LOCALAPPDATA%",
"%APPDATA%",
"%USERPROFILE%",
"%DOCUMENTS%",
"%DESKTOP%",
",",
":",
"rule_collect_recurv",
"rule_size_kb",
"rule_folder",
"rule_files",
"rule_exceptions",
"open",
"https://",
"vbs",
"bat",
"bmp",
"jpeg",
"jpg",
"png",
"docx",
"doc",
"txt",
"scr",
"msi",
"exe",
".exe",
".",
"mark_domains",
"mark_check_history",
"mark_check_passwords",
"mark_check_cookies",
",",
" ",
"ld_url",
"ld_geo",
"ld_marks",
"ld_name",
"s",
"\\Telegram",
"\\maps",
"\\key_datas",
"\\tdata",
"\\Telegram Desktop",
"VaultGetItem",
"VaultCloseVault",
```



```

"VaultOpenVault",
"VaultFree",
"VaultEnumerateVaults",
"VaultEnumerateItems",
"vaultcli.dll",
"SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies",
"\\cookies.sqlite",
"file:/// ",
")",
"SELECT url FROM moz_places WHERE (`id` = ",
"SELECT place_id, content FROM moz_annos WHERE (`anno_attribute_id` = 1)",
"\\places.sqlite",
"SELECT fieldname, value FROM moz_formhistory",
"\\formhistory.sqlite",
"/ ",
"encryptedPassword",
"encryptedUsername",
"formSubmitURL",
"SELECT encryptedUsername, encryptedPassword, formSubmitURL FROM moz_logins",
"\\signons.sqlite",
"\\logins.json",
"Path",
"Profile",
"\\profiles.ini",
"PK11SDR_Decrypt",
"PK11_Authenticate",
"PK11_FreeSlot",
"PK11_GetInternalKeySlot",
"NSS_Shutdown",
"NSS_Init",
".ldb",
".log",
"Local Storage\\leveldb",
"\\Local State",
"dQw4w9WgXcQ:",
"dQw4w9WgXcQ:[^.*\\['(.*'\\\\].*$][^\\\\]*",
"email",
"username",
"discord.com/api/v9/users/@me",
"_0.indexeddb.leveldb",
"_0.indexeddb.leveldb\\CURRENT",
"\\IndexedDB\\chrome-extension_",
"\\Local Extension Settings\\",
"\\CURRENT",
"\\Sync Extension Settings\\",
"time",
"history",
"expirationDate",
"secure",
"1",
"httpOnly",

```

```

"domain",
"Network\\",
"\\Network",
"cookies",
"Cookies",
"network",
"status",
"card_art_url",
"bank_name",
"last_four",
"exp_year",
"exp_month",
"nickname",
"billing_address_id",
"origin",
"card_number",
"expiration_year",
"expiration_month",
"name_on_card",
"cards",
"path",
"SELECT tab_url, target_path FROM downloads",
"download_history",
"History",
"value",
"name",
"SELECT name, value FROM autofill",
"autofill",
"Web Data",
"password",
"login",
"url",
"profile",
"SELECT action_url, origin_url, username_value, password_value FROM logins",
"encrypted_key",
"os_crypt",
"logins",
"Local State",
>Login Data",
"\\*",
"v11",
"v10",
"3",
"90",
"An uncaught exception occurred3. The type was unknown so no information was
available.",
"An uncaught exception occurred3: ",
"ZZ",
"1.1.1.1",
"An uncaught exception occurred_ip4. The type was unknown so no information was
available.",
"An uncaught exception occurred_ip4: ",

```

```

"en",
"names",
"iso_code",
"www.maxmind.com/geoip/v2.1/city/me",
"An uncaught exception occurred_ip2. The type was unknown so no information was
available.",
"An uncaught exception occurred_ip2: ",
"countryCode",
"demoInfo",
"db-ip.com/demo/home.php?s=",
"An uncaught exception occurred_ip1. The type was unknown so no information was
available.",
"An uncaught exception occurred_ip1: ",
"city",
"country",
"data",
"ipinfo.io/widget/demo/",
"An uncaught exception occurred_ip0_2. The type was unknown so no information was
available.",
"An uncaught exception occurred_ip0_2: ",
"api64.ipify.org/?format=json",
"An uncaught exception occurred_ip0_1. The type was unknown so no information was
available.",
"An uncaught exception occurred_ip0_1: ",
"cc",
"ip",
"api.myip.com",
"softokn3.dll",
"nss3.dll",
"mozglue.dll",
"freebl3.dll",
"vcruntime140.dll",
"msvcpr140.dll",
"DLL",
"://",
"mark_name",
"Ledger Live",
"Daedalus Mainnet",
"Reddcoin",
"Litecoin",
"digitalcoin",
"devcoin",
"Zcash",
"YACoin",
"Terracoin",
"Primecoin",
"Namecoin",
"Mincoin",
"Megacoin",
"Ixcoin",
"Infinitecoin",
"IOCoin",

```

```

"GoldCoin (GLD)",
"Freicoin",
"Franko",
"Florincoin",
"DashCore",
"BBQCoin",
"Anoncoin",
"Dogecoin",
"Bitcoin",
"\\multidoge.wallet",
"\\MultiDoge",
"\\MultiDoge\\multidoge.wallet",
"\\Guarda",
"\\Ethereum",
"\\Ethereum\\wallets",
"\\app-store.json",
"\\Binance",
"\\Binance\\app-store.json",
"\\ElectronCash",
"\\ElectronCash\\wallets",
"\\bither.db",
"\\Bither",
"\\Bither\\bither.db",
"\\Wasabi",
"\\WalletWasabi\\Client\\Wallets",
"\\Armory",
"\\Coinomi",
"\\Coinomi\\Coinomi\\wallets",
"\\Jaxx",
"\\Jaxx\\Local Storage",
"\\Jaxx Liberty",
"\\com.liberty.jaxx",
"\\Monero",
"\\Monero\\wallets",
"\\ElectrumLTC",
"\\Electrum-LTC\\wallets",
"\\Exodus",
"\\Exodus\\exodus.wallet",
"\\Electrum",
"\\Electrum\\wallets",
"\\Atomic",
"\\atomic\\Local Storage",
"\\wallets",
"\\wallet.dat",
"\\",
"\\Session Storage",
"\\Local Storage",
"\\IndexedDB",
"\\Authy",
"\\Authy Desktop",
"\\*.*",

```

```

"%s [%s]\n",
"DisplayVersion",
"DisplayName",
"%s\\%s",
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall",
"\n[Software]\n",
"%s [%d]\n",
"\n[Processes]\n",
"VideoCard #d: %s\n",
"RAM: %u MB\n",
"CPU Count: %d\n",
"Processor: %s\n",
"ProcessorNameString",
"HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0",
"\n[Hardware]\n",
"TimeZone: UTC%d\n",
"\nLocal Time: %d/%d/%d %d:%d:%d\n",
" / %s",
"Keyboard Languages: ",
"Display Language: Unknown\n",
"Display Language: %ws\n",
"Display Resolution: %dx%d\n",
"User Name: %s\n",
"Computer Name: %s\n",
"x32",
"x64",
"Windows: %s [%s]\n",
"ProductName",
"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
"Location: %s, %s\n",
"IP: %s\n",
"Work Dir: %s\n\n",
"\nPath: %s\n",
"HWID: %s\n",
"_",
"GUID: %s\n",
"MachineID: %s\n",
"MachineGuid",
"SOFTWARE\\Microsoft\\Cryptography",
"Unknown",
"Date: %s",
"Version: %s\n\n",
"wb",
"\\information.txt",
"%X",
"RtlGetVersion",
"Ntdll.dll",
"rb",
"RisePro\r\nTelegram: https://t.me/RiseProSUPPORT",
"50500",
"0.1",

```

```

    "?",
    "/",
    "http://",
    "WinHttpSetTimeouts",
    "WinHttpCloseHandle",
    "WinHttpReadData",
    "WinHttpReceiveResponse",
    "WinHttpSendRequest",
    "WinHttpQueryDataAvailable",
    "WinHttpOpenRequest",
    "WinHttpOpen",
    "WinHttpQueryHeaders",
    "WinHttpConnect",
    "WINHTTP.dll",
    "#",
    ".dll",
    ".",
    "InternetCloseHandle",
    "InternetReadFile",
    "HttpSendRequestA",
    "InternetQueryOptionA",
    "HttpQueryInfoA",
    "InternetOpenUrlA",
    "InternetConnectA",
    "HttpOpenRequestA",
    "InternetSetOptionA",
    "InternetOpenA",
    "wb",
    "HEAD"
]
}

```