SZAKDOLGOZAT

OE-NIK
2021

Bőhm Emese
T/006150/FI12904/N

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Bőhm Emese**
2021 Hallgató törzskönyvi száma: **T/006150/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT

FELADATLAP

Hallgató neve: **Bőhm Emese**
Törzskönyvi száma: T/006150/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

VM detektálási technikák elkerülése

Avoidance techniques for VM detection

Intézményi konzulens: Rigó Ernő
Külső konzulens: Lendvai Péter

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Felhőszolgáltatási technológiák és IT biztonság specializáció

A feladat

A virtuális gépi környezet, vagy másnevén a VME (Virtual Machine Environment) könnyebb menedzsmentet és nagyobb elszigetelést kínál, mint a hagyományos gépek. Ezért a kutatók és automatizált elemző szolgáltatások gyakran virtuális környezetben végzik a kártékony kódok elemzését. Erre válaszként a támadók, olyan rosszindulatú kódokat hoznak létre, melyek VME detektálási technikák segítségével igyekeznek megakadályozni a sikeres elemzést.
A szakdolgozat célja a virtuális gépi környezetek infrastruktúrájának, működésének feltárása, főként az informatikai biztonság, szűkebben pedig a VME detektálási technikák elkerülése szempontjából. A munka célja egy olyan virtuális futtató környezet létrehozása, mely megnehezíti ezen technikák sikeres alkalmazását a kártékony kódok számára.

A dolgozatnak tartalmaznia kell:

- a korszerű virtualizációs technológiák általános jellemzőit feltáró irodalomkutatás eredményét,
- VME detektálás elkerülési módszereit, illetve kiküszöbölési lehetőségeit feltáró irodalomkutatás eredményeit,
- a megvalósítani kívánt virtuális környezettel szemben támasztott elméleti és gyakorlati elvárásokat, a technológia kiválasztásának szempontrendszerét,
- az elvégzendő detektálhatósági tesztek terveit és elvárt eredményeit,
- a megoldás megvalósításának bemutatását, a tesztelés eredményeit és ezek értékelését,
- a munka eredményeként levont tanulságokat, és esetleges továbbfejlesztési lehetőségeket.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest Ph.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2021.12.08.

Bódis Emese Zsuzsanna
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Bőhm Emese Zsuzsanna .. [redacted] 3

Telefon: Levelezési cím (pl: lakcím):

[redacted] ..... [redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

VM detektálási technikák elkerülése .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Avoidance techniques for VM detection .....

Intézményi konzulens: Külső konzulens:

Rigó Ernő..... Lendvai Péter .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	**2021.04.06**	**Virtualizáció alapjai**	
2.	**2021.04.20**	**Virtualizáció kihívásai**	
3.	**2021.04.27**	**VM detektálási technikák**	
4.	**2021.05.04**	**VM detektálási módszerek elkerülése**	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.05.13.....

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Bőhm Emese Zsuzsanna ... [redacted] 4

Telefon: Levelezési cím (pl: lakcím):

[redacted] ..... [redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

VM detektálási technikák elkerülése .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Avoidance techniques for VM detection .....

Intézményi konzulens: Külső konzulens:

Rigó Ernő ..... Lendvai Péter .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.21	Szakdolgozat átbeszélése	
2.	2021.10.12	Tesztkörnyezet bemutatása	
3.	2021.11.30	Prezentáció átbeszélése	
4.	2021.12.07	Szakdolgozat finomítása	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.12.07.....

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalom

1 Előszó

Napjainkban a különböző technológiák mellett, a kártevők fejlődése is egyre jobban felgyorsult. Ennek hatására a különböző rosszindulatú program elleni eszközöket gyártó cégeknek hasonló ütemmel kell produkálni az újnál-újabb termékeket.

A megfelelő kártevők elleni megoldásra az IT biztonság szakterületén dolgozó kutatók a különböző rosszindulatú programok tulajdonságát, viselkedési formáját figyelik meg, illetve elemzik. A virtuális környezet megfelelő egy ilyen jellegű kutatáshoz, mert a virtuális gépek lehetővé teszik, hogy egy hagyományos fizikai gép általános látszatát keltse, illetve a vizsgálat folyamán jobb szigetelést tud nyújtani, mint egy fizikai gép. Így egy egyszerű, átlagos kártevő vizsgálatakor hasonló viselkedés fog lezajlani, mint egy átlagos fizikai gépen. Annyi különbséggel, hogy a virtuális gépet vissza lehet állítani régi állapotában, illetve kutatást alatt sokkal olcsóbb megoldást fog nyújtani, mint egy hagyományos számítógép.

Ezt tudván a támadók olyan kártékonykódú programokat is készítenek, amely képesek észlelni a virtuális környezetet, melyet különböző virtuális gép (vm) detektáló módszerekkel sajátítanak el. Ha egy ilyen típusú program észlel egy virtuális környezetet, akkor jó indulatú programként fog viselkedni, vagy rosszabb esetben át tud szivárogni a fizikai gépre.

Általában a virtuális gépeknek nem feltétlenül látható tipikus tulajdonságaik, jellegzetességeik vannak. Ezt a sebezhetőséget kihasználva alkalmazzák a különböző vm detektálási technikákat. Ilyen tulajdonság a helyi virtuális környezet detektálási módszereknél jellemző a különböző virtuális környezetre utaló hivatkozások, folyamatok (pl. VMtools service), az alapértelmezett eszköz nevek (pl. VirtualBox VESA BIOS), VME-re utaló speciális virtuális hardver, amely pl. a VMware hálózati kártyája, amely alapértelmezett MAC-címmel van ellátva stb.

A szakdolgozatom célja a különböző virtualizációt észlelő kártevők elleni alap technikák ismertetése és megvalósítása. melyek segítségével virtuális gépként eltérő megoldásokkal lehet orvosolni az ilyen jellegű problémákat. A szakdolgozatomban lokális VME detektálási, illetve anti-vm detektálási módszereket fogom bemutatni, illetve leírni az azzal kapcsolatos eredményeket, következtetéseket.

2 Bevezetés

Manapság a virtualizáció mindenhol jelen van az életünkben, amely szerepe egyre jobban növekszik a különböző technológiák által. A virtualizáció és annak különböző megoldásai, megvalósításai érdekes témának tartom, de a szakdolgozatom témája miatt csak a virtuális gépekről lesz szó.

2.1 Virtualizáció alapjai

A virtuális gépek 1960-as évekig nyúlnak vissza, melyet először az IBM Corporation fejlesztett ki. Az akkoriban még drágább hardverek, illetve erőforrások költségeinek csökkentése miatt valósították meg. Lényegében egy fizikai nagyszámítógépet petícionáltak több logikai példányra, melyek azon az egyetlen hardveren futottak. [1]

A virtualizációs technológia fejlődése lassult az egyre inkább olcsóbbá vált hardverek, illetve a többprocesszoros gépeknek köszönhetően.

Majd a különböző PC-alapú hardverek és a változatos sokféle operációs rendszerek 1990-es évek óta újra kezdték élni virágkorukat egészen napjainkig. A piacon megjelent különböző hardverek, illetve operációs rendszerek miatt megoldást nyújtott a virtuális gép.

A virtualizáció lehetővé teszi, hogy egy hagyományos számítógépen, olyan absztrakciós réteg jöjjön létre a fizikai elemek felett, amely által a fizikai erőforrások felosztásával akár több virtuális gép futtatható. [1]

A virtuális gép egy fizikai számítógép általi szoftver segítségével tud implementálódni, ami ennek a gépnek a dedikált mennyiségű processzorát, tárhelyét és memóriáját ideiglenesen fogja használni, amíg a virtuális gép fut.

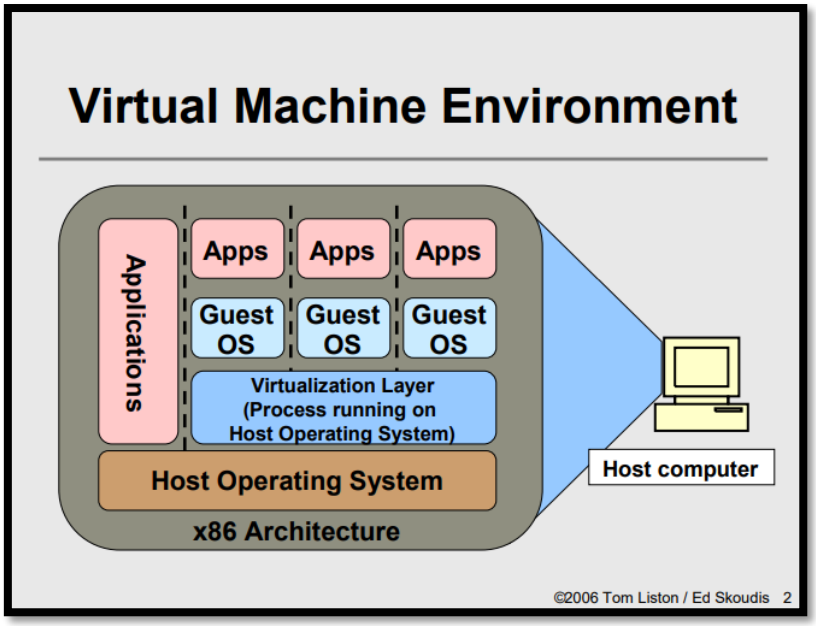
A szoftver, mely segítségével a fizikai és a virtuális gépet megfelelően lehet elszigetelni hypervisor-nak nevezzük. Ennek köszönhetően egyszerre több operációs rendszer is futtatható, illetve lehetővé teszi, hogy egy különböző operációs rendszerű fizikai gépen futhasson egy másik operációs rendszer.

Egy általános virtuális gép szerkezetét tekintve, a fizikai számítógép, és rajta elhelyezkedő hypervisor réteg felett található a vezérlő program által futó vendég gépek, amelyek függetlenek a számítógépen futó folyamataitól. A vendég gépeket lényegében virtuális gépeknek szoktunk nevezni, amelyeken különböző operációs rendszerek, alkalmazások futhatnak.

A virtualizáció két kiemelkedő előnye a forrásmegosztás, melynél a hypervisor által kiosztott erőforrások a virtuális gépekre osztódnak fel a gazdagép erőforrásaiból. A másik előnye az elszigeteltség, melynél a virtuális és a fizikai réteg elkülönül a hypervisor segítségével. A technológiák előnye a sérülékenység is, mivel a felhasználók biztosak az előnyök stabil funkcionalitásában, ezért egyfajta álbiztonságot fog éreztetni a virtuális környezetben. A további előnyei, hogy egy esetleges katasztrófa esetén könnyeben helyre lehet állítani az adott infrastruktúrát. Illetve gazdasági és energia szempontból is elmondható, hogy sokkal költséghatékonyabb, rugalmasabb, dinamikusabb, mint egy hagyományos hardver alapú megoldás. [1]

2.2 Virtualizáció hátránya

A fenti előnyök alapján nem véletlenül terjedt el a virtualizáció. A népszerűségének van hátránya is, mint például jó célpontnak számít támadók, malware írók számára.
A virtuális gép legnagyobb hátrányának tartom, hogy egy álbiztonságot ad a felhasználójának. Mondhatni rá vagyunk utalva a gyártók által már legyártott virtuális gépek infrastruktúrájára.
A gyártók termékeit folyamatosan fejlesztik tovább, ami miatt újabbnál újabb sérülékenységek előfordulhatnak. Természetes a gyártók ezzel párhuzamosan patchelnek, hogy naprakész, biztonságos terméket adjanak fogyasztóiknak.



Ábra 1 [1]: Virtuális környezet

Például a VMware virtuális gépnél VMware Tools kapcsán 2020-ban a „CVE-2020-3941” sérülékenységre bukkantak, melynél jogosultságeszkalációs probléma volt. Lényegében a VMware Tools 11 verziójában jogosultságok kiterjesztés nem volt megtalálható. Ami miatt a támadó a megszerzett jogosultságok által hozzáférhet bármilyen adathoz, akár felhasználó fiókokat törölhet, létrehozhat és még további kárt tud ezzel okozni. [2]Vagy hasonló jogosultsággal kapcsolatos sérülékenységre bukkantak 2021-ben, ami a „CVE-2021-21989” számú azonosítóval rendelkezik. Ez a sérülékenység a VMware Workstation 16.x és a 16.1.2 és az előtti verzióknál fordult elő. A sebezhetőség abból állt, hogy a virtuális géphez vagy a távoli hozzáféréssel rendelkező támadó kihasználhatja a Cortado ThinPrint komponensben található korlátokon kívüli olvasást, amelynek köszönhetően információkat tud megszerezni. [3] A „CVE-2019-16406” számú sérülékenységnél a VMware virtuális gép fájlja az OVA, illetve a VirtualBox virtuális gép fájlja az OVF szintén gyenge jogosultságúak voltak, ezáltal a egy trójai faló Centreon-autodisco futtaható fájlon keresztül a támadó megtudta szerezni a jogosultságokat. Ezt a sérülékenységet 2019-ben fedezték fel. [4]
A legtöbb sérülékenységnél a virtuális infrastruktúra összetevőinél találhatóak meg, ami hatással van vendég és a gazdagépre is egyaránt. [5]

A szakdolgozatban végzett irodalmi kutatás során fontosnak tartom megemlíteni a ZTA-t (Zero Trust Architecture), amit 2018-ban vezetett be a NIST és az NCCoE.A kiadvány lényege a zéró bizalmat szolgáljunk architektúra tervezésekor és megvalósításakor, mely magába foglalja az összes olyan elemek, összetevők közötti kapcsolatokat, folyamatokat, ami alapján egy biztonságosabb munkafolyamatot és hozzáférést eredményez kialakítása után. A legfontosabb elvek a felhasználói identitás egyetlen erős forrása, a gépi hitelesítés, a felhasználói hitelesítés, kiegészítő kontextus, azaz a házirendek megfelelő meghatározása. A további fontos része az alkalmazáshoz való hozzáférés engedélyezési szabályai és a hozzáférés-szabályozás az alkalmazáson belüli lekötések pontjainak leírása. A virtuális gépek infrastruktúráján is érdemes a bizalmatlansági koncepciót felmérni. Például az említett hypervisor-os réteg teljes izoláltságának funkcióját nem szabad biztosra venni. A ZTA tervezési elmélete alapján jobban ellenőrizhető a virtuális infrastruktúra, illetve fenntartásokkal kezelhető. [6] Például fontos szempont, hogy a rendszernek biztosítania kell, hogy megfelelő engedélyezést adjon az alany számár, illetve érvényes legyen a hitelesítés az erőforráshoz. A nulla bizalom alapelvével kell eljárni, ami a hitelesítés és az engedélyezésen alapszik. Az alany személyazonosság ellenőrzése szükséges, ahhoz, hogy hozzáférjen a kért erőforráshoz. Ezért a vállalatoknak olyan dinamikus és kockázatalapú irányelveket kell felépíteniük és fenntartaniuk, ahol az erőforrás-hozzáférést kell megfelelően kezelni. Azaz a házirendek megfelelő

konfigurációja szükséges. Ezek alapján nem szabad megbízhatóságra hagyatkozni egy jogosultsági hozzáféréskor, hanem különböző hitelesítési szinteket meg kell lépni az alanynak, hogy hozzáférhessen az adott erőforráshoz. A virtuális gépen futó alkalmazásokat is jóvá kell hagyni. Az PEP (Policy Enforcement Point), azaz a szabályzati végrehajtási ponttal kell kommunikálhatnak az alkalmazások a kért erőforrások hozzáférése érdekében. Egy nem kívánt alkalmazás kérelmét el is tudja utasítani a PEP. A szabályzati végrehajtási pont általában valamilyen vállalti helyi szolgáltatás, vagy felhőszolgáltatás szokott lenni. Az említett architektúrai védelmi megoldás előnye, hogy az alkalmazások többnyire elvannak szeparálva egymástól. Érdemes minden virtuális gép biztonságos működésről meggyőződni. [7] A legjobb példa a VMware NSX-en alapuló telepítés, ami egy ZTA fejlesztés. A virtuális asztali infrastruktúrát, azaz a VDI.25-t alkalmazza. Az alkalmazások virtuális szerveken vannak futnak. Az szerkezet először a felhasználó hitelesítésével kezdődik a VDI kiszolgálón. Majd ezután a felhasználó számára egy távoli munkamenet jön létre a virtuális gépen. Az NSX-alapó megközelítés két eleme a kiszolgáló és a virtuális asztal. Így az NSX tűzfalként fog funkciónál, ahol a házirenddel kapcsolatos feladatokat kell ellátnia. A megvalósításnak köszönhetően a rendszergazdák a hálózat egész területén, több ponton is végezhetnek hozzáférés-szabályozási finomhangolást, amit a szakirodalom mikró szegmentálásnak nevez. A megoldás előnye, hogy egy vállalat számára a rendszergazda jogosultsággal rendelkező személy az összes virtuális gépet vagy virtuális asztali gépet tudja kezelni, karbantartani vagy frissíteni. A lekorlátozott jogosultság miatt, a vállalat jobban védve van a különböző kibertámadások ellen. A NIST ezt a modellt eszközügynök/átjáró alapúként kategorizálja. [8]

A virtualizáció hátránya kapcsán a hypervisor nem biztos védelmére érdemes említést tennem az esetleges védelmi megoldással kapcsolatban. A virtualizációs szoftverekben, mint például a VirtualBox-ban vagy a VMware-ben felfedezhető hibák, sérülékenységek kihasználása alapján az mutatkozik meg, hogy a véghez vitt sikeres támadást a hypervisor-ok nem megfelelő védelme könnyítette meg. A hypervisor a virtualizáció fő eleme, amely által az összes erőforráshoz hozzáférhet, a fizikai platform kezeléséért is felelős. A biztonságos virtualizáció érdekében érdemes megemlíteni egy kifejezetten hypervisor védelmi architektúrát, ami a HyperWall architektúra. Ez az architektúra a biztosítja a hypervisor biztonságos memória és egyéb erőforrásainak kezelését, viszont, ha egy virtuális gép létrejön, akkor az új CIP, azaz Confidentiality and Integrity Protection táblák védik a vendéggép memóriáját a hypervisor vagy a DMA hozzáféréstől. A CIP táblák a virtuális gép adatait és kódjait titkosítással védik, illetve ezek a táblák csak hardver számára elérhetőek. Így a CIP táblájának logikája megvédi a virtuális gép memóriáját az áldozatul esett hypervisor-tól.

Ha egy vezérlő támadása sikeres lenne, akkor ez a módszer megvédi az esetleges többi gép hozzáférésétől, így csak, azaz egy virtuális gép esik áldozatul. A mikroprocesszor és a memóriakezelő egységek átalakításának köszönhetően nagyobb védelmet nyújt a nem kívánt betolakodók ellen. Ezt az architektúrát leginkább IaaS felhőknél szokták alkalmazni. Az olvasott kutatásnál, ami a HyperWall architektúrát vizsgálja meg, olyan következtetést vontak le a kutatók, hogy igenis jól védi a nem kívánt betolakodókat ez a megoldás, de leginkább felhőalapú környezetben a leghasznosabb, mivel a vállalatok számára a különböző virtuális gépek adatait megfelelően tudja titkosítani, védeni és integrálni. Illetve előnyének mondható, hogy jó egyensúlyt megteremteni a teljesítmény és a biztonság között. [9]

A hypervisor védelme nem fog beleférni a beszámolóm témájában, viszont mindenképpen megakartam említeni néhány megoldást. A szakdolgozatomban a helyi anti-vm detektálási technikákkal fogom foglalkozni, melynél a virtuális gépek sérülékenységeit megpróbálom elkerülni.

3 VME detektálásának jelensége, módszerei

A virtualizációnak köszönhetően egyszerre több operációs rendszert tudunk futtatni, illetve a pillanatképek által van rá lehetőség visszaállítani a korábbi állapotára a virtuális gépeket, illetve az egész rendszer többnyire jól ellenőrizhető. Nem véletlenül alkalmazzák a legtöbb iparágban ezt a technológiát. Az informatikai biztonsági területen dolgozó kutatók számára a virtuális gépi környezet biztosítja a kártékony kód programoknak futtatását, illetve elemzését. Mivel a virtuális környezetben lévő rosszindulatú program jobban is elemezhető, mint egy hagyományos fizikai számítógépes környezetben. Tesztelési és elemzési tevékenységekre ideális környezetet biztosít. A különböző operációs rendszerekben pontosabb elemzési eredményt lehet kapni az összehasonlítások által. A virtuális környezetben lévő rosszindulatú program analizálás jóval olcsóbb és rugalmasabb tud lenni, mint egy fizikai gépeké. [1] [10]

A kutatók jelentős eredményeket tudnak elérni a folyamatosan fejlődő rosszindulatú programok viselkedésének megfejtésével, ami által ezeket a kártékony programokat megfelelően tudják kezelni, hatástalanítani. Ennek tudatában a támadok, olyan kártevőket hoznak létre, amely képes a virtuális környezet észlelésre. Általában két viselkedési típusa lehet az ilyen rosszindulatú programoknak. Az egyik, hogy ha sikeresen érzékelte a kártékony program a virtualizációt, akkor jóindulatú programként működik, vagy megállítja a saját

futtatását. A másik esetnél a program a detektálás után virtuális gépből való menekülést fog végrehajtani, és megfertőzi a fizikai gépet. Mindegyik esetnél a támadó célja, hogy a kutatók elemzését lassítsák vagy szabotálják. [1] [10]

A virtualizáció detektálást két fő részre lehet osztani helyi, illetve távoli virtuális környezet észlelésre. A helyi észlelés célja a gazdagépen lévő virtuális gépnek az érzékelése. Manapság egyre több ilyen jellegű módszer létezik. De az alapvető kategóriák az esetleges virtuális környezetnek a leleteinek keresése a processzorokban, fájlrendszerekben, memóriában vagy a szintén virtualizációra jellemző speciális processzor utasítások, képességek vagy speciális virtuális hardver megkeresése különböző technikákkal. [1] [10]

3.1 Nyilvántartások, folyamatok, fájlrendszerek megvizsgálása

Az első nagyobb kategória, az említett VME-re utaló tulajdonságok felderítése a különböző nyilvántartásban, folyamatokban vagy fájlrendszerben.

A virtuális gép létrehozásának következtében a vendég gépen lévő operációs rendszerben különféle bejegyzések lesznek találhatóak, melyek a használt virtualizációt támogató termékkel kapcsolatos bejegyzések lesznek, mint például egy VMware használatánál ezekkel a bejegyzésekkel találkozhatunk: [10]

„HKEY_LOCAL_MACHINESYSTEMControlSet001ControlClass

{4D36E968-E325-11CE-BFC1-08002BE10318}000DriverDesc”,
„HKEY_LOCAL_MACHINESYSTEMControlSet001ControlClass

{4D36E968-E325-11CE-BFC1-08002BE10318}000ProviderNameVMware,Inc.”

Azonfelül a virtuális gép futásakor a háttérben különböző virtualizációt segítő folyamatok, fájlok vannak jelen, mint például a VMware használatakor a VMwareService.exe és a további virtualizációt segítő eszközei. [10] [11]

Ezt a technikát például a Phatbot-nak nevezett malware is alkalmazza. A Phatbot alapvető működése a számítógépen lévő úgynevezett hátsó ajtó megkeresése. Ennek mentén jól elképzelhető, hogy a rosszindulatú program hasonló elven fogja megkeresni a virtualizációban lévő kis rést. Például, VMware Workstation WinXP vendég esetén a „VMtools” szolgáltatás futtatása, a fájlrendszerekben található több mint 50 különböző hivatkozások „.vmx” kiterjesztésű fájlokra, mely által jól érzékelhető lehet a virtuális környezet megléte. A támadók

lényegében a virtuális gépek jellegzetes folyamatainak és szolgáltatásainak a futtatását, a tipikus fájlok és könyvtárak, illetve a speciális rendszerleíró kulcsoknak alap tudásával használják ki a rendszert. [1] [10]

3.2 Memória feltérképezése

A gazdagépen található memória tartomány különbözik a virtuális gép memória tartományától. Ennek a tudásának a birtokában a támadó a fizikai memórián keresztül a vendég géppel kapcsolatos hivatkozásokat meg tudja találni. Az olvasott egyik forrásból, a teszt során egy WindowsXP VMware vendég gépeket alkalmaztak, ami során kidobta a memóriát az éppen futó virtuális géptől, mely által közel 1500 hivatkozást találtak a memóriában, ami VMware-re jellemző hivatkozásokat tartalmazott. A másik tipikus megközelítése ennek a kategóriának az operációs rendszerek táblázat megismerésének elsajátítás, mely magába foglalja a mutatók megtekintését is. Ilyen jellegzetes tábla az IDT-ként (Interrupt Descriptor Table) ismert adatstruktúra, mely olyan x86-os architektúra megszakítási vektortábla, amit a processzor használ megszakításokra vagy kivételekre. [1] [12]

Így az IDT segítségével megtudható, hogy pontosan hol helyezkedik el az operációs rendszer megszakításkezelői a memóriában. Amellett az IDT a fizikai gépen lévő memóriában alacsonyabban található, mint egy virtuális gép memóriájában. [1] [10] [12]

E hasonló technikát 2004-ben publikálta Joanna Rutkowska, mely a „Red Pill” -ként ismert virtuális környezetet detektáló eszközt tervezte és valósította meg. Az szoftver magyarra lefordítva „Vörös Tabletta”, aminek az értelme, hogy egy egyetlen gépi utasítást alkalmaz az IDT tartalmának ellenőrzéséhez. Az használt utasítás az SIDT (Store Interrupt Descriptor Table), ami tárolja a fent említett IDT tartalmat a processzor egyik regiszterében, azaz az IDTR-ben (Interrupt Descriptor Table Register). Általában az IDTR regisztert a hossz bájtokban történő tárolásra és a fizikai alap cím alkalmazására használják. [1] [10]

Rutkowska megfigyelte kutatásai során, hogy a különböző vendégépek általában a vendég gép típusához egy adott címértéket használnak, mint például VMware-nél a 0xffXXXXXX cím. Azonfelül az IDT a gazdagépen található operációs rendszernél alacsonyabban helyezkedik el a memóriában. A vizsgálat során a konklúzió az lett a VMware esetében, hogy ha az IDTR nagyobb címértékű mint az „0xd0000000” érték, akkor megállapítható, hogy a vizsgált címérték virtuális géptől származik. [1] [13]

A másik hasonló program a Scoopy, ami hasonlóképpen jár annyi különbséggel, hogy az IDT mellett a GDT (Global Descriptor Table) az SGDT (Store Global Descriptor Table)

utasítással együtt, illetve a LDT-t (Local Descriptor Table) SLDT (Store Local Descriptor Table) utasítással valósítja meg. A másnevén ismert Globális Leíró Táblázat az Intel x86-os architektúrájának olyan adatstruktúrája, ami különböző memóriaterületeket használ a végrehajtás során, melynek célja az ezzel kapcsolatos jellemzők meghatározása. Például tartalmaz alapcímet, hozzáférési jogosultságot, illetve méretet. Az ilyen jellegű memóriaterületeket szegmenseknek nevezzünk. Az SGDT utasítás csak operációs rendszerben lehet használni, amely tárolja a GDT regiszter tartalmát. Az LDT egy szintén x86-os architektúrához tartozó memória táblázat, mely memória szegmens leírókat foglal magában. A GDT-hez hasonló, de abban különbözik, hogy csak bizonyos rendszer bejegyzéseket tud tárolni. Az LDT SGDT utasítást használ, mely utasítás csak védett módban hajtható végre. Ez a végrehajtás a cél leíró operandus LDT-ből tárolja szegmensválasztót. Összesítve a Scoopy hasonló technikát alkalmaz, mint a Vörös Tabletta, annyi különbséggel, hogy más táblázatokat is használ. [1] [11] [12]

3.3 Virtuális hardverek észlelése

A harmadik kategória a virtuális környezetben lévő speciális virtuális hardverek keresése. Ennek a technikának az alapelve, hogy olyan jellegzetes ismertetőjel után kutassunk, mint például a hálózati kártyákon lévő MAC címek stb. Például, a VMware-nél a MAC cím általában 00-05-69, 00-0c-29, 00-1c-14 vagy 00-50-56 kezdetűek szoktak lenni, míg a VirtualBox-nál 08:00:27 kezdetűek. [11] [14] [15]

Felderítéskor általában a kártevő BIOS sorozatszámát is tartalmazza, mely azért fontos, mert például a VMware-nél a létrejött virtuális gépekhez „VMware” karakterláncot csatolnak a BIOS sorozatszámukhoz. [15]

Egyéb hardverellenőrzéseknél a kártevők különféle tulajdonságokat kérdeznek le, mint például a SocketDesignation, SerialNo a Caption. Ennél az esetnél a processzor, alaplap, illetve a SCSI vezérlő értékeinek a lekérdezése volt a cél. [11]

Ehhez szükséges felkutatására a Doo eszköz alkalmas lehet. Tobias Klein által már kiadott Scoopy eszköz után, a Doo is támogatja a virtuális környezet észlelését a virtualizált hardver felkutatása által. Egy vizsgálat során átkutatja a különböző könyvtárakat, a VMware -hez társított rendszerleíró kulcsokat böngészi, amik a VMware hardvernek az osztály azonosítóinak, illetve SCSI adaptereknek ellenőrzésével derítheti fel. [1]

A hardveres felderítéskor a malware az átalános tulajdonságokat is szemügyre veszi, mint például a megfelelő képernyőfelbontás, memória méret, merevlemez méret, illetve CPU szám. Azonfelül a virtuális hálózatot is detektálja. [16]

3.4 Kommunikációs csatorna megfigyelése

Az utolsó főbb kategória a virtuális környezetben lévő speciális processzor utasítások és az ehhez szükséges képességek feltárása a cél. A legszélesebb körben használt virtuális környezetben a leggyakoribb, a VMware-nél. A gazdagép és a vendég gép közötti kommunikáció egy egyedi kommunikációs csatornán zajlik, mely a kettő komponens működésért, funkciókért felel. A technika vizsgálata során kiderült, hogy egy rosszindulatú programmal fertőzött kliens válaszintézkedésénél egy kis kód szegmens ellenőrizte fent említett kommunikációs csatorna meglétét. [10] [11]

A talált kódot fájldobóként (file dropper) ismerte fel az eszköz. Ez egy futtatható fájl, ami kódolt adatként tárol egy másik programot, és azt az adatot dekódolja a végrehajtáskor. A kártékonykódú programoknál megtalálható ezeknek a rekurzív csavara.

A következő példán az „IN” utasítást fogják kihasználni a támadók. Az említett parancs által jön létre a vendéggép és gazdagép közötti kommunikációs csatorna. Általában az „IN” x86-os utasítás egy bájt vagy szó szokott lenni egy I/O porton, ami általában például fizikai számítógépen található meg. Az utasítás használatakor két paraméter szükséges, az egyik a regiszter szükséges rendeltetés helyének meghatározása, míg a másik egy számérték, ami megadja, hogy melyik portról elérhető. [10]

Az 2. ábrán jól látható, a pontos folyamat menete. Az EAX regiszteren megtörténik az „564D5868” értékű „VMXh”, ami szükséges a kommunikációs csatorna meghívásához. Majd ezt követően a „EBX” -t kinullázzuk a későbbi tárolás miatt. A következő sorban lévő utasítás, mely „0A” értékű, azaz decimális értéke a tíz azt mutatja meg, hogy pontosan melyik VMware verziónak az ellenőrzését szeretnénk elvégezni. Az „EDX” regiszterbe az „5658” -os értéket rakjuk, mely a „VX” -vel egyenértékű, ami VMware-hez társított speciális hardver portot definiálja. Ezután „IN” utasítást alkalmazzunk, ami alapjába véve az x86-os processzoroknál olvasásra szokták használni, de a VMware-nél a virtuális gépekhez szükséges kommunikációs csatorna megvalósításához, emiatt azt alkalmazzuk, így ellenőrizve lesz a VMware számára. [1] [10] [16]

Az utolsó lépésként az összehasonlítjuk az „EBX” regiszterben lévő értéket a „VMXh” -val, mert ezáltal megtudható, hogy fut-e a VMware vendég gépben a kódunk. Ha nem vendég gépként érzékeli, akkor ezeket az utasításokat kivételként fogja kezelni.

Egy „IN” gépi utasítás meghívásakor egy nem virtuális számítógép processzorkivételt dob, amely egy hiba üzenettel jár. A fent említett példában egy VMware virtuális környezetben az utasítások hiba nélkül végrehajtódnak, majd a végrehajtás által a processzorok értékei megváltoznak. A VME észlelés sikerét a varázsérték megfigyelése, és a kivétel dobás nélküli végrehajtás fogja okozni. [1] [10] [16]

```
MOV EAX,564D5868 <-- "VMXh"
MOV EBX,0
MOV ECX,0A
MOV EDX,5658 <-- "VX"
IN EAX,DX <-- Check for VMWare
CMP EBX,564D5868
```

Ábra 2 [1]: VMware kommunikációs csatorna

Összesítve a támadok a jól ismert gépi utasításokat használják ki, melyet egy általános processzor architektúránál jellemzően használunk. Az utasítás lényege, hogy használata által a szoftver különböző tulajdonságokat, részleteket ismer meg, amin az utasítást végrehajtották. A gépi kód utasítások az x86-os architektúráknál elengedhetetlenek, emiatt ez egy eléggé elterjedt vm detektálási technika. Természetes megemlített fő kategóriák általában vegyesen szokták alkalmazni, például a hypervisor-okkal kapcsolatos sérülékenységeknél. [1] [16]

3.5 Érzékeny utasítások vizsgálása

Fontos rátérni két alapfogalomra a virtuális gép kilépésről, mely során a virtuális gép futása után a vendég gép visszatérhet a hypervisor-hoz, azaz meghívhatja a kilépés kezelőjét. A másik fogalom a virtuális gép belépés, aminél a hypervisor által elvégzett művelet után vezérlés visszatérhet a vendéggéphez. A két jelenséget érzékeny utasításokkal, illetve a VT-x jelenségének kihasználásával lehet elérni az észlelést. A VT-x az Intel által gyártott virtualizációs technológia, mely a virtualizáció teljesítményét és átláthatóságát növeli. [10]

Ezek közül az egyik technika az x86-os utasítások végrehajtásának az idejének a mérése. Az eltelt processzor ciklus számának a mérése, mely az TSC -t felhasználva, illetve

utasításokkal elősegítve egy képlet segítségével meg lehet becsülni a ciklusok számát. Az időzítési tulajdonságok a natív végrehajtás során nem fordulhatnak elő, ennek következményeképpen a megállapítható a hypervisor jelensége. [10] [16]

A virtuális gép kilépési költségének észlelés alapja a virtuális gép kilépésekor használt processzor ciklusok. A virtuális gép kilépést elő lehet segíteni egyes CPU utasítások által, ami natív rendszerben nem lehetséges. Az Intel által készített kézikönyv tartalmazza ezeket a provokáló speciális utasításokat, ami körülbelül harminc utasítást tartalmaz. A vizsgálatnál egy előjog nélküli utasítást használtak, ami a „cpuid" parancs volt, melynél a hypervisor-os csapdákat nem lehet letiltani. A „cpuid" alap funkciója, hogy az x86-os processzorral kapcsolatos információkat lekérdezi. Az egész módszer lényege, hogy jelenesetben a „cpuid" utasítás idejét megmérjük, ami által definiálható egy küszöbérték. Ha küszöbérték eléri az adott értéket, akkor a VT-x jelensége észlelhető, amely következtethető virtualizációra. Mivel nem annyira pontos, ezért érdemes egy másik utasítással is elvégezni a becslést, melynek arányaival pontosítható a becslés. [10]

A másik detektálási technika a TLB (Translation Lookaside Buffer) szolgáltatásainak kihasználása lehet. A TLB egy gyorsítótárazó, ami virtuális és a fizikai memória közötti leképezését segíti elő, ami leváltotta az oldal sétát (page walk), mert költség hatékonyabb volt. A fordítási keresési puffer más nevén TLB a chip memória kezelő egységének a része. A fizikai memóriában tárolja a TLB a legutóbbi a két memória közötti fordításait. Ennek köszönhetően, ha megfelelően hozzáférünk a TLB-hez, akkor megtudható, hogy virtuális gép fut-e vagy sem. A TLB, és a virtuális gép kilépési költségeivel kapcsolatos módszereknél a nagy mennyiségű speciális utasítások feltűnőek lehetnek egy automatizált rosszindulatú program elemzésekor. Emiatt érdemes fontolóra venni, hogy egy fajta lopakodó virtuális kilépési detektálást kéne végrehajtani inkább, melyen elkerüljük, például a nagymennyiségű „cpuid" utasítást, illetve nem használunk gyanús, speciális utasításokat. Ha maximum három négy „cpuid" hívást végzik azt várhatóan jóindulatú viselkedésnek fogja érzékelni a rendszer. A lopakodó módszert a „cpuid" utasítást párhuzamos számítással fogjuk kiszámolni, mely minden egyes szál folyamat után növekedni fog egy bizonyos számláló. A műveletek után a számláló értéke alapján összehasonlítható az adott küszöbértékkel, ha kisebb, mint a küszöbérték akkor virtualizált környezet észlelhető. [10]

Az Intelnek az utasítási készlete az MMX, ami segíti a grafikus alkalmazások gyorsabb feldolgozását. A virtuális gépeket általában nem támogatja ez az utasítási készlet, emiatt hiánya észlelésekor feltételezhető, hogy virtuális környezetben vagyunk. [1]

Jól látható a fent említett technikák alapján, hogy jól megtervezett stratégia, illetve minél pontosabb, komplexebb számítás szükséges a virtuális környezet észleléséhez.

4 Népszerű VME detektáló kártevők

Az elmúlt években ezeket a technikákat alkalmazták a támadók. Szeretnék néhány virtuális környezetet észlelő rosszindulatú programot bemutatni, amelyek az ilyen jellegű módszerekkel emlékezetes támadást vittek véghez.

Az Andromeda egy trójai típusú, amelyet 2011-ben észlelték. A leginkább érintett országok India, Vietnám és Irán volt. Amikor a kártevő virtuális környezetet észlelt, akkor elrejtette a jellegzetes kártékony tulajdonságát, mint a legtöbb kártevő a megfertőzött számítógéptől információt lopott, amely email és fájlmelléklettel terjedt, például .docx, .xls, .pdf, .zip kiterjesztésű fájlokkal. A rendszerből kép fájlokat töltött lefutatott, távoli shell-eket hajtott végre és képes volt eltávolítania saját magát. [17]

A ZeuS Trojan Horse-en alapuló rosszindulatú program leginkább banki kártevőként volt jelen, melyet 2014-ben észleltek először. Ez a Dridex volt, mely már a Cridex továbbfejlesztett változata. A virtuális gépen végzett dinamikus elemzéskor nem működik a kártékony kód, elrejti a saját karaktereit, illetve megváltoztatja a virtuális gép viselkedését. A támadók email keresztül fertőzték meg az eszközöket, mely által képesek voltak lopni különböző személyes információkat és banki hitelesítő adatokat. Az ehhez hasonló banki kártevő a Vawtrak, ami a Dridex és a zsarolóvírus, mint a Locky ötvözete, azaz multifunkcionális kártevő. Az áldozatoktól tanúsítványokat, hitelesítő vagy egyéb személyes adatokat, FTP jelszót el lehet vele lopni. [18] [19]

Talán a legismertebb zsarolóvírus a 2016 nyarán felfedezett Locky volt, mely leginkább spam formában támadott a különféle levelezői alkalmazásokon. Az áldozatoknak a fájljait titkosította váltságdíj fejében. A zárolt mellékletek .docx, .xls és .zip kiterjesztésű fájlok.

A szintén zsarolóvírus családba tartozó Cerber 2016-ban észlelték a hatos verzióját, amely képes volt észlelni a virtuális gépek, illetve a homokozók meglétét. A fejlett anti-vm kártevő levelezésen keresztül leli meg a potenciális áldozatait. A szociálisan jól megtervezett spam üzenet egy .zip kiterjesztésű fájlt mellékelnek, mely egy kártékony kódú java script-et, illetve egy PowerShell parancsfájlt tartalmaz. A mellékelt fájlok által megtörténik a számítógép fertőzése. A legtöbb ilyen típusú zsarolóvírus támadás az Egyesült Államokban, Japánban, Tajvanban, Ausztráliában, illetve Kínában történt. [18] [20]

A publikált virtuális környezetet észlelő rosszindulatú programok tulajdonságait a Mitre Att&ck oldal megtalálhatóak. Az oldal egy amerikai non-profit szervezet által fenntartott, amit kutatási és fejlesztési központok irányítanak, és célja, hogy egy hivatalos és megbízható kiberbiztonsági információkat osszon meg az olvasók számára. Különböző sérülékenységek is megtalálhatóak, melyet CVE kezdetű azonosítókkal látnak el. [21]

5 VME detektálási technikák kiküszöbölése

A fejlődő a technológia ellenére az informatikában megszokhattuk, hogy nem létezik tökéletesen kialakított biztonságos rendszer, így tökéletes megoldása soha nem lesz a virtuális környezet detektálási technikák ellen. Viszont léteznek olyan eszközök és megoldások, melyek által többnyire meghiúsíthatóak. A következő ajánlott megoldások csak lokális VME észlelési technikákat kezeli.

5.1 Kommunikációs csatorna

Az első fortély a széles körben elterjed VMware kommunikációs csatorna problémájára nyújt megoldást. A vendég gépen különböző paraméterű „.vmx” kiterjesztésű konfigurációs fájlok találhatóak meg, amivel a VMware gazdagép operációs rendszerén megváltozhatok módosítással, vagy hozzáadással. A beállítási opciók lehetővé teszik az különböző alrendszerek viselkedésének és a vendég gépen lévő komponenseinek (hajlékonylemez, CD-ROM stb.) vezérlését. [1] [10] [16]

A bináris fordítási módszer lényegében, ami megváltoztat néhány vendég gépnyelvi utasítást, ezáltal megakadályozza, hogy a kártékony program eljusson a gazdagéphez. A konfigurációs beállítások a vendég gépen lévő „.vmx” fájlokban helyezkednek. Ez a módszer

```
isolation.tools.getPtrLocation.disable = "TRUE"
isolation.tools.setPtrLocation.disable = "TRUE"
isolation.tools.setVersion.disable = "TRUE"
isolation.tools.getVersion.disable = "TRUE"
monitor_control.disable_directexec = "TRUE"
monitor_control.disable_chksimd = "TRUE"
monitor_control.disable_ntreloc = "TRUE"
monitor_control.disable_selfmod = "TRUE"
monitor_control.disable_reloc = "TRUE"
monitor_control.disable_btinout = "TRUE"
monitor_control.disable_btmemspace = "TRUE"
monitor_control.disable_btpriv = "TRUE"
monitor_control.disable_btseg = "TRUE"
```

Ábra 3 [1]: „.vmx” fájl konfigurációja

a Red Pill, illetve a Scoopy-t is kiküszöbölte. Az ábrán a fent említett „.vmx” kiterjesztésű fájl látható. A módszer nem hivatalos, nem támogatott a VMware által, így dokumentálatlannak mondható. [1] [14] Az ábrán a VMware „vmx” fájl karakterlánc átkonfigurálása van bemutatva.

A másik megoldás virtuális kommunikáció csatorna elleni támadáshoz a kommunikációs csatornához társított „VMXh” varázs értékének a megváltoztatása. Ezzel a kiküszöbölhető a Jerry nevezetű VME detektálási módszer. [1] [16]

Kutatók kifejlesztettek egy VMmutate nevezetű eszközt, ami egy VMware lemezképen keresztül megváltoztatja a VMware binárist és keresi azt, illetve a felhasználó által definiált értékre módosítja a fent említett „VMXh” értéket. Az eszköz kódja ellenőrzi az értéknek a kontextusát mielőtt elvégezi a módosítást. Sajnos, még így is egy jó támadási felület a kommunikációs csatorna a virtuális környezet észleléséhez. [1] [11]

5.2 Memória manipulánsa kártevők ellen

A kártevők memória ellenőrzésének tevékenységét a gyakran használt utasítások megfigyelésével (SIDT, SLDT, SGDT, STR stb.) és az utasítások miatti használt regiszter értékeket érdemes megváltoztatni, mert a kártékonykódú programok a VME detektálásakor az utasítások értékeit beolvassák a regiszterekbe. [20]

5.3 Nyilvántartások, folyamatok, fájlrendszerek

A virtualizáció észleléskor a rosszindulatú programok API-hívások végrehajtása által szereznek különböző információkat az adott rendszerből. Mikor a kártevő a vendég gépen lekérdezi a különféle fájlokat, mappákat, akkor az API hívást érdemes elfogadni, viszont egy egyéni üzenetet érdemes kiadni a kártékony program felé, hogy megtagadja a kért virtualizációval kapcsolatos fájlok felismerését. Később erről a jelenségről is szó lesz, melyet API hooking-ként fogok definiálni. [20] [22]

A leírt megoldásokon kívül a már egyik fejezetben említett virtualizációs biztonsági megoldások alkalmazásával, a különböző eszközök, vagy beállításokkal megfékezhetőek a VME detektáló kártékony programok.

A bejegyzési kulcsokat a Windows operációs rendszerben a Registry Editorban található meg grafikusan. A virtuális gépre utaló bejegyzési értékeket módosítani vagy akár törölni kell. A dxdiag és a Device Manager használatával jól ellenőrizhető a módosítható bejegyzések adatai. A Device Manager-ben a gépnek a hardveres tulajdonságait tartalmazza, a dxdiag-nál meg az adapter adatok állapíthatóak meg. A virtuális környezet detektálásk elkerülés érdekében érdemes minél inkább, olyan értékeket megadni, hogy minél jobban hasonlítson egy fizikai gépre. A módosítási folyamat végén érdemes a „Systeminfo" paranccsal a command prompt-ban ellenőrizni a gép tulajdonságait. Általában a parancs lefutásakor, ha manuálisan állítgattuk az értékeket, akkor nagy valószínűséggel lesznek a használt virtualizációt támogató szoftverre utaló nyomok. Ezeket virtuális gépet elrejtő szoftverekkel vagy saját magunk által írt programmal el lehet rejteni. [11]

A segédkomponensek, mint például a VMware tools nevű szoftvert el kell távolítani a virtuális gépről, hogy ne legyen árulkodó nyom a virtuális környezetre. Általában ezek a segédprogramok segítik a virtuális gépek kinézetét, és felhasználó számára kényelmesebb és egyszerűbb beállításokat tartalmaz, mint például a megfelelő képernyőfelbontás biztosítása.

A malwarek figyelik a különböző hardver tulajdonságokat, így például érdemes minél nagyobb memória méretet, merevlemez méretet, CPU számot megadni, hogy minél jobban hasonlítson egy fizikai gépre. Amellett a hálózati kártyát érdemes letiltani, és inkább egy USB adapter használni, illetve az alapértelmezett MAC címet mindenképpen érdemes átírni.

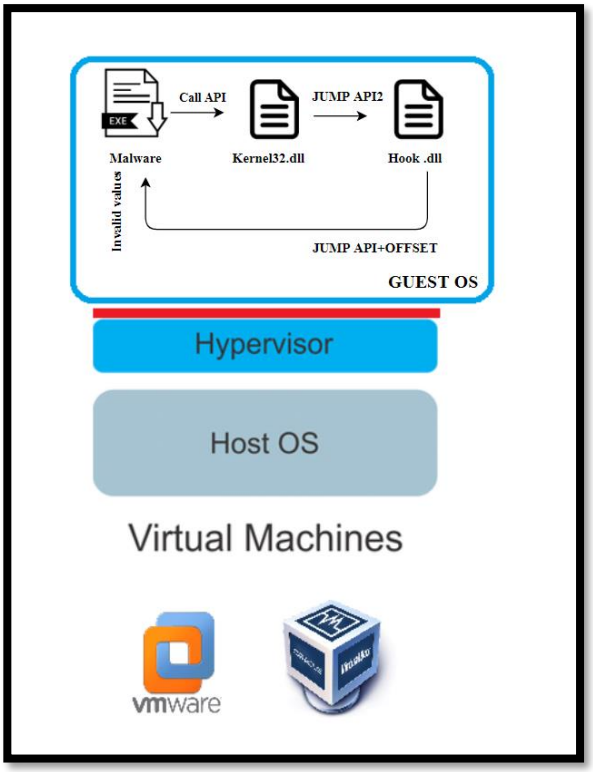
A további árulkodó elemeket, mint például a virtualizációra utaló fájlok (például: C:\windows\System32\vboxtray.exe), folyamatok (például: vboxservice.exe), vagy további futó szolgáltatások (például: Vmscsi) el lehet rejteni különböző szoftverek segítségével. Vagy API hooking megoldás által meg lehet valósítani, hogy a fenyegető kártevő téves információt kapjon vissza ezekről a tulajdonságokról. [23]

6 Tesztkörnyezet

Az népszerű alapvető technikákat megvizsgálva, illetve a különböző virtuális infrastruktúrák felmérése alapján szeretnék megvalósítani egy olyan virtuális környezetet, amely megnehezíti a később rajta lefutó rosszindulatú programok VME detektálását.

A virtuális infrastruktúrákon a legelterjedtebb operációs rendszert fogok igénybe venni, azaz Windows10 operációs rendszert. A különböző virtualizáció támogató termékek használatával, mint például a VMware, VirtualBox stb. törekedni fogok, hogy olyan virtuális környezetet hozzak létre, amely az ismertebb VM észlelési technikákat megfelelően tudja kiküszöbölni a 5.fejezetben említett megoldások alapján. A legjobbnak tűnő virtuális gépen egy kártevő futtatása alapján fogom tesztelni a módszerek hatékonyságát.

A létrehozott tanulmányi kutatások által kapott eredmény értékek alapján megfelelően levonhatom az esetleges tanulságot.



Ábra 4: Tesztkörnyezet

A tesztkörnyezet végső elképzelése elméletben a 4.ábra mutatja meg. Az ábrán a használt virtuális gépeket támogató szoftvereket, azaz a VMware és a VirtualBox emblémáját jelenítem meg, amiket a vizsgálat során is alkalmazni fogok. Az alap virtuális gép szerkezete a fizikai gépen lévő operációs rendszerével kezdődik. Ez az ábrán a „Host OS”, azaz a gazdagép

operációs rendszert képviseli. A következő réteg a hypervisor, ami virtualizációnak a fő elemének mondható. Feladata, hogy teljes elszigetelést biztosítja a gazdagép és a vendéggép között. A struktúrának köszönhetően lehetővé teszi, hogy egyszerre több virtuális gép futhasson. Az ábrán a felső réteg a „Guest OS”, azaz maga a vendéggép operációs rendszere helyezkedik el. A különböző módosítások és megfelelő beállítások mellett fontosnak találtam szemléltetni egy lényeges folyamatot. Az ábrán egy „.exe” fájl található meg, melynél egy rosszindulatú virtuális környezetet észlelő kártevő fájl futtatására gondoltam. Általában a legtöbb VME detektáló malware API hívásokat hív le, hogy felederítése a különböző tulajdonságait a környezetnek. Majd ez alapján le tudja vonni a következtetést, hogy virtuális környezetben van-e vagy sem. A teszt során Windows operációs rendszert fogok használni a gazdagépen, ezért most példaképpen kernel32.dll-t ábrázoltam a folyamat ábrán. A kártevő által kért API hívás a megfelelően módosított kernel32.dll fájlban tovább ugrik ez a kérés, egy „.dll” fájlban. Ez fájl olyan funkciót tölt be, melynél a felderítő kártevőknél egy hamis értéket fog visszaadni. Ennek köszönhetően például egy virtualizációt észlelő zsarolóvírus nem fogja meg tudni megnyitni a fájlokat, és így titkosítani sem fogja tudni. Ezzel a módszerrel a legtöbb ilyen tulajdonságú rosszindulatú programot jól lehet hatástalanítani.

6.1 Tervezés

Az alapos irodalmikutatás után megvizsgáltam, illetve utána megterveztem az esettanulmányt pontos menetét. A legtöbb kutatásnál a legnépszerűbb virtualizációt támogató szoftvereket használták. Előnyük, hogy többnyire könnyen elérhetőek, illetve ingyenesek a felhasználók számára. A hatékony anti-detektáló technikák elsajátításához jobbnak láttam az ismert szoftverek használni. Tervezéskor az irodalmi kutatások mellett, mindkét konzulensem véleményét ki kértem, és az alapján döntöttem a fent említett szempontokon kívül.

6.1.1 A virtualizációt támogató szoftverek és vendéggépen futó OS kiválasztása

Az esettanulmány tervezésekor fontos volt számomra, hogy virtualizációt támogató termékekkel dolgozzak, amik egy átlagos felhasználónak könnyen és olcsón elérhető

A legtöbb kutatási cikkben a kutatók a VMware, VirtualBox és a XEN általi termékeket használták, mert a piacon ezek a legelterjedtebbek, így nagyobb a valószínűsége, hogy a detektáló malwarek is ezekre a termékekre működnek a leghatékonyabban. Ennek függvényében a VMware Workstation Pro-t és a VirtualBox-ot választottam ki.

Korábban már dolgoztam ezekkel a szoftverekkel, így ez is oka volt, hogy végül ezeket választottam. Azonfelül ezekkel a szoftverekkel kapcsolatban több információ, módszer megtalálható a virtualizáció észlelő kártevők elleni védekezésével kapcsolatban.

A vendéggépen futó OS választás volt számomra a legkönnyebb, mert ott biztosan tudtam, hogy a világon legismertebb operációs rendszert, a Windows 10-et választom. Linux-os operációs rendszert azért nem választottam, mert azáltal egyszerűbb lett volna a megvalósítás. A Windows-os operációs rendszer kihívásnak tűnt számomra, mert nehezebben alakítható át, mint egy Linux-os OS. Illetve szerettem volna bemutatni, hogy hétköznapi laikus személy által, hogy lehet megoldani egy ilyen jellegű problémát.

Az esettanulmányának tervezésekor a különböző kutatásokat vettem alapul, mint például Ed Skoudis és Tom Liston által bemutatott kutatás [1].

6.1.2 Célok kitűzése

Tervezéskor fontos, hogy legyen egy fő cél, ami ennél az esettanulmánynál a VME elrejtése, és ennek a környezetnek való módosítása, hogy minél jobban egy fizikai gépre hasonlítson.

Ezért elengedhetetlen kitűzni ezeket a változásokat:

- Telepített segédprogramok törlése: VMware Tools, VirtualBox Guest Additions
- Eszközök érétkeinek átírása
- Memóriaméret megemelése
- Minimum 2vCPU használata
- Hálózati kártya eltávolítása
- Képernyőfelbontás megváltoztatása
- Rendszerleíró kulcsok módosítása

6.1.3 Tesztelés tervezése

A biztonságos tesztelés érdekében biztos voltam, hogy a „vm escape” tulajdonságú malwarekkel nem szeretnék foglalkozni, mivel nincs hozzá megfelelő tudásom és tapasztalatom. Illetve azt is számításba kellett vennem, hogy nem feltétlenül fogok tudni megteremteni olyan környezetet, ami egy komolyabb detektáló malwarenél ellen tud állni.

Tervezéskor így mindenképpen a külső konzulensem véleményét kértem ki a biztonságos tesztelés érdekében. Így tesztelni néhány séma, egyszerű malwaret fogok a megvalósított környezetben, ami alapján következtetéseket tudok levonni.

A kiválasztott kártevők:

- Andromenda
- RansomwareWannaCry
- RansomwareLocky
- RansomwarePetya
- RansomwareCerber

6.2 Megvalósítás

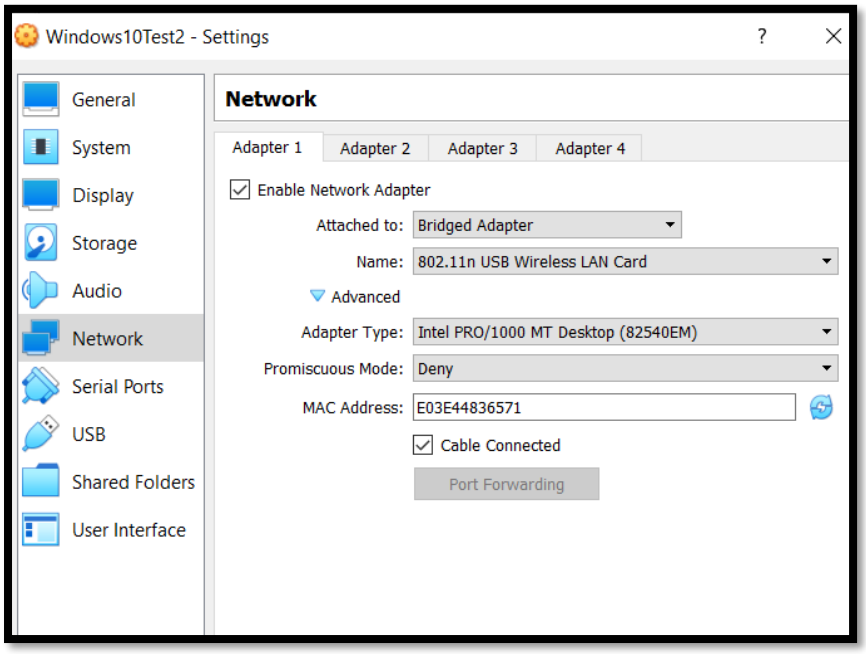
Az alapos irodalmikutatást követő tesztelés után a megvalósítás volt a következő lépés. Az eddigi ismeretek mellett, még jobban utána kellett nézni néhány megoldás miatt. A tesztkörnyezet létrehozásakor a cél az volt, hogy minél jobban hasonlítson egy fizikai számítógépre. Néhány detektálási technikánál a saját fizikai gépen hardveres kapacitását számításba kellett vennem.

6.2.1 Hálózatikártya/MAC cím

Az alapértelmezett MAC cím VMware-nél: „00-0C-29-83-65-79” és VirtualBox-nál: „08-00-27-B1-1D-5A” volt. Illetve alapértelmezetten NAT, virtuális hálózat volt beállítva a virtuális gépeken.

Az első lépés a megfelelő tesztkörnyezet megteremtéséhez a virtuális hálózat eltávolítása volt. Mivel egy egyszerű leleplező nyom lehet egy „vmnet” az észlelő kártékony program számára. A hálózati kártyát eltávolítottam, majd hálózati kártya helyett egy USB hálózati adaptert használtam. Az USB wifi adapternek köszönhetően virtuális gép fizikai hálókártyának fogja érzékelni. Így leváltható lesz a virtuális hálózat, azaz „vmnet”. A

beállításokat hasonlóan kell elvégezni a VirtualBox és a VMware-nél. A sima beállításokon belül a hálózatoknál be lehet állítani a csatolt adapter típusát.

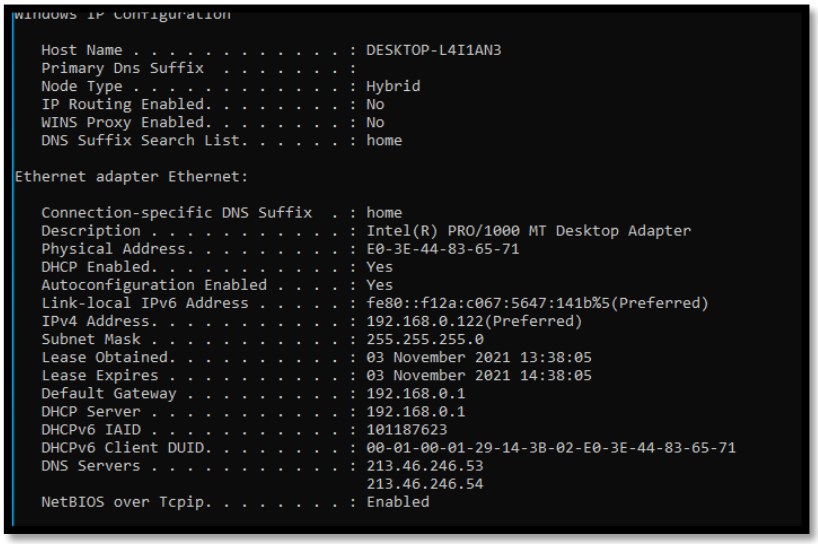


Ábra 5:USB Adapter beállítása

A következő lépés az alapértelmezett MAC-címek módosítása volt.

A különböző virtuális környezetet észlelő rosszindulatú programok az alapértelmezett MAC-címeknél rögtön kiderítik, hogy virtuális gépről van szó vagy sem. Általában ezek virtualizációt támogató programoknál más-más szokott lenni az alapértelmezett MAC-cím A VirtualBox-nál az alapértelmezett :080027000001, 080027000002, 080027000003 stb. azaz a 080027 kezdetű értékkel szoktak kezdődni. Míg például a VMware-nél a gyárilag beállított MAC-cím érték kezdete 00:50:56.

Hasonló helyen található a MAC- cím értékének átírása, amelyet szintén a hálózati beállításban található. Fontos, hogy a virtuális gépre vonatkozó különböző beállításoknál a vendéggépnek kikapcsolt állapotban kell lennie. A MAC-cím adásnál érdemes nem a generálásra hagyatkozni, hanem az általunk létrehozott egyedi cím értéket érdemes beírni. Majd a konfigurálás után érdemes elindítani a virtuális gépet, hogy megnézzük, hogy jól lett-e beállítva a virtuális gép hálózati része. Az ellenőrzést az „ipconfig /all” vagy „wmic nic list” paranccsal lehet. Az ábrán én az „ipconfig /all” ellenőrzési megoldást választottam. A 6.ábrán jól kivehető, hogy az alapértelmezett MAC-cím megváltozott E0-3E-44-83-65-71 értékűre, illetve az ipv4 érték alapján az USB adapter használata is sikeresnek mondható.



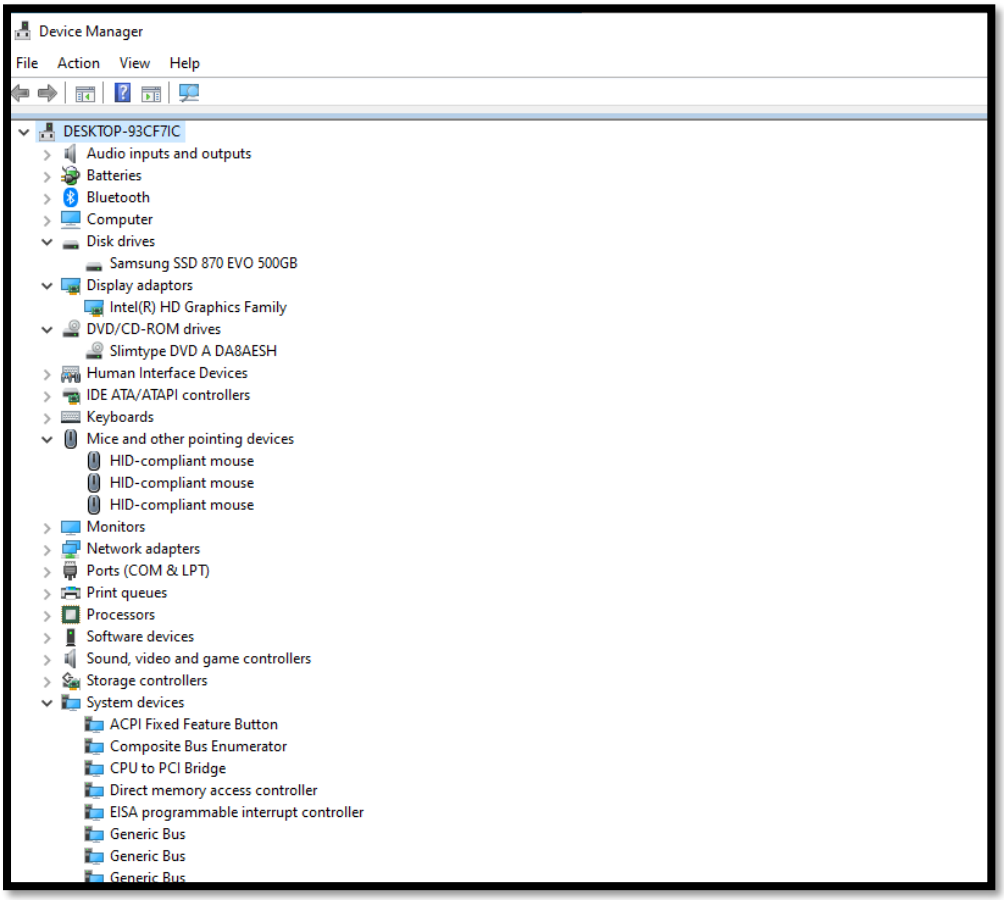
Ábra 6: A hálózat ellenőrzése módosítás után

6.2.2 Hivatkozások, fájlok módosítása

A virtuális detektáló technikáknál már említésre került a virtualizációra utaló hivatkozások, fájlok és háttérben futó alkalmazások jelensége. A különböző virtuális gépnek hardveres tulajdonságai, jellemzői a „Registry Editor” -ként megtalálható a Window10-es operációs rendszereknél. Az említett adatbázis magyarul rendszerleíró adatbázisnak szokták nevezni, célja a Windows alapú operációs rendszer fájljait, hardvereszközeit, felhasználói és konfigurációs értékeit tárolja. Egy új program telepítésekor például egy új utasításikészlet, illetve fájlhivatkozás fog bekerülni az adatbázisba, azaz a Windows regiszter tárolójába. Az adatbázis segítségével az információt lehet kapni, hogy a keresett fájlok hol találhatóak meg. Az Windows operációs rendszer rendszerleíró adatbázisát RAB-ként is szokták rövidíteni. A RAB-ban az adatok hierarchikusan, könyvtárszerű szerkezetként vannak elhelyezkedve. Az adatbázisban levő adatok, vagyis kulcsok tartalmazhatnak még alkulcsokat és értékeket is.

Ennek tudatában a VME-re utaló különböző kulcsok értékét módosítottam. A főmappák elnevezései megkönnyítették a kulcsok keresését, mert így átláthatóbban tudtam haladni az adatbázis adatainak átalakításával. A virtuális gépre jellemző tulajdonságok a „HKEY_LOCAL_MACHINE” alatt voltak találhatók, amik további egymásba ágyazott kulcsok között voltak megtalálhatóak. Viszont voltak olyan mappák, ahol a kulcsok értékeit nem lehetett átállítani, mert átírásával a virtuális gép alapműködését befolyásolta volna. Az értékek átírása után a „Device Manager” és a „Dxdiag” segítségével tudtam ellenőrizni az

elvégzet művelet eredményét. A Windows eszközkezelője lehetővé teszi, hogy grafikusan láthatóak legyen a különböző hardveres tulajdonságok, értékek. Szintén hierarchikusan van a felépítése, hogy felhasználó számára jól látható legyen. A másik ellenőrzési megoldás a „systeminfo” parancs. A rendszer kilistázza az összes tulajdonságát a gépnek. Illetve még ellenőrzési opció a „Dxdiag” azaz teljesen nevén „DirectX Diagnostic Tool”. A Windows alapú diagnosztikai eszköz segítségével megtudható a különböző hardver tulajdonságok. A módosítások után a rendszerleíró adatbázisban még kiexportáltam a módosításokat, hogy biztosan elmentsem. Az ábrán az átírt értékek egyi ellenőrzés módja látható, melynél a fizikai gépre hasonlító értékek láthatóak.

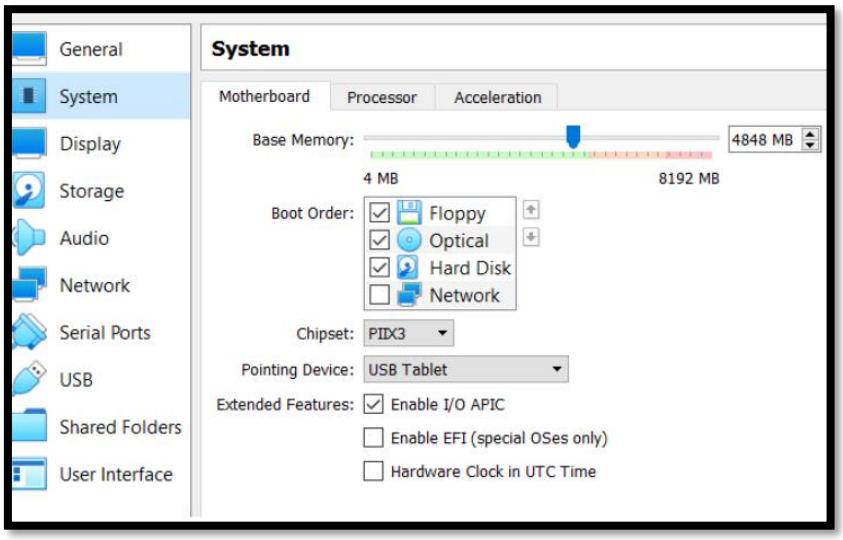


Ábra 7: A rendszerleíró kulcsértékek módosítás utáni ellenőrzése

6.2.3 Alap hardver tulajdonságok beállítása és segédprogramok eltávolítása

A VME kártevők az észlelésnél a rendszer alap eszköz tulajdonságait is megfigyelik. Ezért érdemes arra törekedni, hogy minél jobban hasonlítson egy fizikai számítógépre.

Először a processzor beállításával kezdtem, ami a virtuális gép alapbeállításán belül mindkét szoftveren a rendszer résznél állítható át. Sajnos a használt fizikai gépem kapacitása révén nagy változást nem tudtam elérni, de amennyire lehetséges volt a processzor számát megemeltem annyival, amennyivel tudtam. Például a VirtualBox CPU értékének módosításánál 2-re lett. A processzor átállítása mellett a többi alapvető hardver értéket is át kell konfigurálnom. Az ábrán a VirtualBox memória mérete látható, ami szintén rendszer beállításában található meg. Az álcán felül a teljesítmény növelés érdekében a virtuális gép memória mennyiségét is megnöveltem maximális értékre, azaz jelenesetben 4848 MB-ra. A memória méret növelése mellet, a merevlemez méretét is növeltem, hogy hardvert tekintve minél inkább fizikai gép tulajdonságaira hasonlítson.



Ábra 8: Virtuális memória méretének növelése

Az alapbeállítási változtatásokon kívül, a további lépés a különböző segédprogramok eltávolítása vagy elrejtése volt.

A rosszindulatú programok különböző lekérdezésekkel, mint például egy Win API hívásnál vagy Tasklist.exe lekérdezésnél az operációs rendszerben futó virtualizációra utaló alkalmazások, folyamatok elárulhatják a virtuális környezet jelenlétét. Ezért első lépésként a VMware-en futó „vmtools”, illetve a VirtualBox-on lévő „Guest Additions”-t távolítottam el a

virtuális gépről. Ezek a programok lehetővé teszik, hogy szorosabb integrációt valósítson meg a gazdagép és a virtuális gép operációs rendszerei. Az eszközillető programok segítségével a vendéggép használhatóságát és teljesítményét növelik. A tulajdonságaik közé sorolható, hogy a vendéggépen lévő operációs rendszer ablakai zavartalanul tudnak futni a gazdagép operációs rendszer felnyiló ablakai mellett. Azonfelül az egyszerű egérmutató integrációját, a mappák megosztási módját, a fájlokkal kapcsolatos műveleti végrehajtásokat is egyszerűbbé teszi a két gép között. Azonfelül a vendég és a gazda közötti kommunikációs csatornánál az időszinkronizálást is javítja. Az említett tulajdonságokon kívül még van több jellemzője. A segédprogramok eltávolításnak hátránya a hasznos tulajdonságok elhagyása volt. Ami közé beletartozik a megfelelőképernyő felbontás is. A kártevő ezt is észre tudja venni, így fájó szívvel, de úgy döntöttem, hogy az telepített programok elhagyása fontosabb.

Az eltávolítás mellett, szoftverek segítségével el is elrejthetőek az esetleges szolgáltatások, vagy háttérben futó virtualizációt támogató folyamatok. A tesztkörnyezet kialakítása során nem akartam más külső szoftvert használni, hanem saját magam által akartam létrehozni a környezetet. A VMware kapcsán egy olyan megoldást találtam, melynél a Resource Hacker nevű alkalmazást használt. A szoftver egy olyan ingyenes program, mely lehetővé teszi a különböző alkalmazások módosítását, és elrejtését. A VMware-nél ezt a módszert alkalmaztam. Azonfelül szintén a VMware kapcsán a „.vmx” fájlba hozzáadtam olyan karakterláncokat, mely során a virtuális gép jobban elrejthető. A fájlban beillesztett karakterlánc:

monitor_control.virtual_rdtsc = "false"
monitor_control.restrict_backdoor = "true"
isolation.tools.getPtrLocation.disable = "true"
isolation.tools.setPtrLocation.disable = "true"
isolation.tools.setVersion.disable = "true"
isolation.tools.getVersion.disable = "true"
monitor_control.disable_directexec = "true"

6.2.4 Rendszer API-k használata

Az API vagyis Application Programming Interface lehetővé teszi, hogy két alkalmazás tudjon kommunikálni egymással, lényegében egy szoftverközvetítő. A gazdagép és a vendéggép között is API függvényhívások, üzenetek történnek. A legtöbb virtuális környezetet detektáló kártevők ezt módszert is használják, ami ellen vannak különböző megoldások.

A virtuális gép használatakor különböző API hívások mennek végbe. A virtuális környezetet detektáló kártevők ezt használják ki. A „cpuid" utasításakor a bemeneteli érték például „0x0", ami gyártóként eltérő érték, de mindig konstans alapértelmezetten. A bemeneteli értékek használatakor visszaadja a CPU gyártóazonosító karakterláncát. Ezáltal kideríthető, hogy az illető például VirtualBox-ot használ-e vagy sem. Az alacsony szintű módszereken kívül magas szintű módszereket is használnak. Például a VirtualBox-nál: „*HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\VBOX__",* [24]

Ami meg a „*RegOpenKey()*" függvényt használja, mely által virtuális gépet tudja észlelni. Ha a kért kulcs létezik, akkor a rendszer észleli a virtuális gépet. Így elmondható, hogy a VME detektáló kártékonykódú programok ezeket az API hívásokat is megfigyelik, lehallgatják, ami által azonnal be tudja azonosítani a kártevő, hogy virtuális környezetben van-e vagy sem. [22] [23] [24] [25]

Jól látható, hogy a kártevő a különböző kéréseket és az alapértelemezett API hívásokat kihasználja, és ezáltal fel tudja deríteni a környezet tulajdonságait, majd ez alapján dűlőre tud jutni, hogy fizikai vagy pedig virtuális környezetben fut-e éppen. A legtöbb jól megírt kártékonykódú program felderítéskor ezeket a technikákat használja először.

A felvázolt problémára a megoldás az API hooking. Ez a módszer egy olyan alacsony szintű technika, amelynél az adott szoftverkomponens viselkedését lehet manipulálni. Az virtuális környezet kiküszöbölése esetén az adott kernelben, amit a virtuális gép használ, meg kell írni egy olyan alacsony szintű kódrészt, amelynél a rendszerhívásoknak a visszatérési értékét megváltoztatja. A szakdolgozat már említett módszerként írtam a speciális gépi utasítások megfigyeléséről, mint például a „cpuid”, „RegOpenKeyEx()” utasítások. A virtuális gépet észlelő malwarek, szofisztikált kártevők is ezt az eljárást használják. Sokan úgy valósítják meg a hooking-ot, hogy a virtuális környezetben futó malwaret hookolják meg a védő által megírt DLL befecskendezésével. A folyamat eredménye az lesz, hogy a kártevő fájlok megnyitásakor mindig érvénytelen értéket fog kapni, ami által nem fog működni és kárt tenni. Ez a módszer jól működik a legtöbb VME detektáló malwarenél. [22] [23] [24] [25]

Az API hooking megvalósításakor sok irodalmi kutatás által rájöttem, hogy a tanulmányaim nem elegendőek egy ilyen megoldás leimplementáláshoz. Magasfokon kell érteni az architektúra teljes működéséhez, hogy bele lehessen nyúlni a kernelbe. Illetve megfelelő szakértelem szükséges egy ilyen módszer megírásához.

Mindenképpen fontosnak tartottam, hogy megmutassam az ötletet. Az irodalmi kutatások során néhány szakembernek a megvalósítását sikeresen megtaláltam, ami által be tudom mutatni az ő általuk megvalósított megoldást.

A kártevők az API hooking használatával szinte teljes mértékben tudja irányítani a futó rendszert. Először képesnek kell lenniük a kód futtatására a folyamaton belül, amit kód befecskendezéssel lehet megoldani. Megvalósítani shellkód vagy teljes könyvtári fájlok, mint például DLL fájlokkal lehet elvégezni. Általában támadáskor az áldozat címterében API horgokat helyeznek el. [24]

Az irodalmi kutatás során talált kísérletnél egy notepad.exe-t teszteltek. Az „exe” kiterjesztésű fájlok Windows operációs rendszerben Win-API-t fog használ, például egy WriteFile() folyamatnál. Általában a C:\Windows\System32\Kernel32.dll-nél találhatóak meg a Windows DLL-je, ami szükséges a Win-API-k hívásakor. A már említett notepad.exe új modult fog kapni az eljáráskor, ami a kernel32.dll egyik példányát fogja megkapni és ezáltal a vele járó összes funkciót is. Elemzéskor a Ghidra nevű disassembler-t használták, mely segítségével a kernel32.dll példányát elemezték. A „program trees” résznél importált és exportált API-kat mutatja meg. A „decompiler” ablaknál a meghívott funkció a WriteFile pontos alkotó elemei láthatóak. Középen meg a WriteFile függvény összeállított kód része látható. [22]

Ábra 9 [22]: A kernel32.dll vizsgálata Ghidra-n

Irodalmikutatás folyamán találtam egy érdekes megoldást a malwarek által használt API-k ellen, ami a Github-on „nybble04/Shady-Hook”ként megtalálható a meg. A megoldás menete következő képpen valósul meg. A készítő az általa megírt DLL fájlt befecskendezi az tesztelt zsarolóvírusba. A beinjektált DLL fájl összekapcsolja a kártevő által a meghívott API CreateFileW() függvényt, hogy titkosítás előtt megnyissa a zsákmányul ejtett fájlokat. A vezérlő DLL-en belül egy MyCreateFileW() függvényhez fog vinni, ami meg módsítani fogja a meghívást, hogy mindig érvénytelen kezelőértéket , azaz „invalid handle”-t fog vissza adni. Ezáltal a zsarolóvírus titkosítani és még megnyitni sem fogja tudni a fájlokat, dokumentumokat. Ezáltal hatástalanítva lett a kártékony program. Viszont ez a megoldás jelenesetben csak kifejezetten zsarolóvírusokra van, ami miatt nem feltétlenül tökéletes megoldás. Ez a fortély átalakítható más kártevő típusokra is, viszont lehetetlen, hogy minden kártevőre naprakész

legyen. Elképzelhetőnek tartom, hogy egy automatizált rendszerben, AI segítségével akár kivételezhető a legtöbb kártevő ellen is. Sajnos erről konkrét megoldást nem találtam. [26]

6.3 Tesztelés

Egy átgondolt és megvalósított virtuális környezet tesztelésére a legjobb módszer maga a virtuális környezetet detektáló kártékonykódú programok futtatása. Természetesen tisztában voltam vele, hogy tökéletes környezetet nem tudtam megvalósítani, ami egy „vm escape" tulajdonságú malwarenél végzetes lehet. Ezért teszteléskor csak megbízható forrásból, jól ismert kártevőket futtattam, melyek nem képesek átterjedni a fizikai rétegre, hanem csak észlelik a virtuális teret. A másik fontos teszt, az API hívások megfigyelése volt, miközben fut a malware.

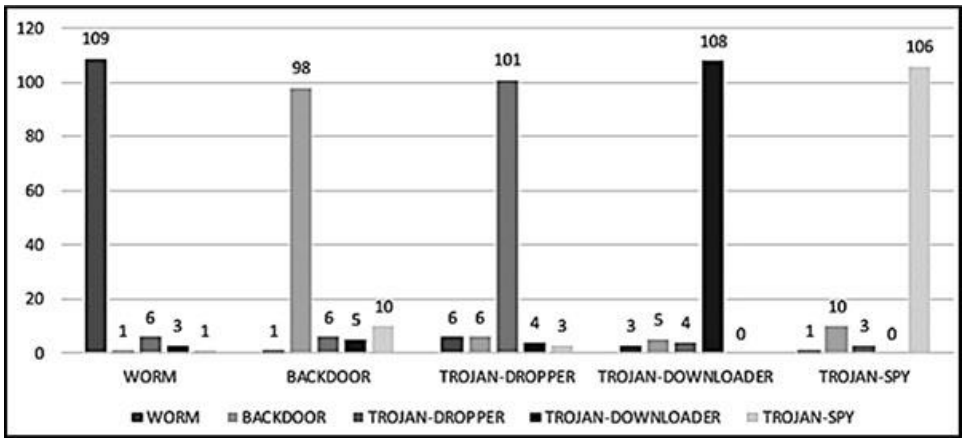
6.3.1 API hívások megfigyelése vm detektáló malware futásakor

A kutatók vizsgálat során különböző rosszindulatú programokat teszteltek, melyek API hívásokat használnak. Az elemzett kártevők a féregből, Backdoor-ból, és több Trojan családban lévő malwarekből álltak, amelyek többnyire jó indulatú programként álcázzák magukat. A kísérlet során összesen 120 mintát futtattak le virtuális gépen. A Windows operációs rendszer számára a Win-API-k biztosítják a megfelelő hozzáférést, kommunikációt az alapvető erőforrásokhoz, melyek a Windows DLL fájlokban van definiálva. Ilyen DLL fájl például a kernel32.dll, shell32.dll stb. Kutatás során a kutatók 534 API hívást vettek igénybe. Kísérletkor C nyelvben megírt API hooking-ot írtak, ami által módosították az API hívások viselkedését. A hooking történhet felhasználó módban az operációs rendszer kernelében. A felhasználói megoldásnál a harmadik féltő származó könyvtárak összevonásával érhető el. Ezen belül két módszer létezik az „Inline Hooking" és az „Import Address Table (IAT) Hooking". Az IAT hooking vagy magyarra lefordítva címimportálási hooking során, a DLL fájlt felülírják egy „jump" függvény segítségével, ami átugrik az esetleges rosszindulatú funkcionalitás végrehajtási függvényéhez. A tesztkörnyezet használt operációs rendszere Windows XP volt. A kutatók 26 kategóriába sorolták az API hívásokat, aminek a célja, hogy magasabb szintű absztrakciót, redundanciamentes és egyszerűbb modellt kapjanak a kísérlet végén. A minták elemzésekor Fuzzy hash technikát alkalmaztak, mely során jobban kimutathatóak a különböző API hívások. A vizsgálat során megfigyelték a malwareket, és megerősítette a kísérlet, hogy

tényleg a kártevők aktívan használnak API hívásokat, azonfelül az elemzett alanyok tulajdonságára is fényderült. [25]

Az ábrán az API hívások számát mutatja ki a diagram, mely összesen 120 mintavételből állt. A legtöbb API hívást a férek okozták, majd a letöltések által szerzett trójai kártevő követte. A vizsgálatkor többször is lefuttatták a tesztelt rosszindulatú programokat. A használt kategorizált API hívások, mint például az „Input/output Create” vagy az „Registry Read” hívásokat a legtöbb kártevő használta felderítéskor.

Az eredmények alapján jól bizonyítható, hogy kártevők tényleg megfigyelik, vagy akár igénybe veszik az API hívásokat, mely során felfedezik a környezet, vagy akár kárt is okoznak az áldozatnak. [25]



Ábra 10 [25]: API hívások használatának bizonyítéka

A további API hívásokkal kapcsolatos kutatás egy GuLoader nevű volt, amelynél VME detektáló által jól szemlélhető a malwarek hasonló észlelése volt. A GuLoader 2020-ban népszerű VME észlelő kártevő. Az AgentTeslánál, Formbook malwareknél is ezt az észlelési módszert használják. A Check Pointtal végzett vizsgálatok során elképzelhető, hogy a GuLoader kód megegyezik a CloudEye-vel, ami viszont egy legális védelmi bináris mechanizmus. A következő ábrán megfigyelhető az észlelés menete [27]

```
_VM_DETECT      proc near
                cld

_VM_DETECT_START:

                fnop
                xor     edi, edi
                nop
                mov     ecx, 186A0h
                cld
                nop

_VM_DETECT_CONTINUE:

                push    ecx
                call    _RDTSC_OPS
                pop     ecx
                cmp     edx, 32h
                jl      short _VM_DETECT_CONTINUE
                add     edi, edx
                dec     ecx
                cmp     ecx, 0
                jnz     short _VM_DETECT_CONTINUE
                cmp     edi, 0
                jl      short _VM_DETECT_START
                cmp     edi, 68E7780h
                jge     short _VM_DETECT_START
                mov     eax, edi
                retn
_VM_DETECT      endp
```

Ábra 11 [27]: VM detektáló kártevő működése I.

Az assembly kódrész elején az „ecx” regiszter tárolja a 0x186A0 értéket, ami visszaadja, hogy az „edi” regiszter hányszorosára növeli az „RDTS_OPS” függvény által, amíg a művelet eredménye nagyobb, mint 0x32. Majd _RDTSC_OPS függvényt meghívja, ami 0-nál nagyobb éréket adhat csak vissza. A következő lépésként ellenőrzi, hogy a 0x32-nél nagyobb értéket kapott. Ha nagyobb értékű, akkor hozzáadja az eredményt az „edi” regiszterhez, és csökkenti az „ecx” értékét eggyel. Ha viszont a mért érték kisebb mint 0.x32, akkor visszatér a „_VM_DETECT_CONTINUE” függvényhez, hogy újra hívja az „_RDTSC_OPS” függvényt. Az eljárás addig történik, amíg az „ecx” regiszter értéke 0 nem lesz. A függvény eredménye hozzáadódik az „add edi,edx” végrehajtással „0x186A0”-szor az „edi” értékhez. A művelet végén az eredményt ellenőrzi a szoftver, hogy nagyobb-e vagy egyenlő e a „0x68E7780”-nal. Ha kisebb akkor virtuális környezet észlelt, ha nem akkor a művelet visszaáll a „_VM_DETECT_START”-ra. [27]

Az említett „_RDTSC_OPS” használata lényeges ennél a technikánál. A következő algoritmus a két RDTSC hívás közötti végrehajtási időt fogja visszaadni, ha a „cpuid eax” regiszter értéke 1. [27]

```
_RDTSC_OPS
                lfence
                rdtsc
                lfence
                shl     edx, 20h
                or      edx, eax
                mov     esi, edx
                pusha
                mov     eax, 1
                cpuid
                bt      ecx, 1Fh
                jb      short $+2
                popa
                lfence
                rdtsc
                lfence
                shl     edx, 20h
                or      edx, eax
                sub     edx, esi
                cmp     edx, 0
                jle     short _RDTSC_OPS
                retn
_RDTSC_OPS
```

Ábra 12 [27]: VM detektáló kártevő működése II.

A művelet elején az „aex” alsó rész regiszterbe és az „edx” felső rész regiszterbe az RDTSC segítségével lekéri az eltelt időt. Majd a két rész között „vagy” műveletet végez és az eredményt eltárolj az „esi” regiszterbe. Ha az „aex” regiszter értéke 1, akkor a „bt ecx,1Fh” utasítás által ellenőrzi, hogy virtuális környezetben fut-e vagy sem. A „cpuid” meghívásakor virtuális gépből való kilépést generál. Ennek következtében a hypervisor átadj a műveletet a virtuális gép menedzsernek. Így megállapítható pontosan, hogy virtuális gép vagy sem, mert a hívás több időt vesz igénybe, mint egy fizikai gépen. Majd meghívja az „_RDTSC” függvényt, és ismét lekéri az „aex” és az „edx” értékét, azaz eltöltött időt. Ezután megint „vagy” műveletet hajt végre a két regiszter között, és kivonj az „esi” regiszterben tárol előző eredményt. Ha a

kapott eredmény érték 0-nál nagyobb, akkor a függvény „edx” regiszterben adja vissza az eredményt. De ha nem, akkor a függvény elejére fog visszatérni [27].

A vizsgálat jól megismerteti egy virtualizációt észlelő kártevő viselkedését és működését. A legtöbb kártevő hasonlóan működik, így következtetésképpen elmondható, hogy leginkább az API hívások hamis érték adásával lehet megelőzni ezeket a rosszindulatú programokat.

6.3.2 Malwarek futtatása

A teszt során, fontos volt számomra, hogy több malware típust is lefuttassak, megfigyeljek és kielemezzek. A használt kártékonykódú programokat egy megbízható forrásból töltöttem le. [28] A theZoo csomagban vm detektáló malwarek is megtalálhatóak, illetve több híres kártevő is. A csomagban malwarek forráskódjai, hash függvényei, leírása szerepeltek, illetve ami még említésre méltó, hogy ezeket a malwareket phyton-ba írták meg, melyet titkosított tömörített mappával védtek. A kiválasztott malwarek: Andromenda, RansomwareWannaCry, RansomwareLocky, RansomwarePetya és a RansomwareCerber volt. Azért választottam ezeket a kártevőket, mert többnyire VME észlelő tulajdonságúak, illetve a zsarolóvírusi tulajdonságuk miatt, még látványosabb eredményt mutat a teszteléskor. A tesztelt kártevők hash függvény kódjait teszt előtt megvizsgálat a Virustotal használatával, és megbizonyosodtam róla, hogy tényleg érvényes tesztre létrehozott malwarekről van szó. A használt ellenőrző weboldalon a kártevők tulajdonságait és viselkedéseit leírják, ami számomra nagyon hasznos volt.

Tesztelés előtt ellenőriztem az általam készített anti-vm technikákat, melynél a VirtualBox-nál és a VMware-nél egy kicsit eltért. A biztonságos teszt érdekében a hálózatot kikapcsoltam, melyet a például a VirtualBox-nál a Network-on belül a „no attached” -ra menve lehet, illetve ellenőriztem, hogy a gazdagép és vendéggép között nincs semmilyen jellegű fájl megosztás. Majd a virtuális gép futásakor kikapcsoltam a Microsoft Defender Firewall-t. Ha bekapcsolva hagynám a domain, privát és publikus hálózatot, illetve a vírus és támadási védelem beállításnál be lenne kapcsolva a „real-time”, akkor automatikusan jelezne a szoftver, hogy gyanús fájt, objektumot talált, majd észlelés után rögtön ki is törölné. Teszteléskor kíváncsiságból először a szoftver hatékonyságát. A „real-time”-ot nem állítottam át, majd megpróbáltam kicsomagolni egy tetszőleges kártékony programot, de a Defender hatásos volt, és azonnal jelezte és törölte a fájlt. A tesztelés előtti utolsó lépés a „snapshot” vagy másnéven

a pillanatkép készítése volt. Ez azért fontos, mert ha egy rosszindulatú program sikeresen lefut, akkor vissza tudom állítani a tesztelés előtt változatra. Ennek köszönhetően gyorsabb munkát tudok biztosítani minden tesztelés során, illetve jobb képet kapok a vizsgálat során.

Minden egyes malware lefutásakor megfigyeltem a háttérben futó folyamatokat, alkalmazásokat. A legtöbb virtuális környezetet észlelő rosszindulatú programoknál a legtöbb lefutott, viszont amikor észlelte a virtuális teret, akkor eltűntette magát.

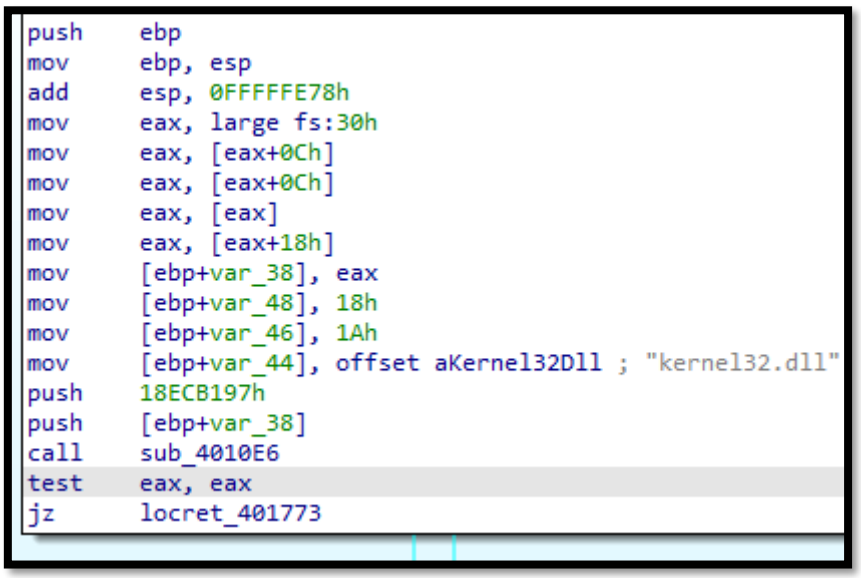
Az észlelés azért volt sikeres, mert az API hívásokat használták ki nagy valószínűséggel. Sajnos az API hooking-ot nem tudtam megvalósítani, mert rendszerprogramozási feladat, ami kívül esik az eddigi tanulmányaimtól. Egy elég komplex megoldási technika, amely magasfokú assembly kódolást, illetve további mély tudást igényel. Az alap reakciókat összegeztem egy diagram segítségével. (Ábra 13)

Ábra 13: Vizsgálati eredmény

A malwarek elemzésekor az alap tulajdonságokat figyeltem meg. Például tudtam, hogy a Cerber az API hívásokat figyeli, hallgatja le először, és a detektálás után eltűnteti magát, ha virtuális környezetet észlel. A háttérfolyamatokat megfigyeltem, és jól látható volt, hogy a „RansomwareCerber.exe” futtatásakor a háttérben, a kernel32dll fájlt is nagy mértékben futott, melynél feltételezhető volt, hogy a már felvázolt módszerrel detektálja a virtuális gépet. A művelet után a rosszindulatú program eltűntette, kitörölte magát. A további vizsgált alany is hasonlóan viselkedett, kivéve a RansomwarePetya és a RansomwareWannaCry, amik csak sima kártevők. A vizsgált RansomwareLocky egy VME detektálási tulajdonsággal rendelkezik, viszont nem tűntette el magát, ha nem jóindulatú programként viselkedett. [5]

Érdekeségből az IDA szoftver segítségével megfigyeltem a „RansomwareCerber.exe” fájl teljes forráskódját, hogy még jobban átlássam az VME detektáló kártevők tulajdonságát.

Az ábrán jól kivehető a már említett kernel32.ddl használata, hogy meg tudja a kártékony program, hogy virtuális környezetben van-e vagy sem. (Ábra 14)



Ábra 14: RansomwareCerber vizsgálata IDA segítségével

A többi sima kártevő leginkább zsarolóvírus volt, így egy látványos viselkedést tapasztalhattam meg, ami a fájlok vagy az operációs rendszer teljes titkosítása, a felhasználó kizárása volt. Teszteléskor egy jó tanulság, illetve élmény volt megfigyelni a kártevők pontos működését, reakcióját.

6.4 A teszt összesítése

Az malware elemzés előtti előkészületnél a VirtualBox-on könnyebben be lehetett állítani a paramétereket. Leginkább a hálózati adapterek beállításánál nyilvánult meg. A VirtualBox-nál van olyan beállítási lehetőség, hogy semmilyen adapter ne tartalmazzon. A VMware-nél ez az opció nincs, így el kell távolítani a teljes hálózati adaptert, hogy biztosítsuk a biztonságos teszt környezetet. Ha nagyobb tudásom és tapasztalatom lenne malware elemzéssel kapcsolatban, akkor a hálózati beállításokat lehetne másként is megoldani, hogy még jobban kimutatkozzon a malwarek viselkedése. Viszont kezdőként mindenképpen csak hálózat kikapcsolással futtattam le a kártevőket, hogy óvjam a fizikai gépemet a különböző fertőzésektől.

Egy táblázatban összesítettem a vizsgálat során elvégzett műveleteket.
A VirtualBox és a VMware szoftvereket az esettanulmány folyamán jobban megismertem. A teszt végén a két szoftver között levontam az előnyöket, illetve a hátrányokat.

Tesztelt szoftverek összesített eredménye:

	Alapértelmezett beállítás	**Anti-detektálási megoldások**
VirtualBox		
	Alapértelmezett MAC-cím: 08-00-27-B1-1D-5A	MAC-cím módosítása: E0-3E-44-83-65-71
	Alapértelmezett adapter: NAT	Hálózati kártya letiltása, USB adapter használata
	Memória méret: 2048MB	Memória méret növelése: 5120MB
	CPU száma: 1	CPU szám növelése: 4
	Registry: pl.: disk drives: VBOX HARDDISK ATA Device	Registry értékek átírása: disk drives: Samsung SSD 870 EVO 500GB
	Segédszoftver: Guest Additions	Szoftver törlése
VMware Workstation Pro		
	Alapértelmezett MAC-cím: 00-0C-29-83-65-79	MAC-cím módosítása: E0-3E-44-83-65-71
	Alapértelmezett adapter: NAT	Hálózati kártya letiltása, USB adapter használata
	Memória méret: 2048MB	Memória méret növelése: 4096MB
	CPU száma: 1	CPU szám növelése: 4
	Registry: pl.: disk drives: VMware Virtual NVMe Disk	Értékek módosítása, pl.: disk drives: Samsung SSD 870 EVO 500GB
	Segédszoftver: vmtools	Szoftver törlése és elrejtése

Ábra 15: Összesített táblázat

Egy kutatás során kimutatták, hogy több kártékonykód lefutásakor, milyen detektálási technikákat alkalmaztak.

Anti-vm detektálási módszereket összesítve a VirtualBox-nál jobban ki tudtam alakítani az elvárt környezetet, mivel a hardveres tulajdonságait tekintve jobban átalakítható volt. A VirtualBox segédprogramja a Guest Additions letiltása után nem futott a szoftver, míg a VMware-nél a VMtools futott a háttérben.

A különböző bejegyzések, hivatkozások módosításakor egyik szoftvernél sem volt különbség, mindkettő támogatta azoknak a módosítását. Viszont mélyen nyúló adatbázisban lévő virtualizációra utaló értékeket nem engedte átírni, amik a „Computer\HKEY_LOCAL_MACHINE\SYSTEM\DriverDatabase\DriverPackages*”-ben voltak.

A vizsgált szoftvereknél meglepődtem, hogy végül a VirtualBox-ot tartottam egy kicsivel jobb szoftvernek ehhez a feladathoz, mivel a VMware-t jobban ismertem, illetve felhasználóbarátabb funkciói voltak, mint a VirtualBox-nak.

Érdemes kipróbálni olyan készen megírt megoldásokat, amik például megtalálhatóak a Github-on. Ilyen például egy olyan fortély, aminél a készítő, olyan DLL fájlt írt, ahol kifejezetten VMware utaló rendszert leíró értékeket, folyamatokat, szolgáltatásokat és fájlokat rejtett el. A már említett API hookingot alkalmazta. Lényegében az „AppInit_DLL”-ek registry értéket be kell hívni a felhasználónak az elérési útját, és utána minden folyamat ami betölti a user32.dll-t, be fogja tölteni ezt a kreált „.dll” fájlt is. Ami alapján eltudja rejteni a virtuális környezetet, mert hamis értékeket fog visszaadni a detektáló malware számára. [29]

Több ilyen jellegű megírt programok, szoftverek találhatóak meg, viszont a nagy hibájuk, hogy gyorsan változik a technológia, illetve vele együtt a kártevők is. Így nem mondható a legjobb megoldásnak, mert naprakészek ezek a programok.

7 Konklúzió

A szakdolgozatban sikeresen bemutattam a VME detektáló malwarek működését, és az ellenük irányuló védekezés mentét, megoldását a lehetőségekhez mérten. A tesztelt virtualizációs szoftverek átkonfigurálásával jól lehet biztosítani egy virtuális gép védelmét a virtualizációt detektáló tulajdonsággal rendelkező kártevők ellen. A vizsgálat végén kiderült, hogy a teszt során kiválasztott VME detektáló kártékony programok az API hívásokat figyelik meg, mert a hívások által több információt tudnak felderíteni a környezetről, így el tudják dönteni, hogy virtuális környezetben futnak-e vagy sem. További vizsgálatot érdemel az esettanulmány, mivel a biztonságos virtuális környezet kialakításához szükséges lenne elhárítani a kártevők által kért API hívásokat. Ezt egy olyan programmal lehetne megvalósítani, ami hívásokkor hamis értékeket adna vissza a kártevőnek. Ez úgy kivitelezhető, hogy a megírt fájl befecskendezi a malware DLL fájlját, és ennek köszönhetően a kártevőt hatástalanítani fogja. Valamint, érdemes a környezet tesztelésekor nagyobb mennyiségű mintával, azaz nagyobb mennyiségű kártevőkkel dolgozni, hogy pontosabb eredményt kapjuk a vizsgálat folyamán. Ez azért hasznos, mert így jobban lehetne védeni a virtuális környezetet, illetve a kártevők viselkedésének az elemzését és kutatását is megkönnyítené ez a módszer. A mai világban egyre jobban terjed a virtualizáció, ami miatt még több figyelmet érdemelne a téma.

8 Conclusion

In the thesis, I have successfully demonstrated the operation of VME detecting malwares and the way to defend against them as far as possible. By reconfiguring the tested virtualization software, it is possible to provide a good protection of a virtual machine against malware with virtualization detecting property. At the end of the test, it was found that the VME detecting malware selected in the test listen to API calls, because they can discover more information about the environment through the calls, so they can decide whether they are running in a virtual environment or not. The case study deserves further investigation, as it would be necessary to block API calls requested by malware to create a secure virtual environment. This could be achieved by a program that would return invalid handle to the malware when calls are made. This could be done by injecting the written file into the malware DLL file, which would disable the malware. It is advisable to work with a larger sample size when testing the environment, for instance, a larger number of malware samples, to get more

accurate results during the testing. This is useful, because it would allow better protection of the virtual environment and would also facilitate analysis and research into the behaviour of the pests. In today's world, virtualization is becoming more and more widespread, which would make this topic worthy of even more attention.

Irodalomjegyzék

[1] T. L. Ed Skoudis, „On the Cutting Edge:Thwarting Virtual Machine,” 2006.

[2] Mitre, „CVE Mitre,” 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-3941. [Hozzáférés dátuma: 1 09 2021].

[3] Mitre, „CVE Mitre,” 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-21989. [Hozzáférés dátuma: 22 11 2021].

[4] Mitre, „CVE Mitre,” 2019. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-16406. [Hozzáférés dátuma: 4 11 2021].

[5] „CVEDETAILS,” 2021. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-20406/Oracle-Vm-Virtualbox.html.

[6] N. NIST, „NIST,” 2020. [Online]. Available: https://www.nccoe.nist.gov/projects/implementing-zero-trust-architecture. [Hozzáférés dátuma: 04 11 2021].

[7] NIST, „NIST,” 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf. [Hozzáférés dátuma: 7 11 2021].

[8] V. T. T. H. E. Lampis Alevizos, „Wiley Online Library,” 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/spy2.191#spy2191-tbl-0003. [Hozzáférés dátuma: 23 11 2021].

[9] R. B. Jakub Szefer, „Architectural support for hypervisor-secure virtualization,” 2012.

[10] M. B. a. C. R. M. Brengel, „Detecting Hardware -Assisted Virtualization,” 2016.

[11] Y. Assor, „Cyberbit,” 2018. [Online]. Available: https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained/. [Hozzáférés dátuma: 17 11 2021].

[12] T. A. Abdurrahman Pekta, „A dynamic malware analyzer against virtual machineaware malicious software,” 2013.

[13] O. D. Yasir Shoaib, „Pouring Cloud Virtualization Security Inside Out,” 2014.

[14] J. S. Reube, „A survey on virtual machine security,” Helsinki University of Technology, 2007.

[15] S. Vilkomir-Preisman, „Deepinstinct,” 2019. [Online]. Available: https://www.deepinstinct.com/2019/10/29/malware-evasion-techniques-part-2-anti-vm-blog/.

[16] O. S. Saydjar, „Hiding Virtualization from Attackers and Malware,” 2007.

[17] A. C. A. Rashid, „Virtualization and its Role in Cloud Computing Environment,” 2019.

[18] H. Team, „HND,” [Online]. Available: https://hackernewsdog.com/vm-bypassing-escaping-infect-host-machine/.

[19] A. Moore, „Research Note - Banking Malware Explained: the Case of Dridex,” 2016.

[20] R. B. a. H. Trivedi, „Malware in spam email: Risks and trends in the Australian Spam Intelligence Database,” 2020.

[21] M. Att&ck, „Mitre Att&ck,” 2019. [Online]. Available: https://attack.mitre.org/techniques/T1497/. [Hozzáférés dátuma: 11 04 2021].

[22] M. K. Stubman, „Improsec-User-mode API hooks and bypasses,” 2021. [Online]. Available: https://improsec.com/tech-blog/user-mode-api-hooks-and-bypasses. [Hozzáférés dátuma: 11 10 2021].

[23] P. Kemkes, „GdataSoftware,” 2020. [Online]. Available: https://www.gdatasoftware.com/blog/2020/05/36068-current-use-of-virtual-machine-detection-methods. [Hozzáférés dátuma: 2 10 2021].

[24] M. M. F.-U.-A. R. D. A. A.-G. M. S. G. G. Andrew Casea, „ScienceDirect-HookTracer: A System for Automated and Accessible API Hooks Analysis,” 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287619301604. [Hozzáférés dátuma: 13 11 2021].

[25] H. S. S. K. Sanchit Gupt, „Malware Characterization Using Windows API Call Sequences,” 2018.

[26] nybble04, „Shady-Hook,” 2018. [Online]. Available: https://github.com/nybble04/Shady-Hook. [Hozzáférés dátuma: 1 09 2021].

[27] Blueliv, „Playing with GuLoader Anti-VM techniques,” 2020.

[28] ytisf, „Github,” 2021. [Online]. Available: https://github.com/ytisf/theZoo. [Hozzáférés dátuma: 8 07 2021].

[29] rsumner33, „Github,” 2018. [Online]. Available: https://github.com/rsumner33/anti-anti-vm-detection-dll. [Hozzáférés dátuma: 21 11 2021].

[30] I. E. IBM, „IBM,” 2019. [Online]. Available: https://www.ibm.com/cloud/learn/hypervisors.

[31] C. Wueest, „Threats to virtual environments,” 2014.

[32] VMware, „VMware,” 2021-06-22. [Online]. Available: https://www.vmware.com/security/advisories/VMSA-2021-0013.html.

[33] Mitre, „CVEMitre,” 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-3941.

[34] C. H. L. D. a. W. J. Ping Chen, „Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in use of anti-debugging and anti-VM techniques in,” 2016.

Rövidítésjegyzék

- VME: Virtual Machine Environment, magyarul: virtuális gépi környezet
- vm: virtual machine, azaz virtuális gép
- malware: rosszindulatú/kártékony kódú program
- OS: Operating System, operációs rendszer
- Patch: javítják, foltozzák az adott terméket, hogy naprakész legyen
- Host: gazdagép, azaz a fizikai gép
- Guest: vendéggép
- Hypervisor: A szoftver, mely segítségével a fizikai és a virtuális gépet megfelelően lehet elszigetelni

- API hooking: alacsony szintű technika, amelynél az adott szoftverkomponens viselkedését lehet manipulálni.

Ábrajegyzék

Melléklet

A szakdolgozatban kész környezetet, virtuális gépeket a biztonság érdekében ezen az elérhetőségen lehet kérvényezni.

Elérhetőség: emi.bohm@gmail.com