Tampereen ammattikorkeakoulu

Haittaohjelmien Analyysi

Vili Eskelinen

OPINNÄYTETYÖ
Huhtikuu 2022

Tieto- ja viestintätekniikka
Tietoliikennetekniikka ja tietoverkot

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma
Tietoliikennetekniikka ja tietoverkot

ESKELINEN, VILI:
Haittaohjelmien Analyysi

Opinnäytetyö 103 sivua, joista liitteitä 9 sivua
Huhtikuu 2022

Opinnäytetyössä kerättiin tietoa haittaohjelmista, haittaohjelma-analyysista ja erityisesti muistianalyysista, sekä suoritettiin haittaohjelma-analyysia virtuaalisessa laboratorioympäristössä oikeilla haittaohjelmanäytteillä.

Opinnäyteyön teoreettinen osuus käsittelee haittaohjelmia, niiden toimintaa ja lyhyesti niiden vaikutusta yritystoimintaan ja yhteiskuntaan tänä päivänä. Enimmäkseen opinnäytetyössä keskitytään haittaohjelmien analyysin teoriaan ja eri analyysimetodeihin sekä tapoihin, joilla haittaohjelmien kirjoittajat yrittävät välttää haitallisten ohjelmien manuaalisia- ja automaattisia havainnointimetodeja.

Työn empiirisessä osiossa rakennettiin virtuaalinen analyysiympäristö kahdella keskenään verkostoiduilla virtuaalikoneella käyttäen Virtualbox-ohjelmaa. Toista virtuaalikonetta käytettiin simuloimaan tiettyjä verkkopalveluita ja tarkastelemaan verkkoliikennettä sekä muistianalyysin suorittamiseen Volatility-ohjelmalla. Toista virtuaalikonetta käytettiin perinteisten, staattisten ja dynaamisten analyysimetodien suorittamiseen. Virtuaalikoneet käyttivät Linux- ja Windows 10 -käyttöjärjestelmiä vastaavasti.

Opinnäytetyön tuloksena on kokoelma tietoa haittaohjelmista ja haittaohjelma-analyysistä, jonka tulisi antaa lukijalle yleisymmärrys haittaohjelmista ja eri metodeista, joilla niitä analysoidaan. Opinnäyteyön sisältämä tieto kerättiin enimmäkseen aiheeseen liittyvästä kirjallisuudesta.

Asiasanat: haittaohjelma-analyysi, muistianalyysi, virtuaalikone, volatility

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Telecommunications and Networks

ESKELINEN, VILI:
Malware Analysis

Bachelor's thesis 103 pages, appendices 9 pages
April 2022

The purpose of this thesis was to gather information on malware, malware analysis, and in particular, malware analysis by means of memory forensics. In addition, the goal was to conduct said analysis in a virtual environment on actual samples of real-life malware.

The theoretical section of this thesis contains information about malware in general, its functionality, and in brief, the effects of its use on business and society today, focusing mostly on malware analysis and the methods malware authors use to avoid manual and automatic detection.

A virtual malware analysis lab was built with Virtualbox, using two networked virtual machines. One of the virtual machines was used to simulate certain network services, and to capture network traffic, while the other was used to analyse the malware using traditional static and dynamic methods. The virtual machines used Linux and Windows 10 operating systems respectively. The memory analysis was conducted with Volatility using the Linux virtual machine.

The result of this thesis is a collection of information about malware and malware analysis that should give the reader a general understanding of malware and the different methods used to analyse it. The theoretical information in this thesis was mostly gathered from literature on the subject.

Key words: malware analysis, memory forensics, virtual machine, volatility

SISÄLLYS

ERITYISSANASTO

API	*Application Programming Interface*, eli ohjelmointirajapinta mahdollistaa eri ohjelmien välisen keskustelun.
Bottiverkko	Bottiverkko on verkko, joka koostuu tartunnan saaneista tietokoneista, jotka ovat hyökkääjän hallinnassa. Tartunnan saaneita tietokoneita kutstutaan ”zombieiksi”
DLL	Lyhenne sanoista *Dynamic-Link Library*, jota käytetään tiedostoista, jotka sisältävät yleisessä käytössä olevaa koodia ja funktiota. Tiedostoilla on .dll-tiedostopääte.
Github	*Git*-versionhallintaohjelmaa käyttäville projekteille tarkoitettu verkkosivu.
HAL	Lyhenne sanoista *Hardware Abstraction Layer*, suomennettuna ”laitteiston abstraktiokerros”, on käyttöjärjestelmän osa, joka tarjoaa järjestelmän ohjelmille rutiineja, joiden avulla ohjelmat voivat käyttää järjestelmään liitettyä laitteistoa.
I/O	Lyhenne sanoista *Input/Output*, suomennetaan ”siirrän-tä”.
IOC	Lyhenne sanoista *Indicator of Compromise*. Termillä viitataan information, jolla voidaan tunnistaa haittaohjelma, ja todeta järjestelmän saaneen tartunnan.
Kerberos	*Active Directory*:n käyttämä todennusprotokolla.
NTLM	Microsoft:n kehittämä turvallisuusprotokollapaketti.

Obfuskointi	Käännös englannin kielen sanasta ”obfuscation”, joka viittaa toimintaan, jossa tärkeää tietoa yritetään piilotella, tekemällä siitä vaikeasti ymmärrettävää.
PE	Lyhenne sanoista *Portable Executable*. Windowsin käyttämä tiedostoformaatti suoritettaville tiedostoille.
PUA	Lyhenne sanoista *Possibly Unwanted Application*, eli “mahdollisesti ei-haluttu applikaatio”. Termillä luokitellaan tiettyjä haitallisia ohjelmia.
Repo	Lyhenne sanasta *Repository*, eli tietovarasto. Termillä viitataan yleisesti Github-palvelun sisältämiin tietovarastoihin.
Skripti	Käännös englannin kielen sanasta ”script”, joka tietotekniikassa viittaa suoritettavaan tiedostoon, joka pitää sisällään jonkun komentotulkin (esim. Windowsissa CMD tai powershell) komentosarjoja. Skripti suorittaa automaattisesti kaikki sen sisältämät komennot, kun itse skripti suoritetaan.
Tor	Lyhenne sanoista *The Onion Router.* Ohjelmisto, joka mahdollistaa Internetin, pimeän verkon, sekä piilopalveluiden (engl. ”*hidden service*”) anonyymin käytön.

1 JOHDANTO

Nykypäivänä Internet ja muiden tiedonsiirtojärjestelmät ovat erittäin suuressa osassa yritysten- ja myös yhteiskunnan toimintaa. Tämän myötä myös erilaiset kyberhyökkäykset näihin kohteisiin kasvaa jatkuvasti. Osissa näistä hyökkäyksistä hyökkääjät käyttävät haittaohjelmia saavuttaakseen tavoitteensa, ja siksi niiden analysointi on kriittisessä osassa puolustuksien rakentamisessa, sekä kyberhyökkäysten tutkimisessa. Tässä opinnäyteyössä tarkastellaan haittaohjelmia ja analysoidaan niitä omassa virtuaalisessa laboratorioympäristössä, ja keskitytään hieman tarkemmin muistianalyysiin.

Työn tavoitteena on kerätä tietoa haittaohjelmista, niiden toiminnasta ja vaikutuksesta nykymaailmassa, niiltä puolustautumiselta, sekä siitä, miten niitä analysoidaan eri metodein. Työn teoriaosuuden pohjalta on tarkoitus rakentaa oma virtuaalinen analyysiympäristö virtuaalikoneiden avulla, jossa suoritetaan muutamia esimerkkianalyysejä oikeilla haittaohjelmanäytteillä. Erityisesti työn analysointiosuudessa on tarkoitus keskittyä muistianalyysiin Volatility-ohjelman avulla, jolla voidaan tarkastella järjestelmän suoritushetkistä tilaa, analysoimalla sen keskusmuistista otettua kaappausta. Työn lopputuloksena olisi ideaalisesti tietopaketti, jonka avulla itse kirjoittaja, sekä lukija saisi hyvän yleisymmärryksen haittaohjelmista ja niiden analysoinnista.

Aihe valittiin kirjoittajan kiinnostuksesta kyberturvallisuuteen, mutta itse haittaohjelmat, ja varisinkin niiden analyysi ovat aiheina enimmäkseen tuntemattomia.

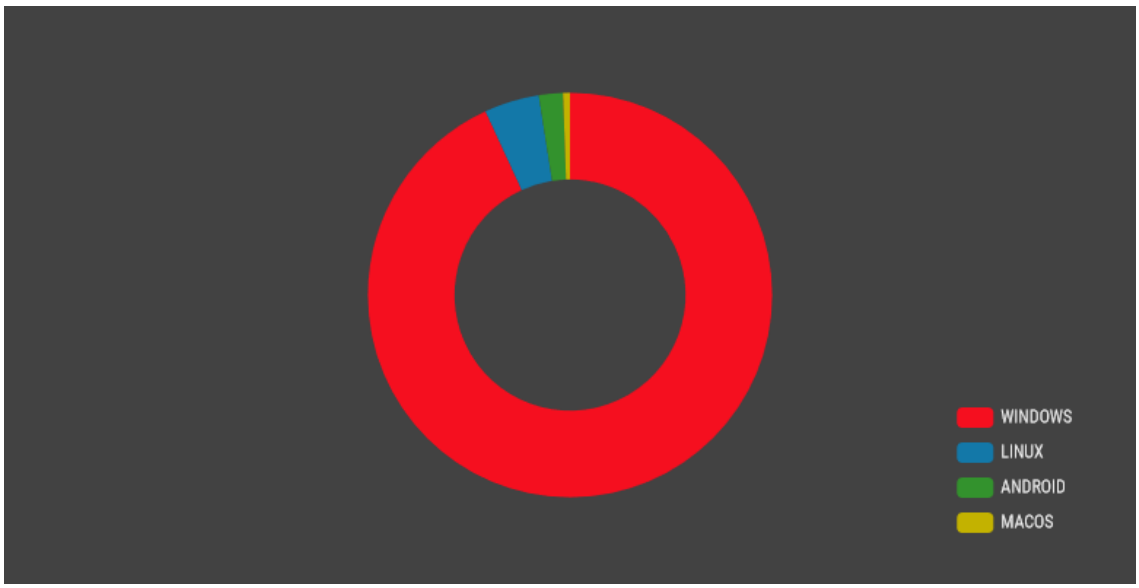
2 HAITTAOHJELMAT

Haittaohjelma voidaan yleisesti määritellä minä tahansa ohjelmana, joka on suunniteltu aiheuttamaan vahinkoa yksittäiselle tietokoneelle tai tietoverkolle (Moir 2009). Termi on kovin laaja ja haittaohjelmia on monenlaisia, moneen eri tarkoitukseen. Haittaohjelmat saattavat mm. häiritä järjestelmän toimintaa, varastaa arkaluontoista tietoa, vakoilla tartunnan saaneiden järjestelmien käyttäjiä, salata tiedostoja ja käyttää niitä kiristykseen tai liittää tietokoneen osaksi bottiverkkoa. (Monnappa 2018, 1.)

Useimmiten haittaohjelman on tarkoitus pysyä huomaamattomissa esimerkiksi esittämällä toista ohjelmaa, poistamalla järjestelmän lokitiedostoja, tai käyttämällä koodin obfuskointia, joka hankaloittaa haitallisen koodin havaitsemista ja ohjelman analysointia.

AV-TEST organisaation ylläpitämän AV-ATLAS uhkatietoalustan mukaan suurin osa haittaohjelmista kirjoitetaan Windows-käyttöjärjestelmälle. Alusta kerää tietoa ja luo statistiikkaa haittaohjelmista ja ns. PUA:sta (Possibly Unwanted Application), ja vuonna 2022 raportoitu, aikojen saatossa kerättyjen näytteiden kokonaismäärä oli 3 609 609 kappaletta, joista lähes 3,4 milj. kappaletta, eli n. 94 % oli kirjoitettu Windows-käyttöjärjestelmälle. Linuxille kirjoitettuja näytteitä oli n. 160 000 ja Androidille kirjoitettuja n. 70 000. (AV-ATLAS 2022.) Statistiikka on esitetty kuviossa 1.

Työssä keskitytään enimmäkseen Windows käyttöjärjestelmään ja sille kirjoitettuihin haittaohjelmiin, tosin samat periaatteet haittaohjelmien toiminnan ja niiden analyysin kannalta pätevät yleisesti muihinkin käyttöjärjestelmiin.



KUVIO 1. Haittaohjelmanäytteet käyttöjärjestelmän mukaan (AV-ATLAS 2022).

2.1 Luokittelu

Haittaohjelmat luokitellaan eri tyyppeihin toiminnallisuuksien ja leviämistapojen mukaan. Alla on lista yleisimmistä haittaohjelmatyypeistä tunnetun tietoturvayrityksen Malwarebytes Inc.:n mukaan.

Adware haittaohjelmat ovat suunniteltu esittämään käyttäjän näytölle mainoksia, yleensä verkkoselaimen avulla. Mainokset tuottavat haittaohjelman kehittäjälle rahallista tuloa. (Malwarebytes n.d. a)

Spyware eli vakoiluohjelmat nimensä mukaisesti vakoilevat tartunnan saaneen tietokoneen käyttäjää esimerkiksi lähettämällä web kameran tallentamaa kuvaa, selaustietoja, näppäinpainalluksia tai muuta käyttäjään liittyvää dataa hyökkääjälle arkaluontoisen tiedon toivossa. Näppäinpainalluksia tallentavia haittaohjelmia kutsutaan keyloggereiksi. (Malwarebytes n.d. a)

Virus on haittaohjelma, jonka on tarkoitus luoda kopioita itsestään tartunnan saaneessa järjestelmässä muokkaamalla toisten ohjelmien koodia ja siten levitä myös toisiin järjestelmiin. Virus voi aiheuttaa haittaa esimerkiksi tuhoamalla dataa. Virus vaatii jonkinlaisen toimenpiteen käyttäjältä levitäkseen. (Malwarebytes n.d. a)

Worm eli mato, on samantyylinen haittaohjelma kuin virus, mutta se voi levitä itsenäisesti, ilman käyttäjää. (Malwarebytes n.d. a)

Trojan saa nimityksensä kreikkalaisessa mytologiassa esiintyvästä Troijan puuhevosesta. Troijalainen esittää jotakin käyttäjälle hyödyllistä ohjelmaa ja asennuttuaan suorittaa hyökkääjän haluavan tehtävän. Troijalainen saattaa suorittaa monia eri toimintoja, kuten järjestelmän hallinta, vakoilu, datan varastaminen tai toisten haittaohjelmien lataaminen. Teknisesti troijalainen siis suorittaa muille haittaohjelmille ominaisia toimenpiteitä ja haittaohjelmaa voidaan kutsua troijalaiseksi silloin, jos se esittää jotain muuta hyödyllistä ohjelmaa, ns. ”lahjahevosta”. (Frotinet n.d.) Troijalaista, joka antaa hyökkääjälle etäyhteyden tietokoneeseen yleensä kutsutaan nimellä Remote Access Trojan tai RAT.

Ransomware eli kiristyshaittaohjelma on suunniteltu estämään tartunnan saaneen järjestelmän käyttö kokonaan tai osittain, yleisesti salaamalla käyttäjän tiedostot. Täten hyökkääjä voi pyytää uhrilta lunnaita saadakseen pääsyn järjestelmään tai tiedostoihinsa. Lunnaita pyydetään kryptovaluutan muodossa, jäljittämisen vaikeuden vuoksi. (Malwarebytes n.d. a)

Tietoturvayhtiö Sophos ilmoitti ”*The State of Ransomware 2021”* raportissaan, että 5000:n kyselyyn vastanneen yrityksen joukosta 37 % oli ollut jonkinlaisen kyberhyökkäyksen kohteena, jossa käytettiin kiristyshaittaohjelmaa. Raportissa kerrotaan myös, että 96 % hyökkäyksen kohteena olleista sai datansa takaisin, joista 32 % maksoivat hyökkääjille lunnaita. (Sophos 2021, s. 3,9.) Kiristyshaittaohjelmat ovat siis selkeästi suuressa osassa varsinkin yrityksiin kohdistuvissa kyberhyökkäyksissä.

Rootkit on haittaohjelma, joka on suunniteltu antamaan hyökkääjälle hallinnan tartunnan saaneesta järjestelmästä korotetuilla oikeuksilla, ts. ”root” tai järjestelmän valvojan oikeuksilla ja samalla pysymään huomaamattomana toisilta ohjelmilta ja itse käyttöjärjestelmältä. (Malwarebytes n.d. a)

Malicious cryptominer tarkoittaa ohjelmaa, joka käyttää tietokoneen resursseja kryptovaluutan louhimiseen ja lähettää louhinnalla tuotetun valuutan hyökkääjän lompakkoon. (Malwarebytes n.d. a)

Exploit eli haittakoodi luokituksella tarkoitetaan koodia tai syötettä, joka käyttää hyväksi järjestelmässä tai toisissa ohjelmissa olevia bugeja ja haavoittuvuuksia antaakseen hyökkääjälle mahdollisuuden suorittaa koodia mielivaltaisesti tai saadakseen hallinnan järjestelmästä. (Malwarebytes n.d. a) Esimerkiksi buffer overflow attack, eli puskurin ylivuotohyökkäys kuuluu tähän kategoriaan. Exploit -koodi ei itsessään ole haitallista, mutta ne suorittavat ei-haluttuja toimenpiteitä ja niiden käyttö hyökkäyksissä yleisesti johtaa haitallisen koodin suorittamiseen.

Zero-day exploit viittaa menetelmään, jossa hyväksikäytetään haavoittuvuutta minkä olemassaolosta ei aikaisemmin tiedetty, ja jolle ei ole vielä olemassa korjausta. Kun haavoittuvuus tulee julkiseen tietoon, se ei teknisesti ole enää nollapäivähaavoittuvuus. (Malwarebytes n.d b).

Puskurin ylivuotohyökkäyksessä hyökkääjä kykenee ylikirjoittamaan suoritettavan haavoittuvuuden sisältävän ohjelman muistialuetta, siten että ohjelma suorittaa hyökkääjän sille syöttämää haitallista koodia (Owasp n.d).

Sikorskin ja Honig:n (2012) mukaan mikä tahansa yksittäinen haittaohjelma saattaa sisältää useamman eri luokituksen piirteitä tai toimintoja ja niitä luokitellaan myös hyökkääjän tarkoitetun kohteen perusteella.

Yleishaittaohjelmien tarkoitus on levitä mahdollisimman useaan järjestelmään. Nämä ohjelmat ovat yleisesti yksinkertaisempia ja helpompia havaita jo ennen vahinkoja levinneisyytensä vuoksi. (Sikorski & Honig 2012, 0.)

Kohdistettu haittaohjelma on suunniteltu toiminnallisuutensa ja hyökkäysvektorinsa kohdilta erityisesti jotakin yksittäistä kohdetta, yleensä yhtiötä varten. Kohdistettuja haittaohjelmia on hankalampi estää varsinkin automatisoiduin puolustuskeinoin ja siten ne yleensä myös leviävät verkoissa, joissa jokin tietokone on saanut tartunnan. Nämä ohjelmat ovat puolestaan usein monimutkaisia ja niiden poistaminen, sekä verkon haavoittuvuuksien korjaaminen vaatii lähes aina tarkempaa analyysiä. (Sikorski & Honig 2012, 0.)

2.2 Leviäminen

Ollakseen hyökkääjälle hyödyllinen, haittaohjelman tulee päästä sille määrätyn kohteen muistiin. Hyökkääjän tarvitsee siis tavan levittää haittaohjelmaansa. Haittaohjelmat tyypillisesti leviävät kolmella eri tavalla; fyysisesti, internetin välityksellä tai sähköpostin mukana. Social engineering, eli käyttäjän manipulointi on isossa osassa haittaohjelmien levityksessä. (Mohanta & Saldanha 2020, 3.)

Termillä ”point of entry” usein kuvaillaan tapaa, jonka kautta haittaohjelma on päässyt tartuttamaan järjestelmän.

2.2.1 Fyysiset leviämiskeinot

Fyysisillä leviämiskeinoilla tarkoitetaan USB-muistitikkuja, kovalevyjä tai muuta massamuistia mitä saatetaan siirtää useampien järjestelmien välillä (Mohanta & Saldanha 2020, kpl 3). Esimerkiksi vuonna 2010 Iranin teollisuusjärjestelmiä kohdistanut Stuxnet-mato levisi tartunnan sisältäneeltä USB-tikulta internetistä irrotetun tietokoneen massamuistiin ja loi itsestään kopion muihin tietokoneeseen kytkettyihin USB-tikkuihin (F-Secure n.d).

2.2.2 Leviäminen internetin välityksellä

Hyökkääjä voi levittää haittaohjelmia internetin välityksellä verkkosivujen avulla. Hyökkääjän omistama verkkosivu saattaa mainostaa tarjoavansa päivityksiä johonkin ohjelmaan tai yrittää muuten saada käyttäjän lataamaan sivulta jotakin haitallista koodia sisältävää. Haittaohjelmia voidaan myös ladata sivustoille, joissa jaetaan ohjelmistoja. (Mohanta & Saldanha 2020, 3.)

Joissain tapauksissa, missä muutoin oikeudenmukainen ja aito verkkosivusto sisältää haavoittuvuuden, haittaohjelma voi levitä ilman toimenpidettä käyttäjältä, pelkkä sivulla vierailu riittää. Hyökkääjä asentaa haavoittuvuuden sisältävälle sivustolle ”exploit kit”:ksi kutsutun haitallisen komponentin, joka löytää haavoittuvuuksia sivulla vierailevan käyttäjän järjestelmästä ja käyttää sitä hyväkseen ladatakseen kohteeseen muita haittaohjelmia. Näitä tapauksia kutsutaan ”drive by download”-nimityksellä. (Kaspersky n.d.)

2.2.3 Leviäminen sähköpostin välityksellä

Sähköposti on vanhin ja silti laajasti käytössä oleva tapa levittää haittaohjelmia. Haitallinen koodi sisällytetään liitetiedostoon tai linkkiin, jonka hyökkääjä yrittää saada vastaanottajan avaamaan, manipuloimalla tätä. Viesti saattaa näyttää tulevan pomolta tai kollegalta, se saattaa ilmoittaa myöhästyneestä laskusta tai voitosta kilpailussa. Ideana on kuitenkin saada käyttäjä klikkaamaan sähköpostissa olevia haitallisia linkkejä tai lataamaan haittaohjelman sisältävän liitetiedoston. (Mohanta & Saldanha 2020, 3.)

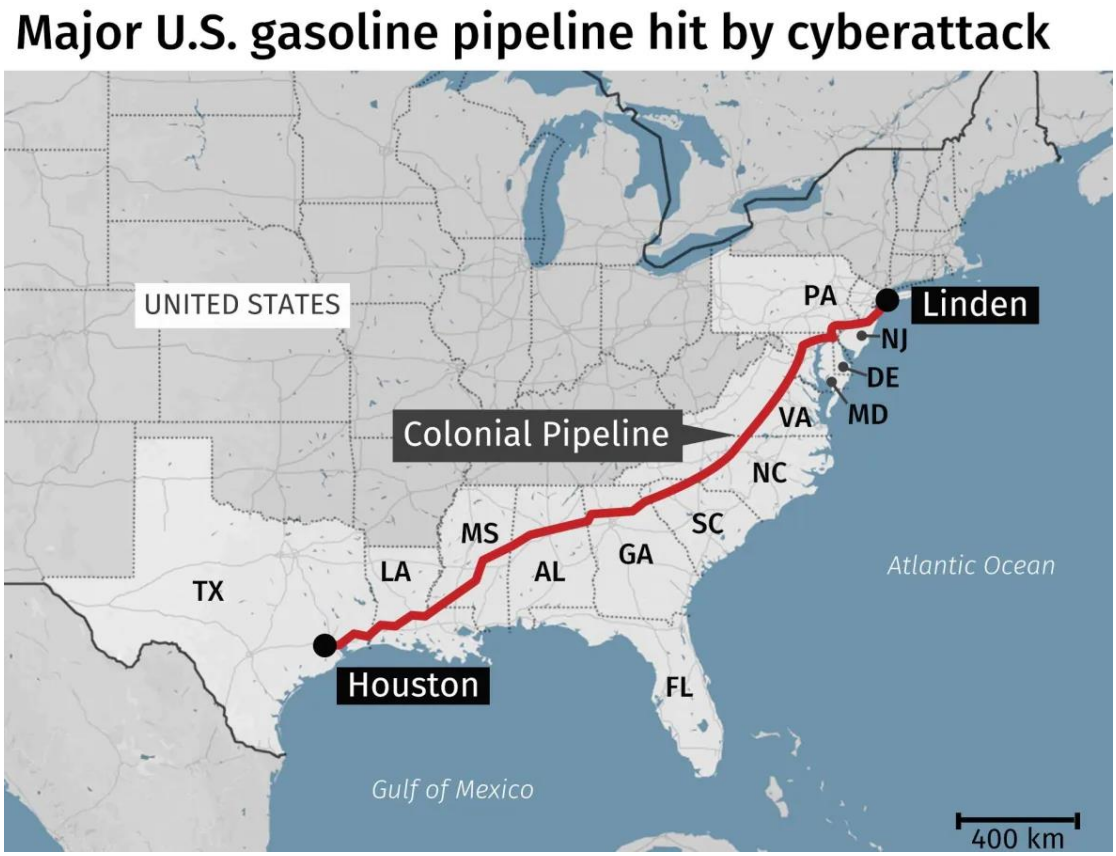
2.3 Esimerkkitapauksia ja haittaohjelmien vaikutus

Vaikka omien kuvien tai muiden tiedostojen menettäminen kiristyshaittaohjelmalle ei saata kuulostaa vakavalta isomassa skaalassa, kriittisiin järjestelmiin iskiessä haittaohjelmilla voi olla suurikin vaikutus.

2.3.1 Colonial Pipeline Company, DarkSide

Toukokuussa 2021 Yhdysvaltalainen Colonial Pipeline Company joutui kyberhyökkäyksen kohteeksi, jossa käytettiin DarkSide-kiristyshaittaohjelmaa. Yrityksen omistama putkijärjestelmä siirtää noin 45 % Yhdysvaltojen itärannikolla käytetystä polttoaineesta, jonka se joutui sulkemaan muutaman päivän ajaksi. Putkijärjestelmän sulku johti polttoaineen hintojen nousemiseen ja hetkelliseen polttoaineen vajeeseen itärannikolla. Useassa osavaltiossa julistettiin myös hätätila. Putkijärjestelmän koko käy selväksi kuviosta 2 (Martinez 2021). Yritys maksoi hyökkääjille lunnaita noin 4.4 miljoonan dollarin edestä bitcoinien muodossa. (Morrison 2021.) Tietoturvayhtiö Kaspersky (2021) kertoo, että haittaohjelman kehittäjä DarkSide on myynyt ohjelmaa ”Ransomware-as-a-service” (RaaS)-mallilla, jossa kolmas osapuoli on löytänyt haavoittuvuuden kohteesta, ja ostaa kiristysohjelman, sekä sen vaativan infrastruktuurin palveluna haittaohjelman kehittäjältä.

Haavoittuvuuden löytäminen järjestelmästä on yleisesti ottaen helpompaa kuin oman kiristyshaittaohjelman kirjoittaminen ja palveluna myyty hyökkäys loi tässä tapauksessa tilanteen, jossa haittaohjelman käytöllä oli suunniteltua suurempi vaikutus. Ei ole vaikeata kuvitella, miten pitkäaikainen tärkeän infrastruktuurin sulkeminen voisi vaikuttaa yhteiskunnan toimintaan.



KUVIO 2. Colonial Pipeline-putkijärjestelmä (Martinez 2021).

2.3.2 Kaseya, REvil

Arviolta noin viidenkymmenen ”managed service provider” (MSP)-yrityksen asiakkaat joutuivat heinäkuussa 2021 kiristysohjelmahyökkäyksen kohteeksi, nollapäivähaavoittuvuuden sisältäneen Kaseya VSA-ohjelmiston välityksellä. Hyökkääjänä toiminut REvil-ryhmä vaati 70 miljoonaa dollaria yleisestä salauksen purkuavaimesta, jolla kaikki tartunnan saanet, olisivat voineet palauttaa tiedostonsa. Yksittäiset lunnaspyynnöt vaihtelivat yrityskohtaisesti, mutta yleisin summa oli noin 45 000 dollaria. Eräs asiakas, ruotsalainen elintarvikekauppaketju Coop joutui sulkemaan 800 liikkeistään kun kauppojen kassajärjestelmät lakkasivat toimimasta hyökkäyksen vuoksi. Kokonaisuudessaan noin 800–1500

yritystä sai tartunnan haittaohjelmasta, joka oli ilmoitetusti todella pieni määrä haavoittuvuuden sisältäneen ohjelmiston käyttäjistä. (Osborne 2021.)

Tapaus on hyvä esimerkki siitä, miten suuri vaikutus yhdellä hyökkäyksellä voi olla, kun se kohdistuu korkeammalle toimitusketjussa. Hyökkäyksen tarkoitus oli levittää haittaohjelmaa mahdollisimman moneen kohteeseen kerralla, ja Kaseyan VSA-ohjelmiston suuri käyttäjämäärä tekikin siksi siitä hyökkääjille kiinnostavan. Kuviossa 3 on esitetty REvil-kiristysohjelman maksusivusto.

KUVIO 3. REvil-kiristysohjelman maksusivusto (Loman, Gallagher & Ajjan n.d).

2.3.3 Hyökkäyksiä terveydenhuollon alalla, Doppelpaymer & WannaCry

Eräs kiristysohjelmatapaus johti ihmisen kuolemaan Saksassa syyskuussa 2020. Düsseldorfin yliopistollisen sairaalan tietokonejärjestelmät olivat toimintakyvyttömiä noin viikon verran, kun sairaalan verkkoon levisi Doppelpaymer-kiristyshaittaohjelma. Hyökkäyksen takia sairaala ei kyennyt hoitamaan erästä tuona aikana saapunutta hätäpotilasta, joka menehtyi matkalla toiseen sairaalaan. Haittaohjelmaa levittäessään hyökkääjät luulivat olevansa Düsseldorfin yliopiston verkossa sairaalan sijaan, tai niin ainakin väittävät. Ryhmä toimitti

sairaalalle salauksen purkuavaimen kuultuaan hyökkäyksen todellisesta kohteesta. (Corfield 2020.) Vastaavanlainen tilanne syntyi myös vuonna 2017, kun WannaCry-kiristyshaittaohjelma levisi yli 200 000 laitteeseen maailmanlaajuisesti, joista osa oli Yhdistyneen kuningaskunnan julkisen terveydenhuoltojärjestelmän (NHS) laitteita. Tapaus ei johtanut kuolemiin, mutta tuhansia ajanvarauksia ja operaatioita peruutettiin ja tietyillä alueilla hätäpotilaita jouduttiin ohjaamaan toisiin hoitopisteisiin. (Morse 2017, s. 4,8.)

2.3.4 Kansainvälisesti laajasti levinnyt esimerkki, Emotet

Tammikuussa vuonna 2021 Emotet-troijalaisen lähes kymmenen vuotta kestänyt leviämiskampanja yritettiin estää Europolin ja Eurojust:n koordinoimalla toiminnalla useassa eri maassa. Haittaohjelman infrastruktuuriin kuului satoja palvelimia ympäri maailmaa. Emotet havaittiin ensimmäistä kertaa vuonna 2014 ja se luokiteltiin pankkitroijalaiseksi, mutta nykyään hyökkääjät käyttävät sitä varastamaan dataa, sekä myyntialustana toisille haittaohjelmille. Se leviää haitallisia makroja sisältävien liitetiedostojen kautta, joita lähetetään roskapostin mukana, sekä tietokoneelta toiselle verkkolevyjen tai jaettujen kansioiden avulla. (Europol 2021.) Troijalaisen uusi versio sisältää analyysia vaikeuttavia metodeja, kuten koodin obfuskointia ja ohjelman merkkijonojen, sekä komentokeskuksen kanssa käydyn keskustelun salaamista. Viranomaisten toiminnasta huolimatta Emotet-troijalaisen uutta versiota havaittiin jälleen marraskuussa 2021. (zscaler 2021.)

Emotetin tekee vaaralliseksi sen toimintamalli ja levinneisyys. Ohjelma on hankala havaita ja poistaa, joka on johtanut useisiin tartuntoihin. Suuri tartuntojen määrä on taas mahdollistanut sen kehittäjille tavan tehdä tuottoa myymällä hallintaa haavoittuneista laitteista muille kyberrikollisille.

Tapauksista voidaan päätellä, ettei haittaohjelmien vaikutusta tulisi ajatella vain rahallisesti, eikä myöskään vain yritysten kannalta. Kyberrikollisten yleisin motivaatio on varmastikin raha eikä pelkästään pahanteko, mutta selkeästi on myös tilanteita missä hyökkäyksien seuraamuksia ei ole ajateltu loppuun, tai suuremmista haitoista ei välitetä. Haittaohjelma saattaa levitä kohteisiin mihin ei ollut tarkoitus, tai sen toiminnalla on suurempi vaikutus mitä ajateltiin. Tapauk-

sia on lukuisia muitakin, mutta jo näistä huomataan, että haittaohjelmilta puolustautuminen on tärkeää varsinkin nykypäivänä.

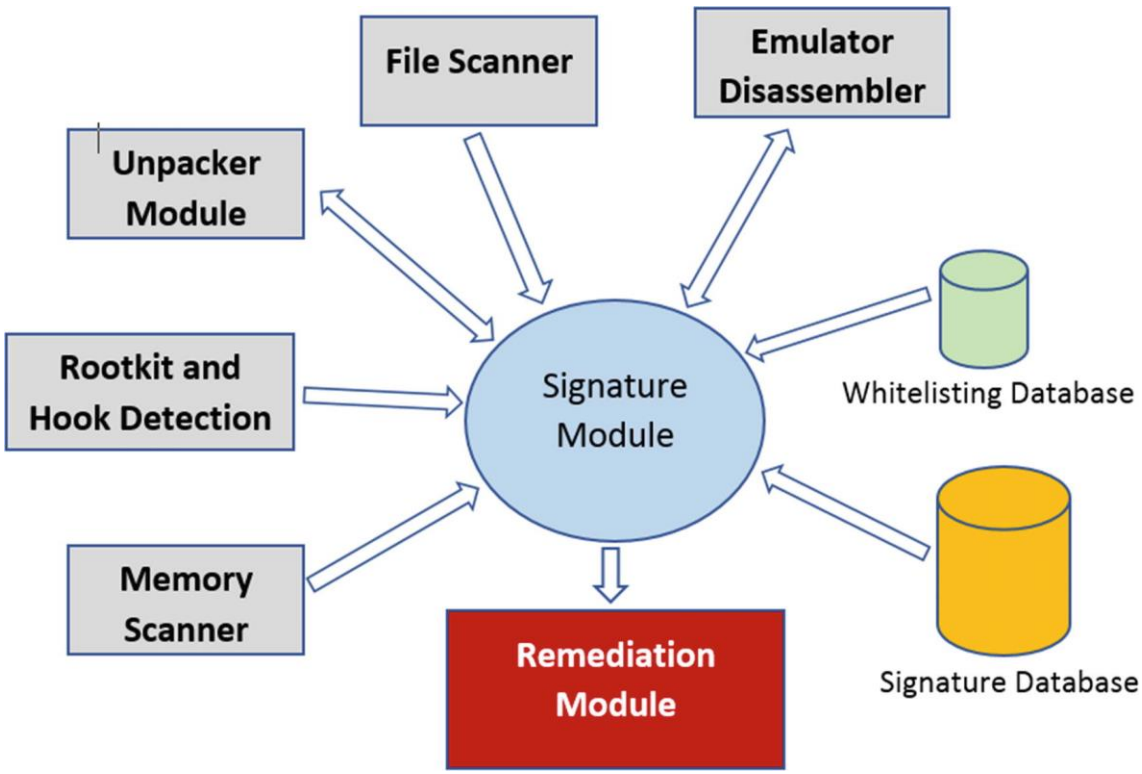
2.4 Puolustautuminen

Haittaohjelmilta voidaan puolustautua kahdella tavalla; käyttäjän tarkkaavaisuudella ja tietoisuudella, sekä ohjelmisto-, tai laiteratkaisuilla.

Turvallisuusalan toimijat ovat kehittäneet erilaisia ratkaisuja uhkien estämiseksi, jotka suojelevat käyttäjiään haittaohjelmilta ja kyberhyökkäyksiltä yksittäisten päätteiden ja verkon tasolla. Ohjelmisto- ja laiteratkaisuihin kuuluvat antivirus-, Endpoint Detection and Response (EDR)-, Intrusion Detection Systems/Intrusion Prevention Systems (IDS-/IPS-) ja Data Leak Prevention (DLP) -ohjelmistot, sekä palomuurijärjestelmät. Haittaohjelmiin erikoistunut antivirus on näistä ratkaisuista yleisin, yksinkertaisin, sekä vanhin ja se toimii lähinnä yksittäisten päätteiden tasolla. (Yehoshua & Kosayev 2021, 1.) Antiviruksen toiminta myötäilee pääpiirteittäin haittaohjelmien analyysimetodeja ja siksi siihen keskitytään tässä työssä tarkemmin.

2.4.1 Antivirus

Antivirusohjelmistot koostuvat useasta eri komponentista, jotka ovat suunniteltu havaitsemaan haittaohjelmia eri tavoin. Haittaohjelmien tunnistus tapahtuu ns. ”allekirjoituksien” (engl. ”signature”) avulla. Allekirjoitukset ovat käytännössä kokoelma malleja tai sääntöjä, jotka kuvailevat haitallisen ohjelman piirteitä tai toimintaa, jonka avulla ohjelma voidaan tunnistaa. Skannatessaan ohjelmia, rekistereitä, tai tietokoneen muistia antivirus vertailee dataa allekirjoitustietokantaan, jota päivitetään aika ajoin antiviruksen kehittäjien toimesta. Havaitessaan allekirjoitusta vastaavaa dataa antivirus yrittää poistaa haittaohjelma ja korjata sen aiheuttamat vahingot. Antivirus toimii yleisesti käyttöjärjestelmän käyttäjä- sekä ydintiloissa (kernel). Kernel-tilassa toimiminen estää haittaohjelman sulkemasta antivirus ohjelmistoa ja mahdollistaa rootkit-haittaohjelmien havaitsemisen. (Mohanta & Saldanha 2020, 7.) Kuviossa 4 esitetään karkealla tasolla useimpien antivirusohjelmistojen komponentit ja niiden välinen toiminta.



KUVIO 4. Antivirusohjelmistojen yleisimmät komponentit (Mohanta & Saldanha 2020, 7).

Allekirjoituksia jaetaan luokkiin sen perusteella, minkälaista dataa vastaan niitä vertaillaan. Staattisia allekirjoituksia tulisi verrata dataan, jota on esimerkiksi kerätty muistista olevista tiedostoista suorittamatta niitä. Staattiset allekirjoitukset voivat koostua esimerkiksi tiedoston tiivistesummasta (engl. hash), tai osittaisesta tiivistesummasta, tiedoston ominaisuuksista, kuten koko, tai tiedostopääte, tai tiedoston sisältämistä merkkijonoista tai tietyistä koodinpätkistä. (Mohanta & Saldanha 2020, 7.)

Käyttäytymiseen pohjautuvia allekirjoituksia verrataan dataan, jota kerätään järjestelmässä tapahtuvista muutoksista, kun skannattava ohjelma suoritetaan virtuaalisessa ympäristössä, erillään tietokoneen omasta muistialueesta. Tätä prosessia kutsutaan emuloinniksi. Näihin allekirjoituksiin sisältyy esimerkiksi API-kutsujen tarkkailuun tai prosessien muistiin liittyviä sääntöjä. (Mohanta & Saldanha 2020, 7.)

Käyttäytymiseen pohjautuva sääntö saattaa esimerkiksi tarkkailla, jos skannattava ohjelma tai prosessi yrittää kerätä dataa Windowsin LSASS-prosessista, joka sisältää käyttäjien NTLM-tiivisteet ja kerberos tiketit. Avoimia verkkoyh-

teyksiä ja portteja tarkkailemalla säännöllä voidaan havaita haittaohjelma, joka yrittää käydä keskustelua komentokeskuksen (Command & Control) kanssa. (Yehoshua & Kosayev 2021, 1).

Allekirjoitukset voivat olla joko tiukkoja, tai löysemmin määriteltyjä. Tiukat allekirjoitukset perustuvat ennalta tunnettujen haittaohjelmien ominaisuuksiin ja eivät yleensä tuota vääriä positiivia tuloksia. Löysemmin määriteltyjä allekirjoituksia kutsutaan heuristisiksi ja niiden on tarkoitus havaita haittaohjelmia, joita ei tunneta ennalta, ja siten ei voida tiukemmilla allekirjoituksilla havaita. Heuristiset allekirjoitukset tuottavat enemmän vääriä positiivisia tuloksia ja antivirus ei välttämättä ryhdy toimiin niiden perusteella. (Mohanta & Saldanha 2020, 7.) Heuristisiin allekirjoituksiin pohjautuva skannaus yhdistää staattiset ja käyttäytymiseen pohjautuvat allekirjoitukset ja antaa jokaiselle tarkkailtavalle ohjelmalle pistemäärän statistisen analyysin perusteella (Yehoshua & Kosayev 2021, 1).

Antivirusta käytetään sekä koti- että yritysympäristöissä, mutta loput ohjelmistoratkaisuista ovat käytännöllisyydeltään enemmän yrityskäyttöön sopivia.

2.4.2 EDR

Endpoint Detection & Response, eli EDR-ohjelmistot tarkkailevat verkossa useita päätteitä haitallisten toimintojen tai muutoksien varalta. Huomattuaan verkkoon liitetyssä päätteessä näitä ennalta määriteltyjä tapahtumia, siitä syntyy ilmoitus ohjelmiston hallintapaneeliin, jota yleisesti tarkkailee verkon ylläpitäjä. (Yehoshua & Kosayev 2021, 1.)

2.4.3 IPS/IDS

Intrusion Prevention- ja Intrusion Detection System -ohjelmistot ovat suunniteltu tarkkailemaan verkkoliikennettä haitallisen liikenteen varalta. Verkkoliikennetarkkailu perustuu yleisiin allekirjoituksiin, samoin kuin antiviruksen toiminta. (Yehoshua & Kosayev 2021, 1.)

2.4.4 DLP

Salaiseksi tai tärkeäksi merkittyä dataa voidaan varastaa esim. muistitikuilla, sähköpostilla tai lähettämällä sitä ulkoiselle palvelimelle. Data Loss Prevention-, eli DLP-ohjelmistot on luotu tämän ongelman ratkaisuksi. Ne suojelevat salaiseksi määriteltyä dataa monitoroimalla tapahtumia ja luomalla ilmoituksia, kun sitä käsitellään tai siirretään. (Yehoshua & Kosayev 2021, 1.)

Tulee huomioida, että ennakointi on myös hyvin toimiva metodi haittaohjelmilta suojautumisessa, sillä haittaohjelmien leviäminen ja suorittaminen vaatii lähes aina jonkinlaisen toimenpiteen käyttäjän toimesta, kuten aikaisemmin mainittiin. Ennakoinnilla tarkoitetaan siis käyttäjän tarkkaavaisuutta ja tietoisuutta haittaohjelmien leviämistavoista. Yritysympäristössä tämä on tietysti vaikeampaa, missä käyttäjien määrä on suurempi ja käyttäjien tieto aiheeseen liittyen voi vaihdella suuresti. Hyökkääjällä on yleisesti suurempi mahdollisuus hyötyä haittaohjelmien levittämisestä tässä ympäristössä, joten henkilöstön kouluttaminen tulisi olla osa minkä tahansa yrityksen kyberturvallisuusstrategiaa.

3 HAITTAOHJELMIEN ANALYYSI

Haittaohjelmien analyysia suoritetaan, jotta voidaan oppia analysoitavan haittaohjelman toiminta. Näin ohjelman suorittamia haitallisia tapahtumia voidaan yrittää korjata ja sama haittaohjelma voidaan tunnistaa myös tulevaisuudessa, sekä mahdollisesti estää ennen kuin se tekee vahinkoa. Analysoinnilla voidaan myös kerätä tietoa yleisellä tasolla siitä, millaisia toimenpiteitä haittaohjelmat suorittavat ja miten. Tämän tiedon avulla voidaan rakentaa puolustuksia, jotka kykenevät havaitsemaan myös uusia, ennennäkemättömiä haittaohjelmia. Haittaohjelman analysointi voi useinkin johtaa myös kyberhyökkääjän jäljille ja kertoa hyökkääjän toimintatavoista ja motiiveista. (Monnappa 2018, 1.)

Haittaohjelmien analyysi voidaan jakaa staattisiin ja dynaamisiin metodeihin. Kaksi merkittävää metodia ovat koodi- ja muistianalyysi, joka tunnetaan myös muistiforensiikkana.

3.1 Staattinen analyysi

Haittaohjelmien staattinen analyysi kuvailee metodeja, joilla voidaan kerätä tietoa ohjelmasta suorittamatta sitä. Näihin metodeihin kuuluu mm. tiedostotyypin ja mahdollisen paketoinnin tunnistaminen, tiedostorakenteen ja merkkijonojen tarkastelu, sekä antivirus skannereiden hyväksikäyttö. Myös koodianalyysiä voidaan suorittaa disassemblerin tai decompilerin avulla. (Monnappa 2018, 2.)

3.1.1 Merkkijonot

Merkkijonot ovat binääritiedoston sisältämää tekstimuotoista dataa. Merkkijonot voivat sisältää tietoa haitallisen ohjelman toiminnasta ja niitä voidaan käyttää myös saman ohjelman tai samaan perheeseen kuuluvan haittaohjelman tunnistamiseen. Merkkijonot voivat sisältää mm. komentoja, tiedostonimiä, polkuja, IP-osoitteita tai verkko-osoitteiden nimiä. Haittaohjelmien kehittäjät usein käyttävät obfuskointimetodeja tärkeiden merkkijonojen ja koodin salaamiseksi, jolloin niitä ei voida kerätä binääristä ilman oikeanlaista työkalua. (Monnappa 2018, 2.)

3.1.2 Tiivisteet

Tiivistefunktiot ovat matemaattisia yksisuuntaisia funktioita, jotka muuttavat niille annetun syötteen määrätyn pituiseksi numeeriseksi ulostuloarvoksi. Tiivistefunktioita käytetään salasanojen säilyttämisessä, sekä datan eheyden takaamiseksi. (Tuorialspoint n.d.)

Kun haittaohjelmasta luodaan tiiviste, sen identtiset kopiot voidaan myös tunnistaa tulevaisuudessa vertailemalla sitä tietokantoihin, jotka sisältävät jo tunnettujen haittaohjelmien tiivisteitä. Haittaohjelmien analyysissa käytetään yleisesti MD5-tiivistefunktiota (Sikorski & Honig 2012, 1.) Ongelmia tiivisteiden käytössä aiheuttaa se, että haittaohjelmien tulee olla identtisiä tuottaakseen saman tiivisteen. Esimerkiksi samaan haittaohjelmaperheeseen kuuluvat haittaohjelmat voivat olla toiminnallisuudeltaan samoja, mutta pientenkin eroavaisuuksien vuoksi niitä ei voida tunnistaa pelkällä perinteisellä tiivisteellä. Näissä tilanteissa käytetään ”fuzzy hashing”-nimistä tekniikkaa, joka kertoo kahden binaarin välisen samankaltaisuuden prosenttiarvona. (Monnappa 2018, 2.)

3.1.3 YARA

YARA on työkalu, jolla luodaan tekstipohjaisia sääntöjä ja sitä käytetään haittaohjelmien tunnistamisessa ja luokittelussa. YARA-säännöt koostuvat merkkijonoista ja boolean-ehdoista, jotka täyttäessään, sääntöön verrattava ohjelma voidaan merkitä haitalliseksi. YARA sääntöjä voidaan luoda manuaalisesti millä tahansa tekstieditorilla tai automaattisesti YarGen-skriptillä ja niitä voidaan verrata yksittäisiä ohjelmia-, tai vaikkapa RAM-kaappauksia tai levykekuvia, ts. koko kovalevyn sisältöä vastaan. (Johansen 2020, 4.)

YARA:n pohjalta on rakennettu useampia tietokantoja ja skannereita ja se on hyvinkin yleisessä käytössä kyber- ja tietoturvallisuuden aloilla. Eräs esimerkki tästä on Nextron Systemsin rakentama YARA-sääntötietokanta nimeltään VALHALLA ja siihen liittyvät skannerit THOR ja sen ilmainen, kevyempi versio LOKI. Kuviossa 5 esitetään yksinkertainen esimerkki YARA-säännöstä.

```
rule esimerkki
    meta:
        description = "Esimerkkisääntö"
    strings:
        $a = "123.45.22.80"
        $b = "haittaohjelma_nimi"
        $c = "esimerkki_merkkijono"
    condition:
        $a or $b or $c
}
```

KUVIO 5. Yksinkertainen YARA-sääntö.

Esimerkkisääntö on määritelty etsimään kolmea eri merkkijonoa; $a, $b, tai $c. Ehdoissa on määritelty, että minkä tahansa näistä kolmesta merkkijonosta kohdatessaan, sääntö täyttyy ja skanneri luo siitä ilmoituksen. YARA sisältää monia muitakin ominaisuuksia sääntöjen luomiseen, mutta niitä ei käsitellä tässä työssä.

3.1.4 Verkkotyökalut

Verkkotyökalujen avulla on helppoa tunnistaa aikaisemmin tunnetut haittaohjelmat, ja se tulisi olla staattisen analyysin ensimmäisiä askeleita. Verkkotyökaluihin kuuluvat esimerkiksi useita antivirus-skannereita hyödyntävät verkkosivut kuten Virustotal.

Virustotal-sivustolle pystyy lataamaan tiedostoon liittyvää metadataa, kuten tiivisteitä, IP-osoitteita, verkkotunnuksia ja -osoitteita tai kokonaisia tiedostoja itsessään. Dataa ajetaan useamman antivirus-skannerin läpi ja jokaisen skannerin tulokset ja mahdolliset haittaohjelmanimikkeet esitetään erikseen. Tämän lisäksi sivusto tarjoaa myös graafisen työkalun, jolla voidaan tarkastella ladatun tiedoston ja siihen liittyvien tunnisteiden (kuten aikaisemmin mainitut IP-osoitteet) välisiä suhteita. (Monnappa 2018, 2.)

Verkkopohjaisia skannereita on muitakin, mutta Virustotal on niistä varmastikin tunnetuin.

3.2 Dynaaminen analyysi

Dynaaminen analyysi tarkoittaa mitä tahansa analyysitoimenpidettä, joka tehdään haittaohjelman suorittamisen jälkeen ja niitä yleisesti käytetään staattisen analyysin hyödyllisyyden päättyessä. Haittaohjelman toimintaa saatetaan tarkkailla muistissa sen suorituksen aikana, tai sen toimintaa mallinnetaan ottamalla huomioon sen vaikutukset ympäristöönsä, eli käyttöjärjestelmään tai laitteeseen, jossa sitä suoritetaan. (Sikorski & Honig 2012, 1.)

Haittaohjelmalla voi olla monia eri tehtäviä tai toimintoja, joita se suorittaa järjestelmässä, kuten uuden prosessin luominen, tiedostojen kirjoittaminen muistiin, rekisteriavaimien luominen, tai verkkoyhteyden käyttäminen. Kun näitä järjestelmän osa-alueita monitoroidaan reaaliajassa, haittaohjelman toiminta voidaan mallintaa paljon tarkemmin kuin pelkästään staattisella analyysillä. Järjestelmän sisäinen monitorointi tapahtuu erilaisilla ohjelmistoilla ja se voidaan jakaa kolmeen osa-alueeseen, verkkoliikenteen tarkkailun lisäksi. Nämä kolme osa-aluetta ovat prosessien monitorointi, tiedostojärjestelmän monitorointi ja rekisterin monitorointi. (Monnappa 2018, 3.)

Myös merkkijonoja tarkastellaan dynaamisesti. Ohjelmien pakkauksen takia merkkijonot eivät aina ole luettavissa binääristä ennen sen suorittamista, mutta lataamalla ohjelma muistiin, pakkaus poistuu ja siten myös tekstipohjainen data muuttuu luettavaksi ohjelman synnyttämän prosessin muistialueesta. (Mohanta & Saldanha 2020, 4.)

3.2.1 Prosessien monitorointi

Järjestelmän prosesseja tarkkailemalla saadaan selville minkälaisen, tai minkälaisia prosesseja haittaohjelman suorittaminen synnyttää. Prosesseista voidaan kerätä tietoa, kuten prosessinimet ja -tunnisteet (PID), sekä tiedostopolut. (Monnappa 2018, 3.)

3.2.2 Tiedostojärjestelmän monitorointi

Tiedostojärjestelmän monitoroinnilla havaitaan, jos suoritettava haittaohjelma luo uusia, tai muokkaa jo olemassa olevia tiedostoja. Haittaohjelma saattaa esimerkiksi kopioida itsensä toiseen sijaintiin (vaikkapa */System32*) eri nimellä hämätäkseen mahdollisia tarkkailijoita. (Mohanta & Saldanha 2020, 4.)

3.2.3 Rekisterin monitorointi

Windowsin rekisteri on tietokanta, joka sisältää käyttöjärjestelmän komponenttien ja muiden ohjelmien asetuksia ja konfiguraatioita. Rekisteri löytyy osittain massamuistista ja osittain keskusmuistista, jonne se luodaan järjestelmän käynnistyessä ja se sisältää runsaasti tietoa järjestelmästä. Siksi se onkin haittaohjelmalle hyödyllinen tiedonkeruun, sekä ns. ”pysyvyyden” (engl. ”persistence”) kannalta. Termillä tarkoitetaan tilanteita, joissa haittaohjelma kykenee muuttamaan tai luomaan reksiterin run-avaimiin uusia tietoja Win32 API:en avulla, jolloin haittaohjelma käynnistyy aina järjestelmän käynnistyessä. Haittaohjelmat usein myös piiloutuvat esittämällä itseään palveluina (engl. ”services”), jotka ovat erityisiä käyttöjärjestelmän hallitsemia prosesseja. Tämä tapahtuu myös rekisterin kautta services-avaimilla. Rekisterimuutoksia tarkkailemalla voidaan siten kerätä lisää tietoa haittaohjelman toiminnasta. (Mohanta & Saldanha 2020, 3 & 2.)

Reksiterin yksi käynnistykseen liittyvistä *run*-avaimista löytyy seuraavan polun alta:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

3.2.4 Verkkoliikenteen monitorointi

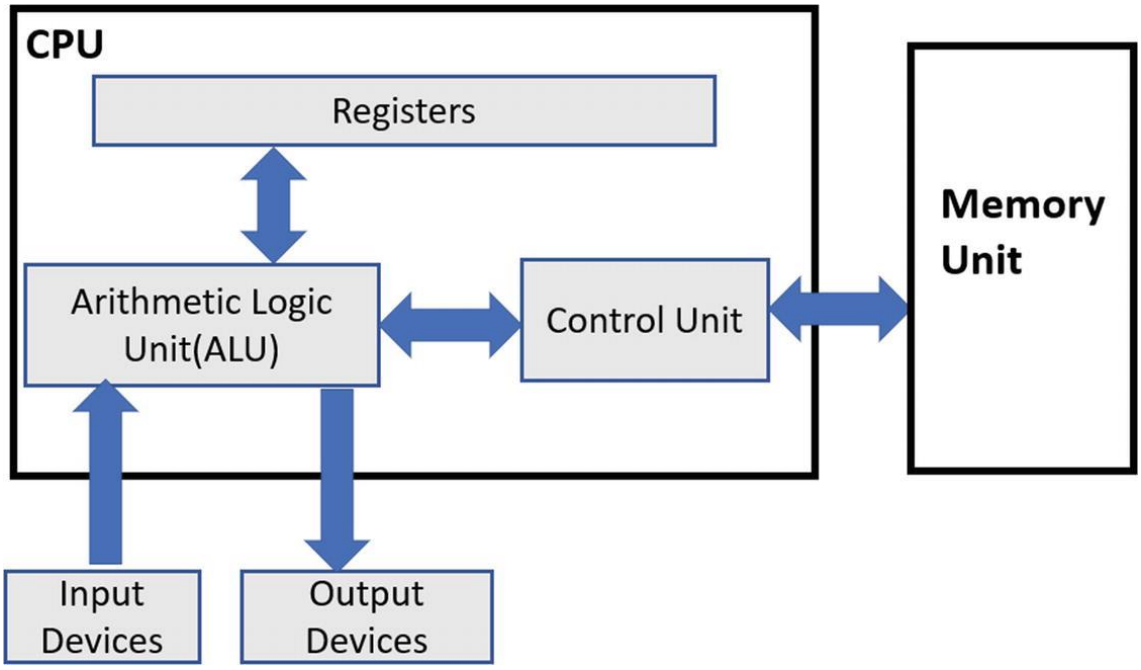
Verkkoliikenteen monitoroinnilla kerätään haittaohjelman tunnistamisen avuksi verkkopohjaisia tunnisteita kuten IP-osoitteita, DNS-nimiä ja IP-paketteihin perustuvia allekirjoituksia. Suorituksen jälkeen monet haittaohjelmat käyttävät verkkoyhteyttä johonkin tarkoitukseen. Näistä yleisin on yhteydenotto komentokeskukseen, joka nimensä mukaisesti kertoo haittaohjelmalle seuraavan toi-

menpiteen tai mahdollisesti kerää haittaohjelman lähettämää dataa. Tunnisteiden keräämiseksi virtuaalisessa ympäristössä verkon ja erilaisten verkon palveluiden simulointi on usein tarpeellista. (Sikorski & Honig 2012, 1).

3.3 Tietokoneen arkkitehtuuri

Suoritettavien ohjelmien, ja varsinkin koodianalyysissä esiintyvän assemblyn ymmärtämiseksi on hyvä myös ymmärtää tietokoneen arkkitehtuurin perustaa.

Tietokoneohjelmat käännetään suoritettaviksi tiedostoiksi, jotka sisältävät käskyjä konekoodin muodossa, joita järjestelmän prosessori suorittaa. Jokaisella prosessorilla on näiden käskyjen hakemiseen ja suorittamiseen suunniteltu arkkitehtuuri. Monet nykypäivän arkkitehtuurit pohjautuvat John Von Neumannin vuonna 1945 julkaisemaan Von Neumann -arkkitehtuuriin, joka on esitetty kuviossa 6. (Mohanta & Saldanha 2020, 2.)



KUVIO 6. Von Neumann-arkkitehtuuri (Mohanta & Saldanha 2020, 2).

Von Neumann-arkkitehtuuri sisältää kolme pääkomponenttia, jotka ovat prosessori, eli CPU, muisti, sekä input- ja output -laitteet, kuten monitorit, näppäimistöt, hiiret, kovalevyt, tai verkkokortit. Muistin on tarkoitus sisältää ohjelman sisältämät käskyt ja mahdollinen data, jota suoritettava ohjelma tarvitsee. Prosessori koostuu logiikka- ja laskupiiristä (ALU), ohjausyksiköstä ja rekistereistä. Oh-

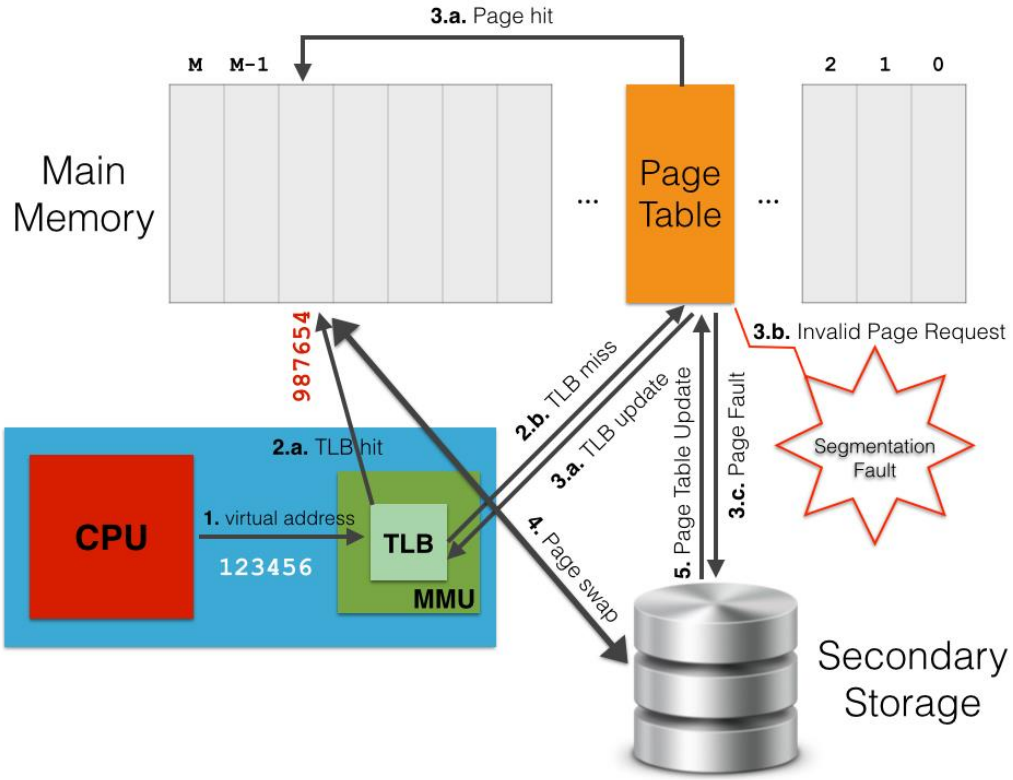
jausyksikkö hakee muistista ohjelman sisältämät käskyt, jotka suoritetaan logiikka- ja laskupiirillä. Suorituksen tulokset siirretään joko prosessorin johonkin prosessorin rekisteriin, tai takaisin muistiin. Rekisterit pitävät sisällään väliaikaisesti erilaista dataa, jota prosessorin käskyillä tahdotaan käsitellä. (Mohanta & Saldanha 2020, 2.)

3.4 Virtuaalimuisti, käyttäjätila ja kernel-tila

Erilaiset laitteiston luomat rajoitteet ovat olleet ongelmana tehokkaan ohjelmistokehityksen kannalta jo kauan, ja fyysisen muistin rajoitteiden lievittämiseksi on kehitetty virtuaalisen muistin konsepti, joka on käytössä tänä päivänä kaikissa käyttöjärjestelmissä. Lyhyesti kuvattuna virtuaalinen muisti käyttää hyväkseen kiintolevyllä olevaa muistitilaa, simuloidakseen fyysistä RAM-muistia. Esimerkiksi 32-bittisessä Windowsissa jokainen suoritettava prosessi saa 4 Gt:n verran virtuaalista muistia, vaikka fyysisen RAM-muistin koko olisi 1 Gt, jolloin prosessit voivat suorittaa tehtävänsä samanaikaisesti haittaamatta toistensa toimintaa. (Mohanta & Saldanha 2020, 2.)

Windowsissa tätä kiintolevyltä varattua muistia kutsutaan sivutiedostoksi (engl. page file) ja Linuxissa vaihtotilaksi (engl. Swap Space). Virtuaalinen-, kuten fyysinenkin muisti on osoitteellista, eli jokaisella tavulla on oma osoitteensa, jolla sitä voidaan käsitellä. Fyysinen muisti jaetaan osiksi, joita kutsutaan kehyksiksi (engl. frame) ja virtuaalisen muistin osia kutsutaan sivuiksi (engl. page), jotka vastaavat toisiaan samassa suhteessa, ts. jokainen sivun sisältämä osoite voidaan kääntää vastaavaan fyysisen muistin osoitteeseen ns. sivutaulun (engl. page table) avulla. (Mohanta & Saldanha 2020, 2.)

Sivutaulu sisältää tietoa siitä, mikä sivu vastaa mitäkin kehystä. Tilanteessa, jossa fyysinen muisti on varattu kokonaan, ja uusi prosessi vaatii virtuaalimuistia käyttöjärjestelmältä, fyysisestä muistista vapautetaan jonkin toisen prosessin sivu, joka ei ole tällä hetkellä käytössä. Jos tätä aikaisemmin vapautettua sivua tarvitaan uudestaan, mutta sitä ei ole ladattu fyysiseen muistiin, käyttöjärjestelmä luo sivuvirheen (engl. page fault) ja vaihtoprosessi toistetaan. (Mohanta & Saldanha 2020, 2.) Tämä prosessi on kuvattu kuviossa 7.



KUVIO 7. Fyysisen- ja virtuaalimuistin vaihtoprosessi (Tolomei n.d).

Muistiosoitteet esitetään usein hex-muodossa, ja esim. 32-bittisen Windowsin virtuaalinen muistialue näyttää seuraavalta: 0x00000000 -- 0xFFFFFFFF, jossa 0x00.. on ensimmäisen tavun osoite ja 0xFF… on viimeisen tavun osoite. Jokaiselle prosessille määrätty virtuaalimuisti jaetaan kahteen osaan; käyttäjätilaan ja kernel-tilaan, joista kernelille varattu muistitila on kaikille prosesseille yhteinen. (Mohanta & Saldanha 2020, 2.)

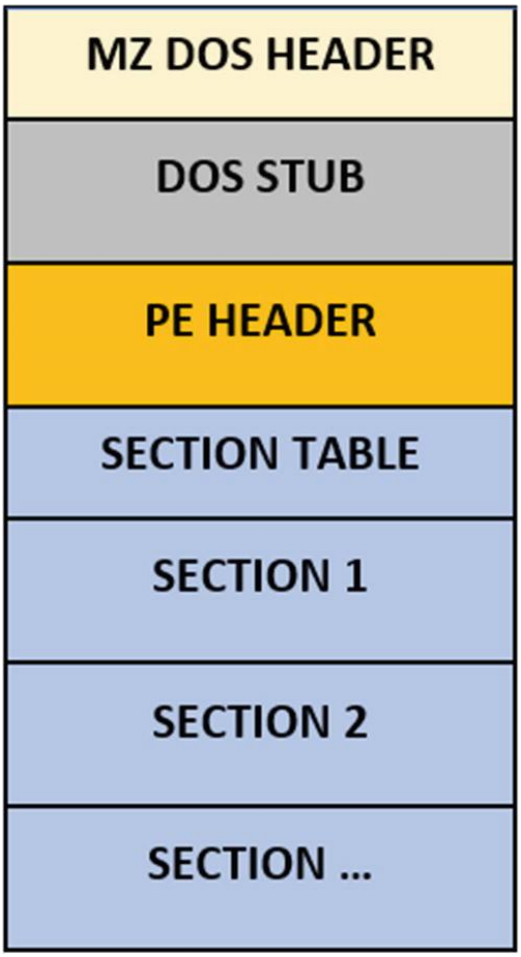
Kernel-tila sisältää itse käyttöjärjestelmän ja sen laitteiston ajurit. Kernel-tilassa suoritettava koodi toimii korotetuilla oikeuksilla, ja se voi käsitellä sekä käyttäjätilan-, että kernel-tilan muistia, jota käyttäjätilassa suoritettava ohjelma ei voi tehdä. Käyttäjätilan prosessit pystyvät käyttämään järjestelmän laitteistoa (esim. tiedoston kirjoittaminen kiintolevylle) Windowsin API-kutsujen avulla, joita se tarjoaa järjestelmän omien .dll-kirjastojensa kautta. (Monnappa 2018, 4.)

Haittaohjelmat hyväksikäyttävät näitä API-kutsuja koodi-injektion ja API-koukkujen avulla, joista kerrottaan seuraavassa kappaleessa.

3.5 PE-tiedostoformaatti

Kun ohjelma suoritetaan, käyttöjärjestelmän tulee tietää, miten ohjelmasta luodaan prosessi, eli ts. kuinka paljon virtuaalimuistia se tarvitsee ja mihin ohjelman koodi ja data luodaan virtuaalimuistissa (Mohanta & Saldanha 2020, 2).

Windows käyttöjärjestelmän suoritettavat tiedostot seuraavat Portable Executable-, eli PE-tiedostoformaattia. Suoritettavia tiedostoja ovat esimerkiksi ”.exe”, ”.dll”, ”.sys”, ”.ocx” ja ”.drv” ja ne koostuvat rakenteista, jotka sisältävät käyttöjärjestelmän vaatimat tiedot ohjelman muistiin lataamiseksi. Tiedostorakenne luodaan ohjelman kääntämisvaiheessa. (Monnappa 2018, 2.) Nämä rakenteet ovat jaettu otsikoihin (engl. ”header”) ja sektioihin (engl. ”section”). Kuviossa 8. esitetään PE-tiedostoformaatin rakenne.



KUVIO 8. PE-tiedostoformaatin rakenne (Mohanta & Saldanha 2020, 2).

PE-tiedoston tunnistaa aina MZ-DOS-otsikosta, joka löytyy ajettavan ohjelman alusta. Näitä ensimmäisiä tavuja, joilla tunnistaa kaikki muutkin tiedostotyypit, kutsutaan termillä taikatavut (engl. ”magic bytes”). (Mohanta & Saldanha 2020, 2.)

Alla on lueteltu yleisimpiä PE-tiedoston sektioita

.text: Sisältää prosessorin suorittamat käskyt. Yleisesti tämä on ainoa sektio, joka on suoritettavissa ja sisältää koodia (Sikorski & Honig 2012, 1.)

.rdata: Sisältää ohjelman globaalia vain-luettavaa dataa. Tähän sisältyy joskus myös ns. tuontiin ja vientiin liittyvää dataa, eli koodia, jota ohjelma ”lainaa” muilta ohjelmilta (tuonti) tai mitä muut ohjelmat saattavat käyttää (vienti). Yleisesti tämä data on erillisissä **.idata** ja **.edata** sektioissa. (Sikorski & Honig 2012, 1.)

.data: Sisältää ohjelman globaalia dataa. (Sikorski & Honig 2012, 1.)

.rsrc: Sisältää ohjelman käyttämiä resursseja, joita ei lasketa kuuluvaksi itse suoritettavaan tiedostoon. Tähän kuuluvat esimerkiksi ikonit, kuvat, merkkijonot ja valikot. (Sikorski & Honig 2012, 1.)

Haittaohjelmat saattavat pakkaamisen takia sisältää epämääräisiä tai kummallisesti nimettyjä sektioita ja niiden tarkempi analysointi voi kertoa onko analysoitava ohjelma haitallinen (Monnappa 2018, 2).

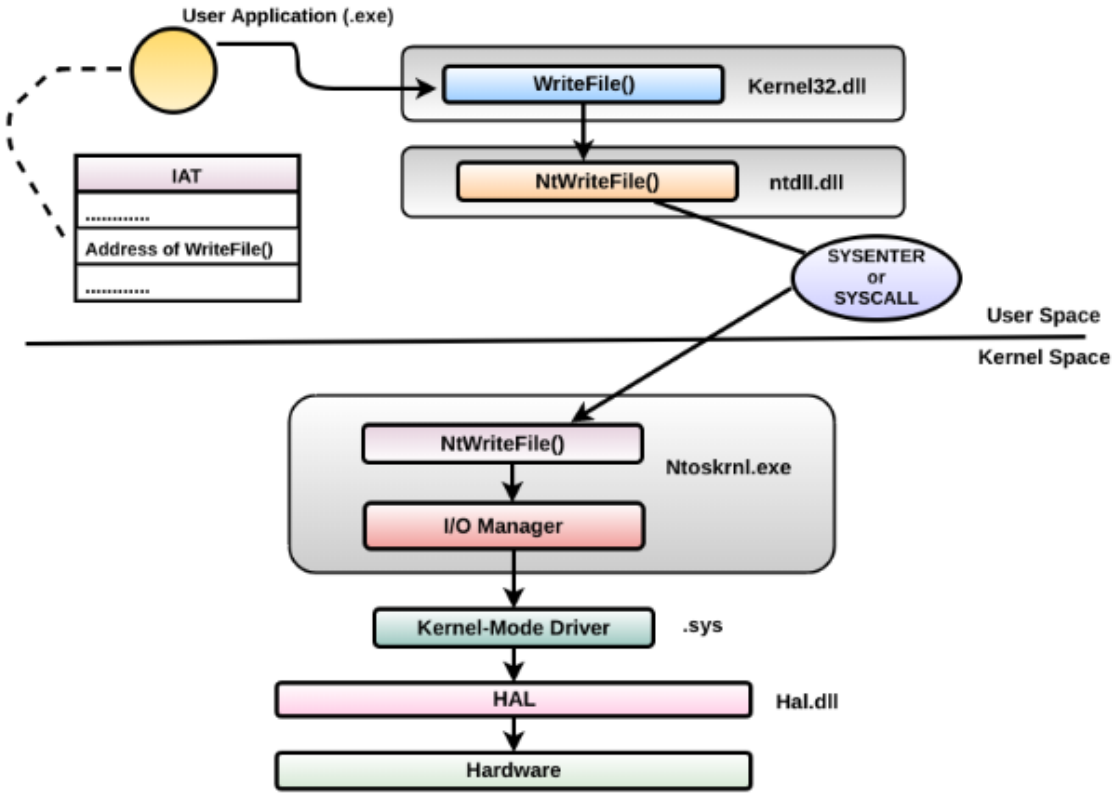
3.6 Win32 API

Windows käyttöjärjestelmä tarjoaa ohjelmille tapoja käsitellä järjestelmän laitteistoa ja käyttöjärjestelmää ohjelmointirajapintojen, eli Application Programming Interface, avulla. Näitä API:ja sisällytetään eri .dll-, eli ”Dynamic Link Library” -tiedostoihin. Kaikki ohjelmat käyttävät näitä API-kutsuja, mutta haittaohjelmat usein kutsuvat tiettyjä API:ja tietyssä järjestyksessä, ja ne voidaan siten tunnistaa kutsujen avulla. Analyysitilanteessa täytyykin kutsuja tarkistellessa huomioida myös kutsun konteksti, eikä vain sen olemassaolo, ts. ”Kuuluisiko

tämän ohjelman tehdä tätä/näitä kutsuja ollenkaan, tai tällä hetkellä?” (Mohanta & Saldanha 2020, 2).

3.6.1 API-kutsun kulku

Normaalisti prosessin kutsuessa esim. DeleteFile API:a, joka tässä tapauksessa sisältyy *Kernel32.dll*-tiedostoon, sen täytyy tietää kyseisen funktion muistiosoite. Tämä muistiosoite löytyy ns. tuontitaulukosta (engl. Import Address Table, IAT), jos ohjelma on tehnyt linkityksen ennen suorittamista. *Kernel32.dll*-tiedoston sisältämä DeleteFile-funktio ohjaa käskyn *ntdll.dll*-tiedostolle, johon on implementoitu NtDeleteFile-funktio. *ntdll.dll*-tiedosto toimii vain välikätenä, joka muuttaa nämä API-kutsut järjestelmäkutsuiksi (engl. system call). Järjestelmäkutsujen avulla suoritettava koodi siirtyy muistialueen kernel-tilaan. Kernel-tilassa sijaitseva *ntoskrnl.exe* sisältää oikean implementaation NtDeleteFile-funktiosta. Funktion muistiosoite löytyy kernel-tilan System Service Descriptor Table (SSDT)-muistitaulusta. Funktion kutsuessaan käsky ohjataan jälleen I/O managerin I/O-funktioille, jotka lähettävät nämä laitteistolle tehtävät pyynnöt vastaaville laitteiston kernel-tilan ajureille, jotka lopulta käyttävät käyttöjärjestelmän HAL:n, eli Hardware Abstraction Layer:n luomia rutiineja tehdäkseen laitteistoon tarvittavat toimenpiteet ja muutokset. (Monnappa 2018, 8). Prosessi on kuvattu kuviossa 9.



KUVIO 9. Windows API-kutsun kulku (Monnappa 2018, 8).

3.7 Mutant/Mutex

Windows käyttöjärjestelmässä ”mutex” on synkronointiobjekti, jota useammat prosessit tai säikeet käyttävät samojen resurssien käyttämisen synkronointiin. Jos jokin prosessi tahtoo varmistaa, että siitä on olemassa vain yksi instanssi kerrallaan, tai kaksi säiettä haluavat käsitellä samaa muistialuetta turvallisesti, eli yksi kerrallaan, ne luovat mutex:n, jota kutsutaan nimityksellä ”mutant”, jos se luodaan kernel-tilaan. Mutex:t luodaan CreateMutex-funktion avulla. Haitta-ohjelmat usein käyttävät tätä toiminnallisuutta varmistaakseen, ettei samaa haittaohjelmaa suoriteta useasti yhdessä kohteessa, ja koska niiden luomien mutex:ien nimet ovat usein määriteltyjä haittaohjelma koodissa, voidaan niiden avulla tunnistaa haittaohjelman olemassaolo, olettaen että sen käyttämät mutex-nimikkeet tunnetaan entuudestaan. (Sikorski & Honig 2012, 2; Mohanta & Saldanha 2020, 2.)

3.8 Koodianalyysi

Koodianalyysi on ohjelman sisältämän koodin analysointia sen toiminnan ymmärtämisen toivossa ja sillä voidaankin selvittää enemmän ohjelman toiminnasta kuin aikaisemmilla analysointimetodeilla. Tämä toimenpide luonnollisesti vaatii ymmärrystä käytetystä ohjelmointikielestä, sekä mahdollisesti myös käyttöjärjestelmästä ja on siten myös vaativampaa. Myös koodianalyysi voidaan jakaa erikseen staattiseen- ja dynaamiseen analyysiin. Samoin kuin aikaisemmissa metodeissa, staattinen koodianalyysi on ohjelmakoodin tarkastelua suorittamatta sitä, joko dissassemblerin tai decompilerin avulla, kun taas dynaamisessa koodianalyysissa ohjelma suoritetaan ja sen toimintaa tarkkaillaan sen ajon aikana debuggerin avulla. (Monnappa 2018, 5.)

3.8.1 Disassembly

Dissassembler on ohjelma, joka kääntää ohjelman binäärin, eli konekoodin assembly-koodiksi. Assembly-koodi on yksinkertaisuudessaan ohjelman konekielen, eli prosessorin ymmärtävät käskyt, muunnettuna ihmisluettavaan muotoon. Prosessorin käskyjä lukemalla ja analysoimalla ohjelman toiminta voidaan määritellä matalimmalla mahdollisella tasolla. Jokaisella prosessoriperheellä on oma käskykantansa. x86-arkkitehtuurille kirjoitetut ohjelmat toimivat 32- ja 64-bittisessä Windowsissa ja siten myös suurin osa haittaohjelmista on kirjoitettu tälle arkkitehtuurille. (Monnappa 2018, 5.)

Kuviossa 10. on esitetty yksinkertainen, C++ -ohjelmointikielellä kirjoitettu, ”Hello World!”-ohjelma, joka on käännetty assembly-kielelle IDA Freeware-ohjelmalla kuviossa 11.

```cpp
#include<iostream>
using namespace std;

int main() {

    cout << "Hello World!";
    system("pause");
    return 0;

}
```

KUVIO 10. C++ kielellä kirjoitettu yksinkertainen ”Hello World!”-ohjelma.

```
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

var_C0= byte ptr -0C0h
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_C0]
mov     ecx, 30h ; '0'
mov     eax, 0CCCCCCCCh
rep stosd
mov     ecx, offset unk_41E027
call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
push    offset Str      ; "Hello World!"
mov     eax, ds:__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; std::basic_ostream<char
push    eax             ; int
call    j_??$?6U?$char_traits@D@std@@@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@@AAV10@PBD@Z ;
add     esp, 8
mov     esi, esp
push    offset Command  ; "pause"
call    ds:__imp__system
add     esp, 4
cmp     esi, esp
call    j___RTC_CheckEsp
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 0C0h
cmp     ebp, esp
call    j___RTC_CheckEsp
mov     esp, ebp
pop     ebp
retn
_main endp
```

KUVIO 11. Kuvion 7 ”Hello World!”-ohjelma käännettynä assembly-kielelle.

Kuvion 11 vasemmassa laidassa näkyvät konekielen käskyt assemblyksi käännettynä; push, mov, sub, jne.

Käskyjä on assemblyssä yhteensä 13 ja niillä siirretään ja käsitellään dataa prosessorin rekistereiden ja tietokoneen muistin väleillä. x86-arkkitehtuurissa yleisrekistereitä on 8 kappaletta: eax, ebx, ecx, edx, esp, ebp, esi ja edi, jotka ovat 32:n bitin kokoisia. Ohjelma voi käsitellä rekistereissä olevaa dataa 32:n, 16:a, 8:n, tai yhden bitin kokoisina arvoina. (Monnappa 2018, 4.)

Yksi ohjelman toiminnan kannalta tärkeä prosessorin rekisteri on ns. instruction pointer, eli eip, joka sisältää seuraavan suoritettavan käskyn muistiosoitteen. (Monnappa 2018, 4). Hyökkääjä voi esimerkiksi saada prosessorin suorittamaan haluamiaan käskyjä toisen ohjelman prosessin kautta, jos tämä onnistuu ylikirjoittamaan eip-rekisterin sisältämän muistiosoitteen.

3.8.2 Decompiling

Decompiler on ohjelma, jolla ohjelman konekoodi voidaan kääntää ”takaisin” korkean tason kiellelle, pseudokoodin muodossa (Monnappa 2018, 5). Käännetty pseudokoodi ei tosin voi koskaan täsmätä alkuperäistä lähdekoodia, mutta on decompiler on työkaluna joka tapauksessa usein hyödyllinen ohjelman toiminnan mallintamisessa.

3.8.3 Debugging

Debuggeri on ohjelma, jolla voidaan hallita sillä käsiteltävän ohjelman suorittamista hyvinkin tarkasti reaaliajassa. Käsiteltävää ohjelmaa voidaan suorittaa vaikkapa käsky tai funktio kerrallaan. (Monnappa 2018, 5.) Debuggerin käyttö on dynaamista koodianalyysia, ja sillä voidaan kerätä usein tarkemmin tietoa ohjelman toiminnasta staattiseen analyysiin verrattuna, varsinkin silloin kun ohjelman koodia on obfuskoitu, sillä muistissa ohjelma on aina suoritettavien käskyjen muodossa.

3.9 Muistianalyysi

Aina kun ohjelma tai käyttöjärjestelmä suorittaa jonkin toimenpiteen, se vaatii muutoksia tietokoneen muistiin. Nämä muutokset ovat usein havaittavissa vielä pitkän aikaa tapahtuman jälkeen ja siksi tietokoneen muisti antaa hyvinkin tar-

kan kuvan järjestelmän suoritusaikaisesta tilasta. Muistia tarkkailemalla voidaan selvittää esimerkiksi, mitkä prosessit olivat käynnissä, mitä verkkoyhteyksiä tietokoneella on avattu, tai mitä komentoja tietokoneella on suoritettu. Myös kriittinen data, kuten kiintolevyjen salausavaimet, salaamattomat sähköpostit ja vaikkapa chat-viestit, tai injektoidut koodinpätkät löytyvät yksinomaisesti muistin sisällöstä. Tästä syystä muistin analysointi on tehokas työkalu haittaohjelmien analysointiprosessissa ja muissa kyberhyökkäyksiin liittyvissä tilanneselvityksissä. (Ligh, Case, Levy, & Walters 2014, Introduction.)

Muistianalyysin avulla voidaan myös havaita haittaohjelmia, jotka eivät ole huomattavissa kiintolevyltä, ts. ne ovat tiedostottomia. Jotkut haittaohjelmat käyttävät analyysia vaikeuttavia metodeja, kuten ns. ”koukkujen” käyttäminen (engl. hooking), tai käyttöjärjestelmän omien rakenteiden muuttamista. Molemmissa tapauksissa nämä välttelymenetelmät voidaan torjua muistia tarkkailemalla. (Monnappa 2018, 9.)

Koska muistianalyysin aikana koko tietokoneen muisti on käytettävissä, voidaan tällä metodilla myös havaita kernel-tilan muistissa olevat haittaohjelmakomponentit ja rootkit-haittaohjelmat. (Mohanta & Saldanha 2020, 4).

Yleisesti ottaen muistianalyysiä suoritetaan ns. tilanneselvityksissä, eli kun ohjelma on jo tehnyt tuhojaan ja muistista on saatu kaappaus, mutta sitä voidaan käyttää myös tehokkaasti dynaamisen analyysin työkaluna. Työssä suoritetaan esimerkkimuistianalyysejä kappaleessa 6.

4 ANALYYSIA VAIKEUTTAVAT TEKIJÄT

Haittaohjelmien analyysin ja puolustusmekanismien parantuessa myös haittaohjelmat ovat kehittyneet niiden mukana. Nykypäivänä monet haittaohjelmat käyttävät erilaisia haittaohjelmien analyysiä vaikeuttavia metodeja, joita tarkastellaan tässä kappaleessa.

4.1 Obfuskaatio

Obfuskointi viittaa toimintaan, jossa tärkeää tietoa yritetään piilotella, tekemällä siitä vaikeasti ymmärrettävää. Haittaohjelmien kontekstissa tämä tarkoittaa enimmäkseen tapoja, jotka johtavat haittaohjelmien analyysin vaikeuttamiseen, ja sen kautta lisäävät todennäköisyyttä siitä, että ohjelman toimintaa ei kyetä mallintamaan. Näitä tapoja voivat olla esimerkiksi tiedostojen pakkaaminen, ja salaus. Salauksella tarkoitetaan kryptausta, ja koodausta ja näillä metodeilla yritetään piilottaa tietoa, kuten komentokeskuksen kanssa käyty verkon välinen kommunikaatio, haittaohjelman käyttämien konfiguraatiotiedostojen, tai merkkijonojen sekoittaminen. (Monnappa 2018, 8.)

Myös suoritettavaa koodia voidaan obfuskoida, joka on yleistä esimerkiksi komentoja sisältävissä skripteissä. Kuvioissa 12. ja 13. on esitetty esimerkki obfuskoidusta JavaScript-koodista.

```
1 // Paste your JavaScript code here
2 function hi() {
3   console.log("Hello World!");
4 }
5 hi();
```

KUVIO 12. Selkokielinen JavaScript-koodi.

```
(function(_0x544253,_0x12ecad){var _0x5b2279=_0x2c2a,_0x2cfbc4=_0x544253();while(!!
[]){try{var _0x393bae=parseInt(_0x5b2279(0x173))/0x1*(parseInt(_0x5b2279(0x175))/0x2)+-
parseInt(_0x5b2279(0x172))/0x3*(parseInt(_0x5b2279(0x174))
/0x4)+parseInt(_0x5b2279(0x16e))/0x5+parseInt(_0x5b2279(0x16f))
/0x6+parseInt(_0x5b2279(0x16c))/0x7+parseInt(_0x5b2279(0x16d))/0x8+-
parseInt(_0x5b2279(0x171))/0x9;if(_0x393bae===_0x12ecad)break;else _0x2cfbc4['push']
(_0x2cfbc4['shift']());}catch(_0x5c3bb5){_0x2cfbc4['push'](_0x2cfbc4['shift']());}}}
(_0x2a1b,0xeaa1f));function hi(){var _0x1538fe=_0x2c2a;console[_0x1538fe(0x170)]
(_0x1538fe(0x176));}hi();function _0x2c2a(_0x5340ed,_0x33d760){var
_0x2a1b3d=_0x2a1b();return _0x2c2a=function(_0x2c2a1e,_0x3cbdfd)
{_0x2c2a1e=_0x2c2a1e-0x16c;var _0x2bdfe4=_0x2a1b3d[_0x2c2a1e];return
_0x2bdfe4;},_0x2c2a(_0x5340ed,_0x33d760);}function _0x2a1b(){var _0x186e25=
['4412964lASKAR','log','14882994fZiKYl','1395mtiBgg','2981XbdYeE','3228hKAVEr','374aup
aMZ','Hello\x20World!','637805qyUvCZ','3043936GCflXA','6127140EhdMpS'];
_0x2a1b=function(){return _0x186e25;};return _0x2a1b();}
```

KUVIO 13. Kuvion 11. JavaScript-koodi obfuskoituna.

Obfuskoitu koodi suorittaa täysin saman toimenpiteen kuin selkokielinen koodi, mutta sen toimintaa on miltei mahdotonta selvittää vain sitä lukemalla. Käännettävän ohjelmointikielen lähdekoodin obfuskointi ajaa saman idean, sen toiminnan mallintaminen on hankalampaa debuggeri- ja decompiler-ohjelmilla, ei-obfuskoituun lähdekoodiin verrattuna.

Kuvion 13. obfuskointi on toteutettu ”obfuscator.io” nimisen verkkosivun avulla. Esimerkkejä salaukseen käytetyistä metodeista ovat esimerkiksi BASE64-, tai XOR-muunnokset.

4.2 Pakkaaminen

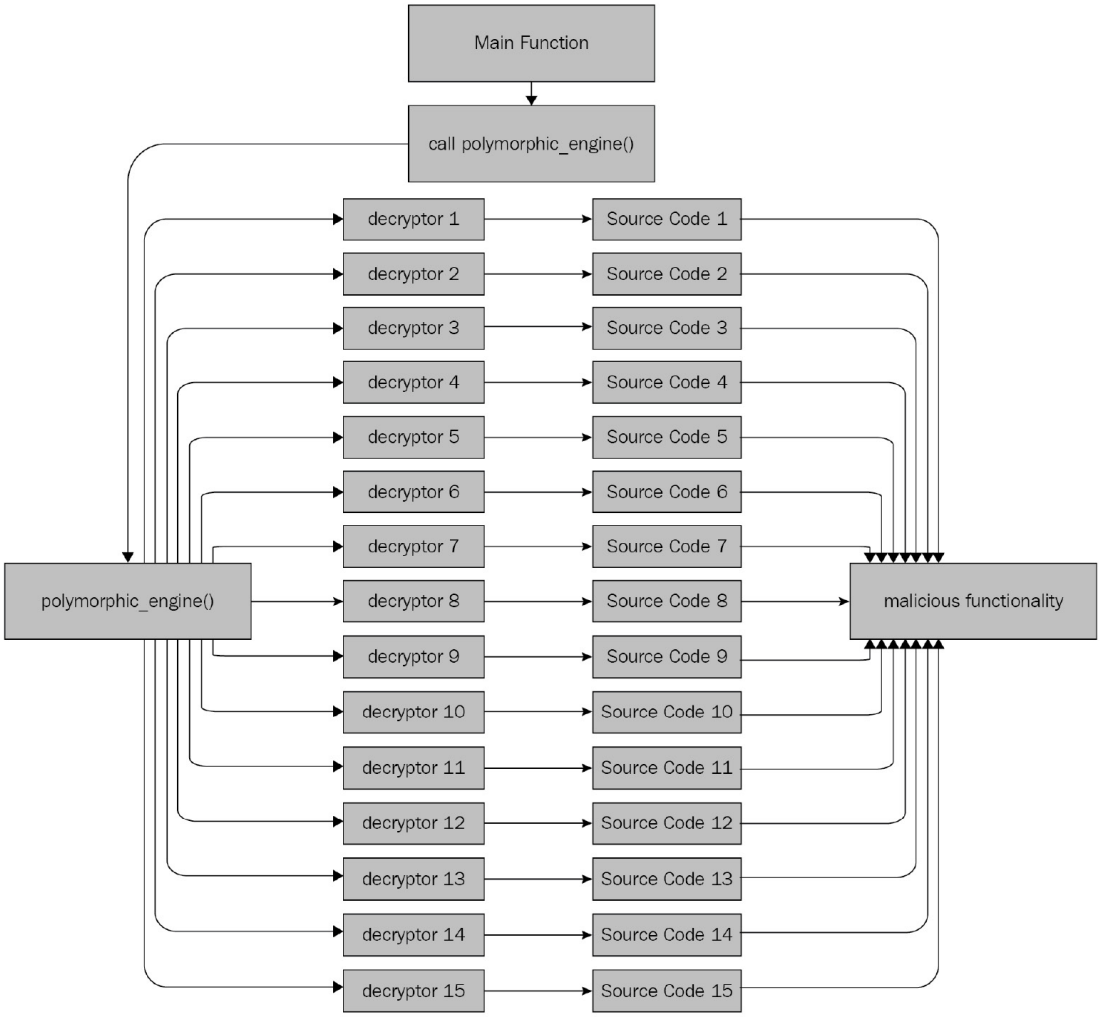
Pakkaaminen on suoritettavan tiedoston kompressoimista tiedostokoon pienentämiseksi, ja sen sisällön salaamiseksi. Pakkaaja sisältää pakattuun ohjelmaan komponentin, joka purkaa tiedoston kompression, kun ohjelma suoritetaan. Pakatun tiedoston staattinen analysointi vaatii aina pakkauksen purkamista, joka voidaan tunnistaa erilaisilla ohjelmistoilla. (Sikorski & Honig 2012, 1.) Myös useat ns. normaalit ja luotettavat ohjelmat käyttävät pakkaamista, joten se ei aina ole merkki ohjelman haitallisuudesta.

4.3 Emulaation välttely, virtuaalikoneen tunnistus

Usein dynaamisessa analyysissä käytetään virtuaalista ympäristöä, turvallisuuden vuoksi. Virtuaaliympäristöä asennettaessa virtuaalikoneelle tulee määrätä tietty määrä prosessointitehoa, muistia ja levytilaa, ja kovinkin usein virtuaalikoneille annetaan vain pieni määrä resursseja. Nykyään jopa halvemmillakin tietokoneilla on käytössään useita prosessien ytimiä ja vähintään 8 Gt muistia, ja tästä syystä haittaohjelmat saattavat suoritushetkellä tunnistaa järjestelmän resurssien avulla, että niitä suoritetaan virtuaaliympäristössä, eivätkä ne siksi suorita haitallisia toimintojaan. Muita tapoja millä haittaohjelmat voivat tunnistaa virtuaaliympäristön ovat esim. prosessien määrä, asennetut ohjelmat (tai niiden puuttuminen), satunnaisten tiedostojen olemassaolo, tai vaikka järjestelmään kirjautumisen jälkeen kulunut aika. (Mohanta & Saldanha 2020, 5.)

4.4 Polymorfismi & metamorfismi

Eräät haittaohjelmien tunnistusta vaikeuttavat tekijät ovat polymorfismi ja metamorfismi. Polymorfismilla tarkoitetaan haittaohjelmaa-, tai -perhettä, joka sisältää monia eri salattuja lähdekoodiversioita, ja niitä vastaavia purkajia. Joka kerta kun haittaohjelma suoritetaan, se valitsee uuden purkajan ja sitä vastaavan lähdekoodin. Ohjelman haitalliset toiminnot ovat aina samat, mutta se kykenee näin välttämään esim. antiviruksella suoritetun staattisen tunnistuksen. Polymorfinen koodi on siis ns. useamuotoista. (Yehoshua & Kosayev 2021, 5.) Polymorfisen koodin toiminta on esitetty kuviossa 14.



KUVIO 14. Polymorfisen koodin toiminta (Yehoshua & Kosayev 2021, 5).

Metamorfinen koodi on koodia, joka muuttaa itseään, joka kerta kun se suoritetaan, siten että ohjelman toiminto pysyy aina samana. Ohjelma saattaa lisätä itselleen uusia toimintoja, jotka ovat merkityksettömiä, tai jotka eivät tee yhtään mitään. Metamorfinen ohjelma eroaa polymorfismisesta ohjelmasta siten, että se ei sisällä itsestään useita versioita, tai käytä useampia pakkaajia, tai salaustapoja. (Yehoshua & Kosayev 2021, 5.)

Molempia tekniikoita käytetään staattisen tunnistuksen välttämiseksi, mutta koska ohjelmien toiminnallisuus pysyy aina samana, ne voidaan silti tunnistaa käyttäytymiseen pohjautuvalla tunnistuksella (Yehoshua & Kosayev 2021, 5).

4.5 Stealth

Jotta haittaohjelma olisi hyödyllinen, sen tulee piiloutua järjestelmän käyttäjältä ja mahdollisilta suojaukseen käytetyiltä ohjelmistoilta, poikkeuksena tähän on esim. kiristyshaittaohjelmat. Määritelmän ”stealth”, alle voidaan luokitella tapoja, joilla haittaohjelmat yrittävät pysyä piilossa. (Mohanta & Saldanha 2020, 3.) Yksi haittaohjelmien käyttämä tekniikka tämän tavoitteen saavuttamiseksi on koodi-injektiot.

4.6 Koodi-injektiot

Koodi-injektiot ovat tekniikoita, joilla prosessi ”syöttää”, tai injektoi osan, tai kaiken omasta koodistaan toiseen järjestelmässä olevaan prosessiin suorittamista varten. Koodi-injektioiden tarkoitus on piilotella haitallista ohjelmaa käyttäjiltä ja automaattisilta puolustusmekanismeilta, tai muuttaa toisen prosessin tai itse käyttöjärjestelmän toimintaa. (Mohanta & Saldanha 2020, 3.)

Käyttäjä, ja varsinkin kokenut sellainen, osaa usein tunnistaa oudot tietokoneella suoritettavat prosessit, ja siksi koodi-injektion avulla haittaohjelma kykenee välttämään helpoimmat havainnointiyritykset. Haitallinen prosessi voi syöttää koodiaan esimerkiksi *”svchost.exe”*-prosessiin, joka on Windowsille ominainen, jolloin käyttäjä ei enää näe outoja prosesseja tarkkaillessaan Windowsin tehtävienhallintaa. (Mohanta & Saldanha 2020, 3.)

Jos järjestelmässä tai verkossa on käytössä palomuuri, haittaohjelma ei välttämättä pysty käyttämään verkkoyhteyttä hyväkseen, jos se yrittää käyttää sitä oman prosessinsa kautta. Jälleen koodi-injektiolla Windowsin ominaiseen prosessiin, palomuurin säännöt voidaan välttää, sillä Windowsin omia prosesseja ei aina ole otettu palomuurin säännöissä huomioon. (Mohanta & Saldanha 2020, 3.)

Muuttamalla toisen prosessin tai käyttöjärjestelmän toimintaa haittaohjelma voi mm. puolustautua sen poistamista vastaan. Win32 API tarjoaa ohjelmille DeleteFile-funktion, jonka avulla voidaan poistaa levyltä tiedostoja. Tätä funktiota käyttävät mm. antivirusohjelmat, ja sitä muokkaamalla haittaohjelma voi varmis-

taa, ettei sitä itseään voida poistaa tätä funktiota käyttämällä. (Mohanta & Saldanha 2020, 3.)

Koodi-injektiometodeja on erilaisia, joista alla on esitetty muutamia:

4.6.1 DLL-injektio

DLL-injektiolla tarkoitetaan tapahtumaa, jossa haittaohjelman suorituksen aikana luoma haitallinen prosessi kirjoittaa kiintolevylle haitallista koodia sisältävän .dll-tiedoston, valitsee kohdeprosessin, ja lataa tämän tiedoston polun kohdeprossin virtuaalimuistiin. Varmistaakseen haitallisen koodin suorittamisen, haittaohjelma luo kohdeprosessiin uuden säikeen, joka lukee haitallisen kirjaston sisältämää koodia. (Hosseini 2017.) Nämä toiminnot suoritetaan Win32 API-kutsujen avulla seuraavasti:

Haittaohjelman prosessi luo ensin ”kahvan” (engl. handle) haluttuun kohdeprosessiin ”OpenProcess” API-kutsulla. Kutsumalla ”VirtualAllocEx” API:a haittaohjelma varaa kohdeprosessiin virtuaalimuistia, johon se voi kirjoittaa haitallisen kirjaston polun, joka kirjoitetaan tähän muistialueeseen ”WriteProcessMemory” API:lla. ”CreateRemoteThread” API:n avulla kohdeprosessille varataan uusi säie, joka lataa haitallisen kirjaston ”LoadLibrary”-kutsulla. Ladattuaan kirjaston, käyttöjärjestelmä automaattisesti suorittaa sen sisältämän koodin ”DllMain”-funktiolla. (Hosseini 2017; Monnappa 2018.)

4.6.2 PE-injektio

PE-injektio on toiminnaltaan samanlainen koodi-injektiotekniikka kuin dll-injektio, mutta haitallinen koodi ladataan suoraan prosessin muistiin ”WriteProcessMemory”-kutsulla, ilman levylle kirjoitettua .dll-tiedostoa. (Hosseini 2017.)

4.6.3 Ontto prosessi

”Process hollowing” on koodi-injektiotekniikka, jossa haitallinen prosessi avaa uuden instanssin jostain tunnetusta kohdeprosessista ns. keskeytetyssä, eli ”suspend”-tilassa, ja korvaa kohdeprosessin muistialueen omalla koodillaan.

Haittaohjelma siis tekee kohdeprosessista ”onton”, ja vaihtaa sen sisällön haitalliseksi. ”CREATE_SUSPENDED”-kutsulla luodaan uusi prosessi, jota ei aloita suorittamaan ennen kuin ”ResumeThread” API:a kutsutaan. Prosessin muistialue voidaan tyhjentää ”ZwUnmapViewOfSection”- tai ”NtUnmapViewOfSection”-kutsuilla. Jälleen ”VirtualAllocEx”-kutsua käytetään kirjoittamaan uusi haitallinen koodi prosessin muistialueeseen, ja kun keskeytetystä tilasta poistutaan, prosessi suorittaa haitallista koodia, jota sen muistialue sisältää. (Hosseini 2017.)

4.7 API-koukut

Yksi koodi-injektioiden tarkoituksista on ns. API-kutsujen ”koukuttaminen”. Tämä tarkoittaa tilannetta, jossa koodia-injektoidaan kohdeprosessiin, jotta sen API-kutsut voidaan kaapata ja ohjata prosessiin injektoidun, haitallisen koodin käsiteltäväksi. API-kutsuja usein muutetaan esim. vaihtamalle niille syötettyjä parametrejä, jonka jälkeen ne ohjataan eteenpäin oikeaan kohteeseen, jotta kaikki vaikuttaisi normaalilta. (Mohanta & Saldanha 2020, 3.) Esimerkkinä API-kutsun koukuttamisesta on aikaisemmin, kappaleen 4.6 alussa mainittu haittaohjelma, joka koukuttaa kohdeprosessin kutsun DeleteFile API:hin, ja yrittää välttää tiedostonsa poistamista.

Käyttäjätilan API-koukut tapahtuvat yleisesti kahdella eri metodilla, joista kerrotaan seuraavaksi.

4.7.1 IAT-koukut

Kappaleessa 3.6.1 kerrottiin, että kutsutun API:n muistiosoite löytyy IAT-taulusta. IAT-koukulla tarkoitetaan tilannetta, jossa haittaohjelma on injektoinut koodia kohdeprosessiin jollain koodi-injektointimetodilla, käynyt läpi prosessin muistialueessa sijaitsevan IAT-taulun, ja valinnut jonkun API-kutsun koukuttamisen kohteeksi. Itse koukutus tapahtuu muokkaamalla tuon API-kutsun muistiosoitetta IAT-taulussa sellaiseksi, joka ohjaa haittaohjelman omaan haitalliseen koodiin. Tämä haitallinen koodi, kuten aikaisemmin mainittiin, saattaa muuttaa kutsua ja sitten ohjata sen takaisin alkuperäiseen osoitteeseen. (Monnappa 2018, 8.)

IAT-koukkujen luominen ei ole mahdollista, jos ohjelma linkittää käyttämänsä kirjastot suorituksen aikana, tai jos Windows käyttöjärjestelmä on 64-bittinen. 64-bittisessä Windowsissa on implementoitu ”PatchGuard”-niminen toiminto, joka estää näiden taulujen tietojen muuttamisen. IAT-koukut ovat myös helppo havaita IAT-taulun osoitteita tarkkailemalla. Esimerkiksi jos DeleteFile-kutsun osoite IAT-taulussa ei johda *Kernel32.dll*-tiedoston muistialueeseen, tiedetään, että osoitetta on muutettu. (Monnappa 2018, 8.)

4.7.2 Inline-koukut

Inline-koukuilla viitataan tilanteeseen, jossa haittaohjelma on muokannut itse kutsuttavan API:n koodin ensimmäisiä tavuja hyppykäskyllä (jmp assemblynä), joka ohjaa suorituksen haitalliseen koodiin, palatakseen myöhemmin takaisin alkuperäiseen funktioon. Joskus haittaohjelma voi hyppykäskyn sijaan käyttää ”call”-, tai ”push”- ja ”ret”-käskyjen yhdistelmää ohjatakseen koodin suorittamista, välttääkseen tunnistusta niiden ohjelmien osalta, jotka etsivät vain hyppykäskyjä API-funktioista. (Monnappa 2018, 8.)

5 ESIMERKKIANALYYSEJÄ

Tässä kappaleessa kerrotaan esimerkkianalyyseistä, joita työn aikana suoritettiin virtuaalisessa laboratorioympäristössä oikeilla haittaohjelmanäytteillä.

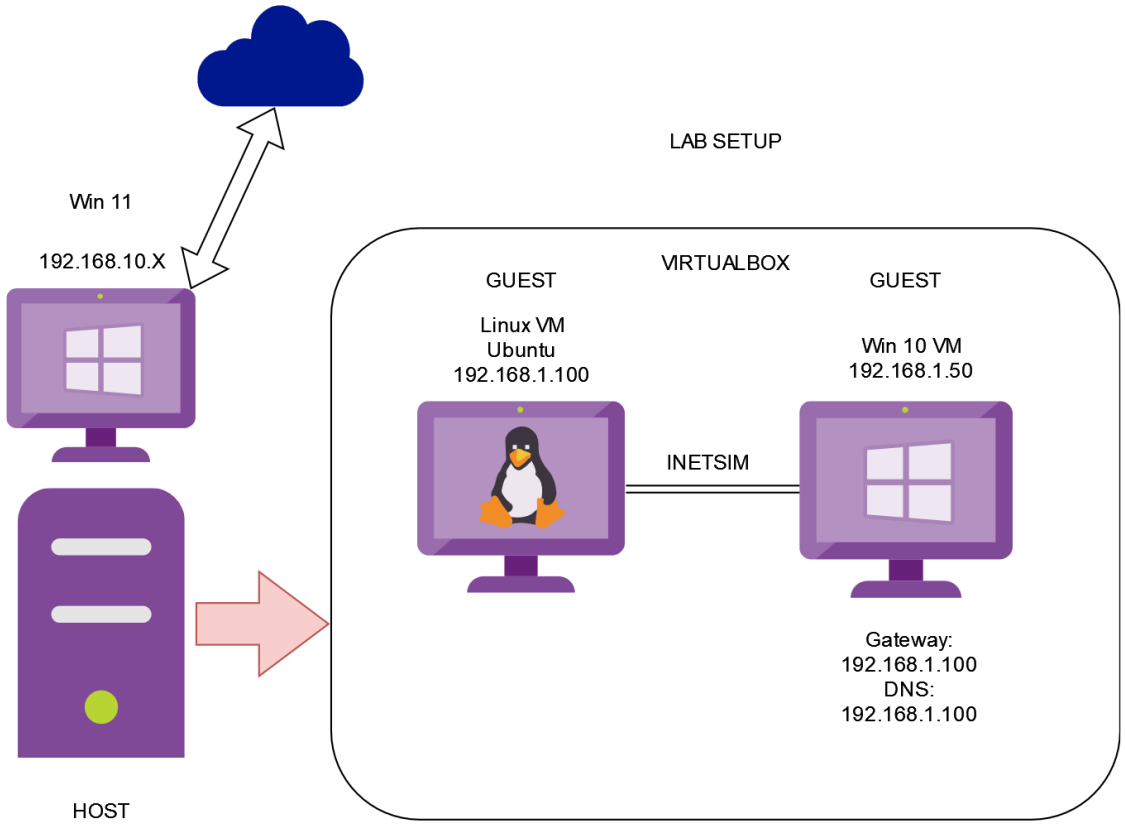
5.1 Laboratorioympäristö

Laboratorioympäristönä työssä käytettiin kahta Virtualbox-alustalle asennettua virtuaalikonetta, jotka eristettiin omaan verkkoonsa. Guest-käyttöjärjestelminä toimivat Windows 10, ja Ubuntu 20.04.4 LTS Linux-distribuutio, jota pyöritettiin Windows 11 Host-käyttöjärjestelmän päällä. Ohjeistuksena käytettiin Monnappa:n ”Learning Malware Analysis”-kirjaa.

Haittaohjelmanäytteet kerättiin erinäisistä lähteistä internetistä, ja ne suoritettiin Windows 10-virtuaalikoneessa, josta Windows Defenderin tuoma automaattinen virusturva kytkettiin pois päältä. Linux-käyttöjärjestelmää käytettiin dynaamisiin analysointitarkoituksiin, kuten verkon simulointiin ja verkkoliikenteen monitorointiin, sekä muistianalyysiin. Haittaohjelmien staattinen analysointi suoritettiin eri ohjelmistoilla Windows 10-virtuaalikoneessa, ennen haitallisten näytteiden suorittamista.

Käyttöjärjestelmien ja käytettävien ohjelmistojen asennuksen jälkeen, molemmista virtuaalikoneista otettiin kopiot puhtaan tilan palauttamista varten, Virtualboxin snapshot-toiminnolla.

Laboratorioympäristön rakenne on esitetty kuviossa 15.



KUVIO 15. Laboratorioympäristön rakenne.

Laboratorioympäristön asennus on esitetty liitteessä 1.

5.2 Työkalut

Windows-koneelle asennettiin seuraavat ohjelmat:

ExeinfoPE Ohjelmien pakkauksen/obfuskoinnin tunnistamista varten.

DumpIt Muistikaappauksia varten.

Floss Obfuskoitujen merkkijonojen lukemista varten.

Ghidra Ohjelman dissassemblya ja decompilaamista varten.

HxD Hex-editori, jolla voidaan tarkastella ohjelmatiedostoja, sekä käynnissä olevien prosessien muistia.

IDA Ohjelman dissassemblya varten, hieman helpompi käyttää kuin Ghidra.

Noriben Python skripti, joka helpottaa Process Monitor-ohjelmalla kerättyjen lokien lukemista.

PEstudio Suoritettavien ohjelmien PE-tiedostoformaatin tarkkailua varten.

ProcessHacker 2 Prosessien tarkkailua varten.

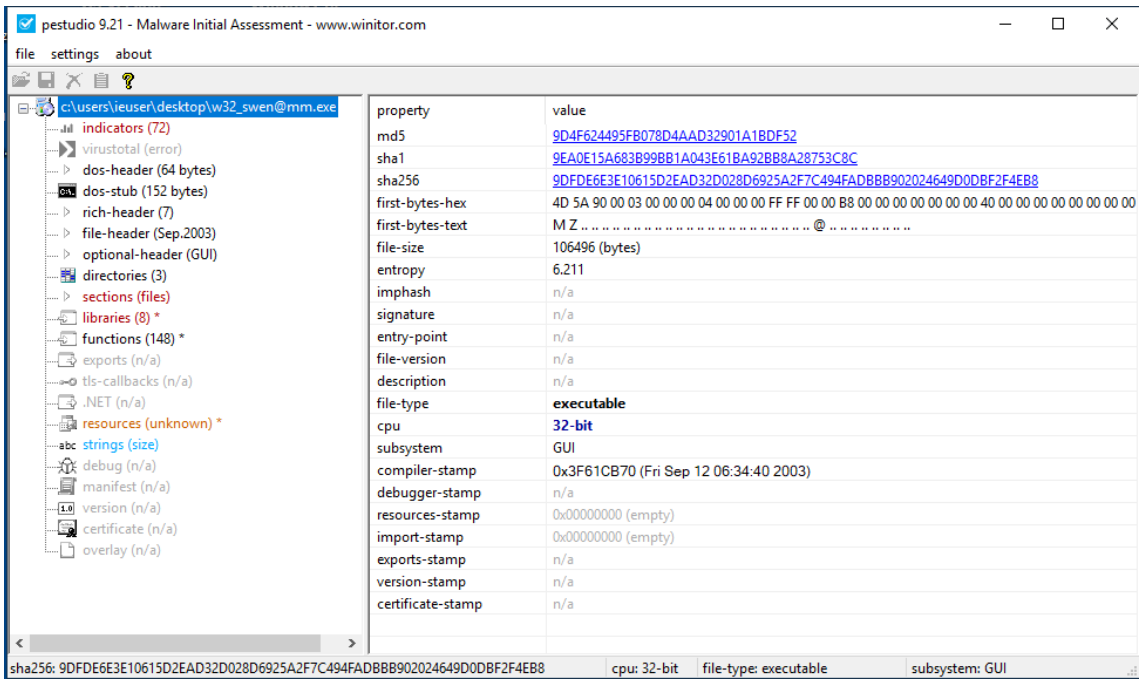
SysInternasSuite Paketti, joka sisältää Windowsin diagnostiikkatyökaluja, kuten *strings.exe*, jolla tarkkaillaan merkkijonoja ja ProcessMonitor, joka esittää prosessien toimintaan liittyvää tietoa.

5.3 Esimerkki 1, W32.Swen

Ensimmäisessä analyysissa tarkasteltiin ”Swen” -nimistä matoa. Haittaohjelma ladattiin ”The Zoo”-nimisestä Github-reposta, joka sisältää paljon eri haittaohjelmanäytteitä analyysi- ja opetustarkoituksia varten. Ladatun tiedoston nimi oli *w32_swen@mm.exe* ja ainakin sen pikkukuvakkeen perusteella ohjelma esitti jonkinlaista asennuspakettia.

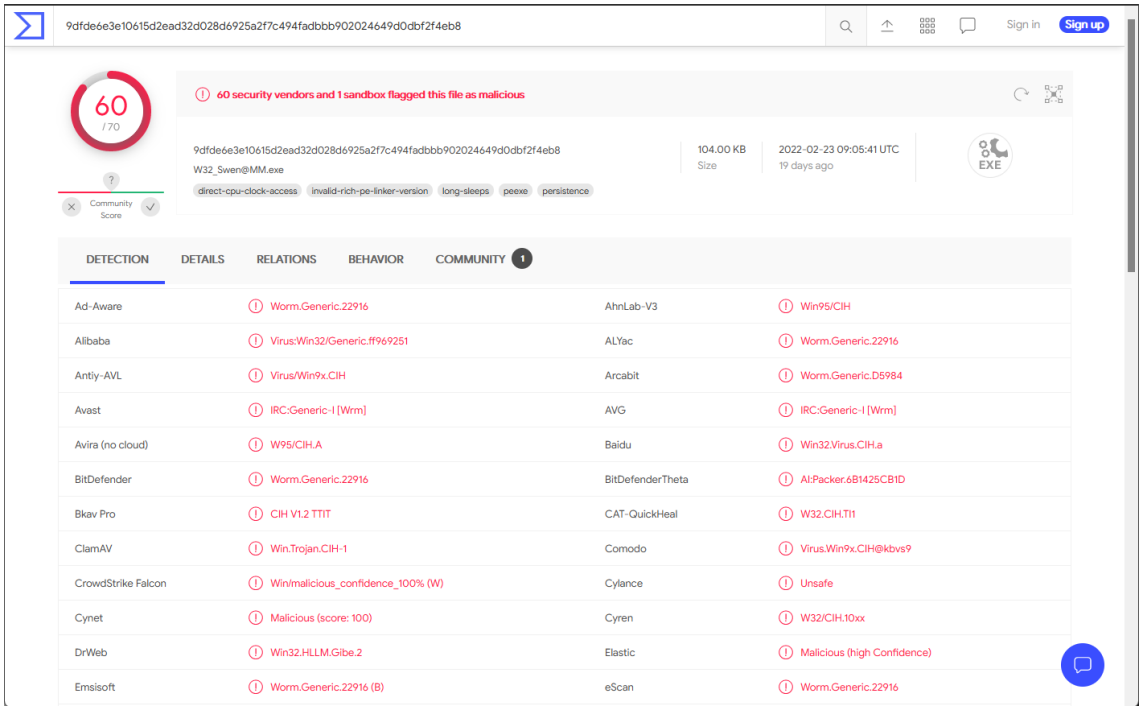
5.3.1 Staattinen analyysi

Kappaleen staattinen analyysi aloitettiin vertaamalla tiedoston MD5 -summaa Virustotalin tietokantaan. Analyysia tehdessä tiedettiin, että tarkasteltava ohjelma on haitallinen, ja jo entuudestaan tunnettu, mutta tämä on järkevä toimenpide oikeissa analyysitilanteissa, joten se suoritettiin joka tapauksessa. Tässä tapauksessa tiedoston MD5-summa saatiin selville avaamalla se PEstudio-ohjelmalla. Tämä on esitetty kuviossa 16. Tuloksista selvisi myös, että ohjelma on käännetty 12.9.2003.



KUVIO 16. Swen-madon suoritettava tiedosto avattu PEstudio-ohjelmalla.

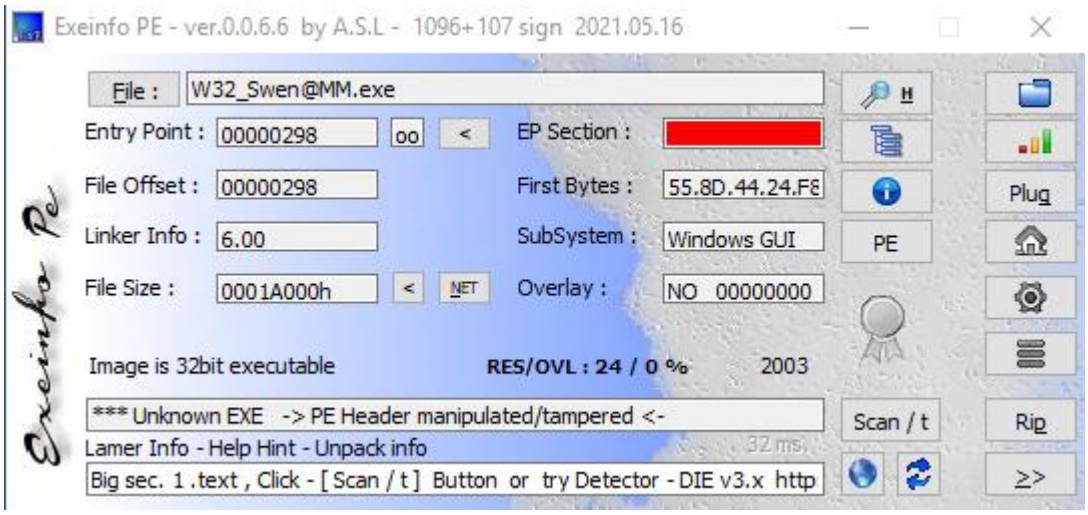
Ohjelmalla nähtiin, että tiedoston MD5-summa on: *9D4F624495FB078D4AAD32901A1BDF52*. Tämä summa syötettiin Virustotal:n ”Search”-kenttään. Haun tuloksista huomataan, että 60 antivirus skanneria 70:stä tunnistaa kyseisen tiedoston haitalliseksi, pelkän tiivisteen avulla. Tulokset on esitetty kuviossa 17.



KUVIO 17. Tiedoston MD5 -summa syötetty Virustotaliin.

Jos tiedoston haitallisuudesta ei olisi tiedetty etukäteen, Virustotalin tulokset kertovat jo tarpeeksi tämän varmistamiseksi.

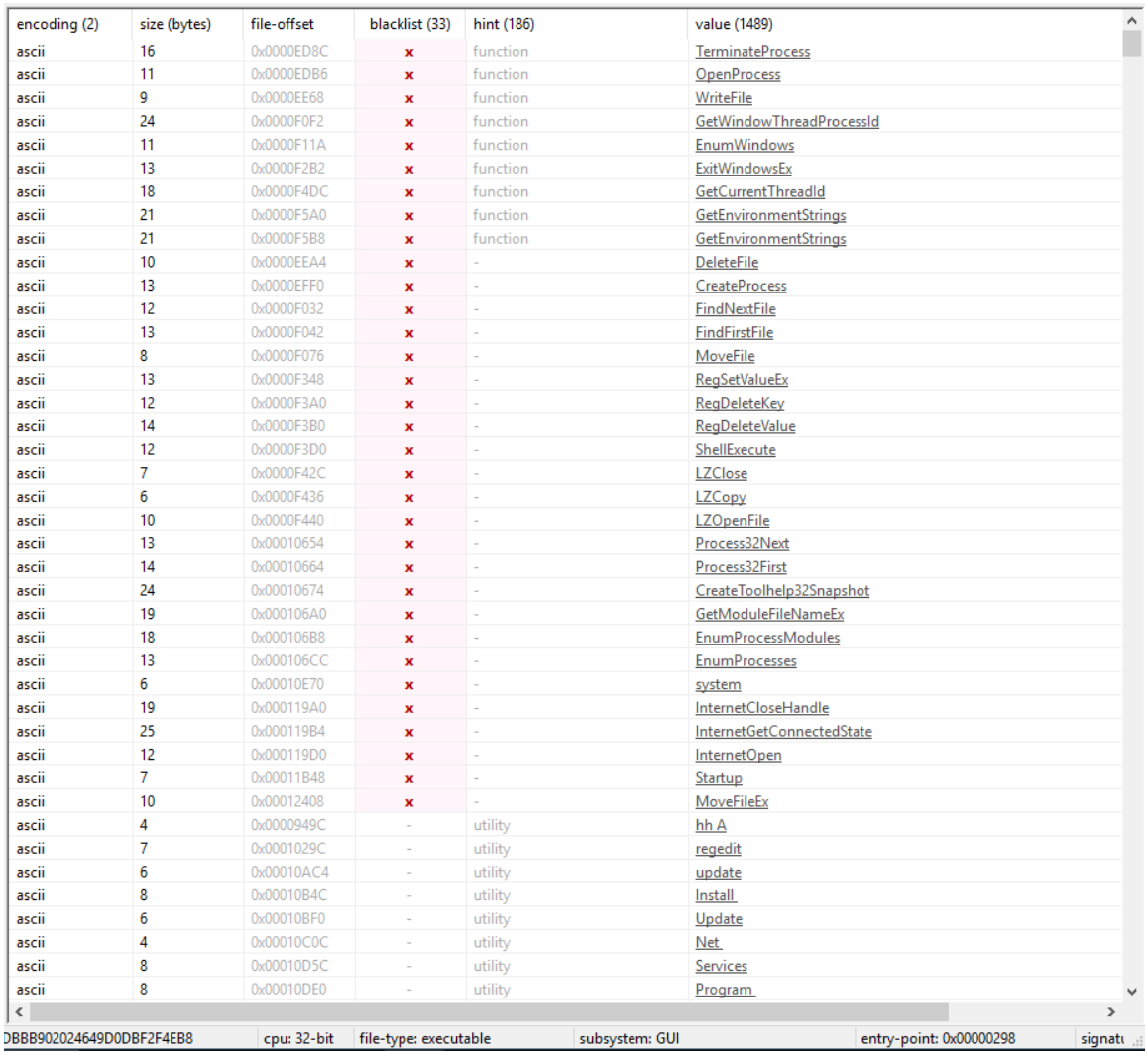
Staattista analyysia jatkettiin tarkistamalla- onko tiedosto pakattu- ExeinfoPE-ohjelmalla. Tämä on esitetty kuviossa 18.



KUVIO 18. Haittaohjelmanäyte avattu ExeinfoPE-ohjelmalla.

ExeinfoPE-ohjelma ilmoittaa tiedoston olevan sille tuntematon suoritettava tiedosto (”Unknown EXE”) ja että sen PE-otsikkoa on muunneltu. Ohjelma ei kuitenkaan ilmoita pakkauksen olemassaolosta. Tiedostolla ei ole myöskään digitaalista allekirjoitusta.

Näytteestä tarkasteltiin seuraavaksi merkkijonoja PEstudiolla. PEstudio sisältää oman merkkijonojen ”mustan listansa”, ja se ilmoittaakin, että tässä näytteessä on 33 merkkijonoja, jotka kuuluvat tuohon listaan. Tämä on esitetty kuviossa 19.



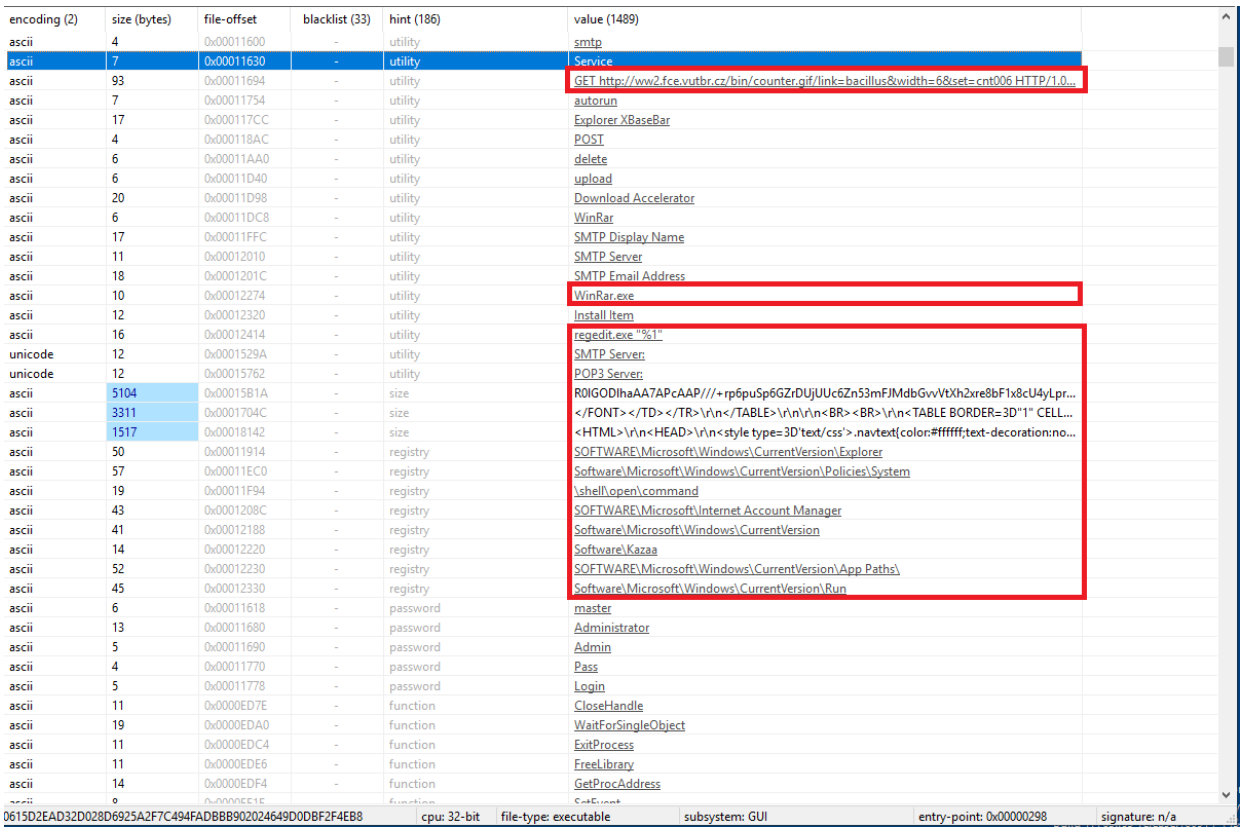
encoding (2)	size (bytes)	file-offset	blacklist (33)	hint (186)	value (1489)
ascii	16	0x0000ED8C	x	function	TerminateProcess
ascii	11	0x0000EDB6	x	function	OpenProcess
ascii	9	0x0000EE68	x	function	WriteFile
ascii	24	0x0000F0F2	x	function	GetWindowThreadProcessId
ascii	11	0x0000F11A	x	function	EnumWindows
ascii	13	0x0000F2B2	x	function	ExitWindowsEx
ascii	18	0x0000F4DC	x	function	GetCurrentThreadId
ascii	21	0x0000F5A0	x	function	GetEnvironmentStrings
ascii	21	0x0000F5B8	x	function	GetEnvironmentStrings
ascii	10	0x0000EEA4	x	-	DeleteFile
ascii	13	0x0000EFF0	x	-	CreateProcess
ascii	12	0x0000F032	x	-	FindNextFile
ascii	13	0x0000F042	x	-	FindFirstFile
ascii	8	0x0000F076	x	-	MoveFile
ascii	13	0x0000F348	x	-	RegSetValueEx
ascii	12	0x0000F3A0	x	-	RegDeleteKey
ascii	14	0x0000F3B0	x	-	RegDeleteValue
ascii	12	0x0000F3D0	x	-	ShellExecute
ascii	7	0x0000F42C	x	-	LZClose
ascii	6	0x0000F436	x	-	LZCopy
ascii	10	0x0000F440	x	-	LZOpenFile
ascii	13	0x00010654	x	-	Process32Next
ascii	14	0x00010664	x	-	Process32First
ascii	24	0x00010674	x	-	CreateToolhelp32Snapshot
ascii	19	0x000106A0	x	-	GetModuleFileNameEx
ascii	18	0x000106B8	x	-	EnumProcessModules
ascii	13	0x000106CC	x	-	EnumProcesses
ascii	6	0x00010E70	x	-	system
ascii	19	0x000119A0	x	-	InternetCloseHandle
ascii	25	0x000119B4	x	-	InternetGetConnectedState
ascii	12	0x000119D0	x	-	InternetOpen
ascii	7	0x00011B48	x	-	Startup
ascii	10	0x00012408	x	-	MoveFileEx
ascii	4	0x0000949C	-	utility	hh A
ascii	7	0x0001029C	-	utility	regedit
ascii	6	0x00010AC4	-	utility	update
ascii	8	0x00010B4C	-	utility	Install
ascii	6	0x00010BF0	-	utility	Update
ascii	4	0x00010C0C	-	utility	Net
ascii	8	0x00010D5C	-	utility	Services
ascii	8	0x00010DE0	-	utility	Program

DBBB902024649D0DBF2F4EB8 | cpu: 32-bit | file-type: executable | subsystem: GUI | entry-point: 0x00000298 | signatu

KUVIO 19. Haittaohjelmanäytteen merkkijonoja PEstudiolla tarkasteltuna.

Näistä merkkijonoista huomattiin, että ne olivat WIN32 API-kutsuja. Ohjelma esittäytyi jonkinlaisena asennuspakettina, ohjelman ikonista, eli kuvakkeesta päätellen. Kuitenkin API-kutsuista nähtiin, että ohjelma mm. etsii ympäristön merkkijonoja, poistaa tiedostoja, lopettaa- ja luo uusia prosesseja, muuttaa rekisteritietoja, ja yrittää avata yhteyttä internettiin.

Merkkijonoja selaamalla huomattiin myös, että ohjelma tekee muutoksia rekisterin RUN-avaimeen, yrittää ladata jotain internetistä, sekä sisältää HTML-dataa. Merkkijonoissa viitataan myös Winrar-ohjelmaan, ja sähköposti protokolliin, SMTP3 ja POP3. Tämä on esitetty kuviossa 20.



encoding (2)	size (bytes)	file-offset	blacklist (33)	hint (186)	value (1489)
ascii	4	0x00011600	-	utility	smtp
ascii	7	0x00011630	-	utility	Service
ascii	93	0x00011694	-	utility	GET http://ww2.fce.vutbr.cz/bin/counter.gif/link=bacillus&width=6&set=cnt006 HTTP/1.0...
ascii	7	0x00011754	-	utility	autorun
ascii	17	0x000117CC	-	utility	Explorer XBaseBar
ascii	4	0x000118AC	-	utility	POST
ascii	6	0x00011AA0	-	utility	delete
ascii	6	0x00011D40	-	utility	upload
ascii	20	0x00011D98	-	utility	Download Accelerator
ascii	6	0x00011DC8	-	utility	WinRar
ascii	17	0x00011FFC	-	utility	SMTP Display Name
ascii	11	0x00012010	-	utility	SMTP Server
ascii	18	0x0001201C	-	utility	SMTP Email Address
ascii	10	0x00012274	-	utility	WinRar.exe
ascii	12	0x00012320	-	utility	Install Item
ascii	16	0x00012414	-	utility	regedit.exe "%1"
unicode	12	0x0001529A	-	utility	SMTP Server:
unicode	12	0x00015762	-	utility	POP3 Server:
ascii	5104	0x00015B1A	-	size	R0lGODlhaAA7APcAAP///+rp6puSp6GZrDUjUUc6Zn53mFJMdbGvvVtXh2xre8bF1x8cU4yLpr...
ascii	3311	0x0001704C	-	size	</FONT></TD></TR>\r\n</TABLE>\r\n\r\n \r\n<TABLE BORDER=3D"1" CELL...
ascii	1517	0x00018142	-	size	<HTML>\r\n<HEAD>\r\n<style type=3D'text/css'>.navtext{color:#ffffff;text-decoration:no...
ascii	50	0x00011914	-	registry	SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
ascii	57	0x00011EC0	-	registry	Software\Microsoft\Windows\CurrentVersion\Policies\System
ascii	19	0x00011F94	-	registry	\shell\open\command
ascii	43	0x0001208C	-	registry	SOFTWARE\Microsoft\Internet Account Manager
ascii	41	0x00012188	-	registry	Software\Microsoft\Windows\CurrentVersion
ascii	14	0x00012220	-	registry	Software\Kazaa
ascii	52	0x00012230	-	registry	SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\
ascii	45	0x00012330	-	registry	Software\Microsoft\Windows\CurrentVersion\Run
ascii	6	0x00011618	-	password	master
ascii	13	0x00011680	-	password	Administrator
ascii	5	0x00011690	-	password	Admin
ascii	4	0x00011770	-	password	Pass
ascii	5	0x00011778	-	password	Login
ascii	11	0x0000ED7E	-	function	CloseHandle
ascii	19	0x0000EDA0	-	function	WaitForSingleObject
ascii	11	0x0000EDC4	-	function	ExitProcess
ascii	11	0x0000EDE6	-	function	FreeLibrary
ascii	14	0x0000EDF4	-	function	GetProcAddress

0615D2EAD32D028D6925A2F7C494FADBBB902024649D0DBF2F4EB8 | cpu: 32-bit | file-type: executable | subsystem: GUI | entry-point: 0x00000298 | signature: n/a

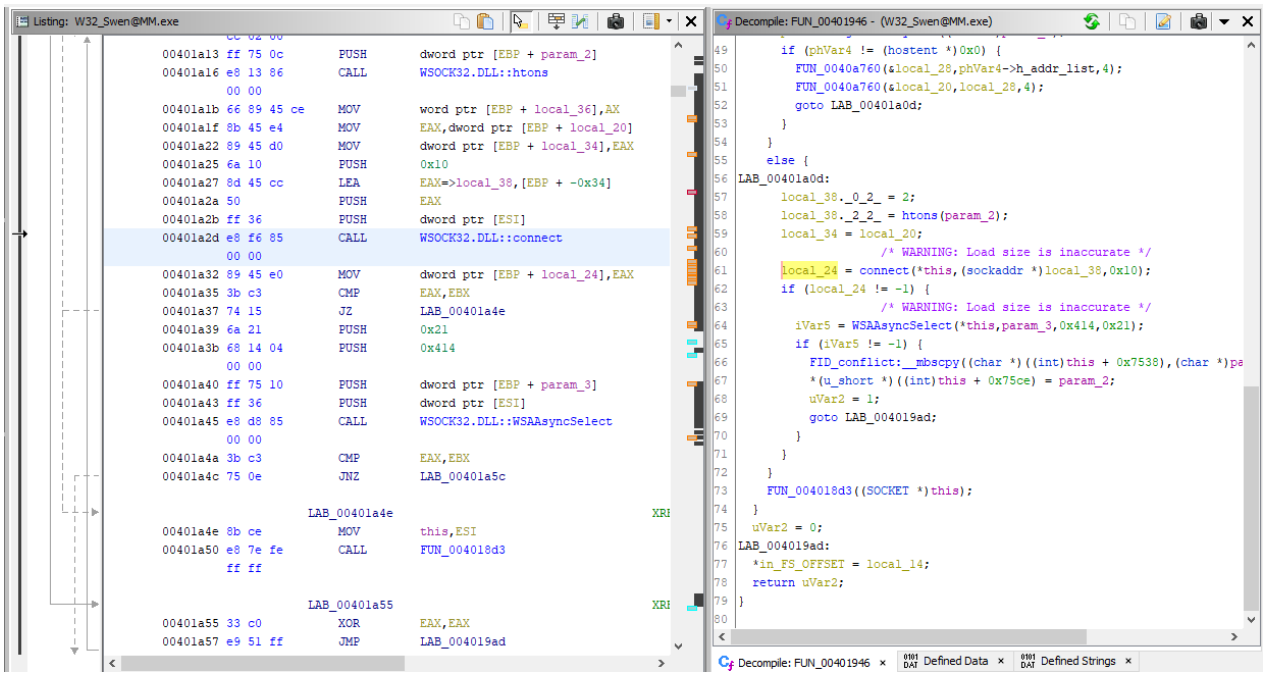
KUVIO 20. Ohjelman sisältämiä epäilyttäviä merkkijonoja.

Merkkijonoista löydettiin myös ns. ”huomiota herättäviä”, otsikoiden tyylisiä ilmaisuja/lauseita, joita oletettiin käytettävän haittaohjelman levittämisen yhteydessä. Niitä ei kuitenkaan sisältönsä vuoksi esitetä tässä työssä.

Merkkijonoissa on viittauksia myös Kazaa p2p-ohjelmaan, joka oli suosittu vertaisverkko-ohjelma 2000-luvun alussa. Tässä vaiheessa analyysia pääteltiin, että ohjelma saattaa levittää itseään sähköpostin ja Kazaa-ohjelma välityksellä.

Merkkijonoja tarkasteltiin vielä Floss-ohjelmalla, mutta se ei tuonut esiin mitään merkkijonoja, joita ei ollut jo nähty.

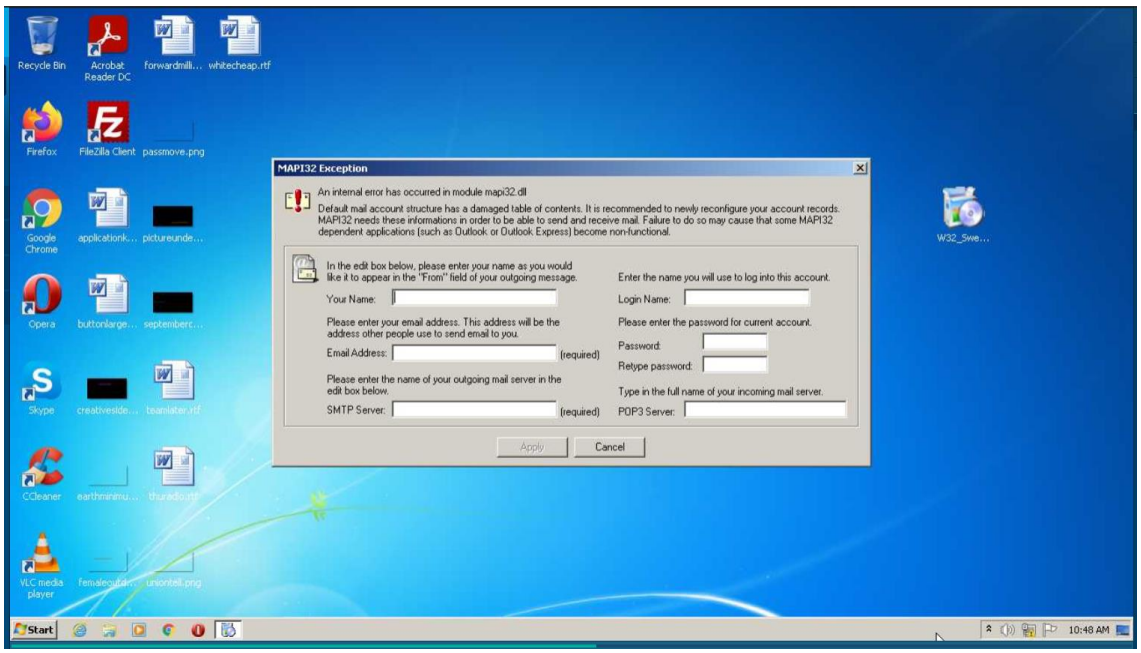
Ohjelma avattiin vielä Ghidra:lla, mutta sen toiminnasta ei osattu kerätä käytännöllistä tietoa, takaisinmallinnuksen hankaluuden vuoksi. Assemblyn ja decompiler:n takaisinkääntämän koodin lukeminen osoittautui odotettua vaikeammaksi. Kuviossa 21 on esitetty näkymä ohjelman eräästä funktiosta, jota tarkasteltiin Ghidran avulla.



KUVIO 21. Ohjelman erään funktion assembly- ja decompiler-esitykset Ghidrassa.

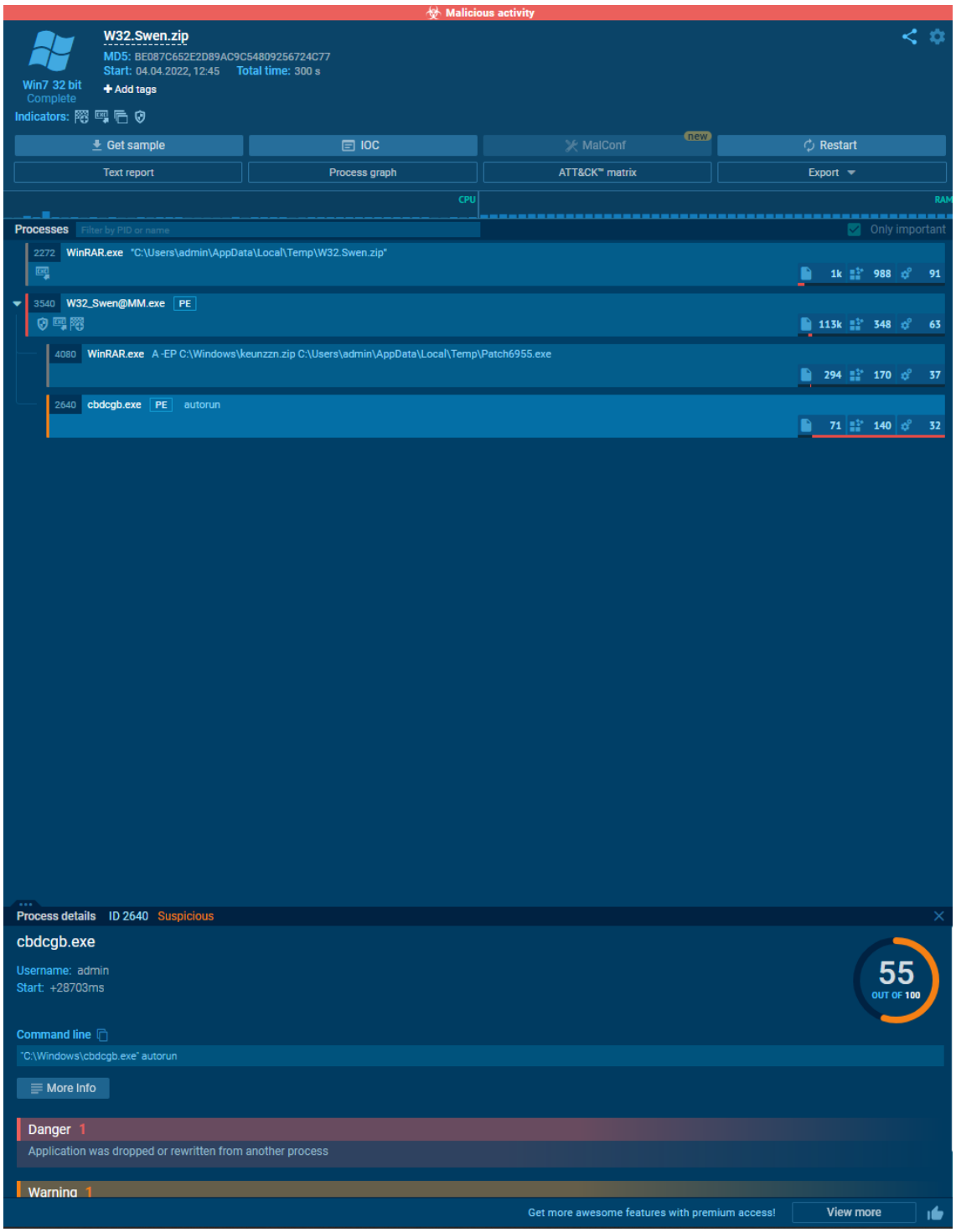
5.3.2 Dynaaminen analyysi

Näytteelle ei onnistuttu tekemään ”kunnollista” dynaamista analyysia, sillä ohjelma ei pystytty suorittamaan nykyisessä käyttöjärjestelmässä. Tämä johtunee näytteen iästä. Näyte ladattiin ANY.RUN-sivustolle, ja se saatiin osittain suoritettua sivuston tarjoamassa analyysiympäristössä. Kuviossa 22 on esitetty sivuston tarjoaman virtuaalikoneen näkymä suorituksen jälkeen.



KUVIO 22. Swen-haittaohjelma suoritettuna ANY.RUN-sivustolla.

Tuloksesta voitiin päätellä, että ohjelma yrittää käyttää sähköpostipalvelua. Analyysiympäristöstä nähtiin, mitä suoritettu ohjelma saa aikaan virtuaalikoneessa. Tämä on esitetty kuviossa 23.



KUVIO 23. Virtuaalikoneen tapahtumat.

Tuloksista nähtiin, että haittaohjelma avasi Winrar-ohjelman, ja purki ”*Patch6955.exe”*-nimisen suoritettavan ohjelman */Temp*-polkuun. Se suoritti myös ”*cbdcgb.exe”*-nimisen ohjelman, jonka se varmaankin on myös purkanut omasta binääristään. Tämä ohjelma pystyttiin lataamaan Virustotaliin ANY.RUN-sivuston kautta, klikkaamalla tätä, ja valitsemalla ”look up on VT”-vaihtoehto. Skannerit tunnistivat tämän ohjelman osaksi Swen-haittaohjelmaa,

samoin kun kuviossa 17 esitetyissä tuloksissa. Tämän enempää ei kuitenkaan ohjelman toiminnasta saatu selville. ANY.RUN virtuaalikoneissa on 5 minuutin maksimi suoritusaika, ainakin ilmaisversiossa, eikä tuossa ajassa tapahtunut muita toimintoja haittaohjelman osalta.

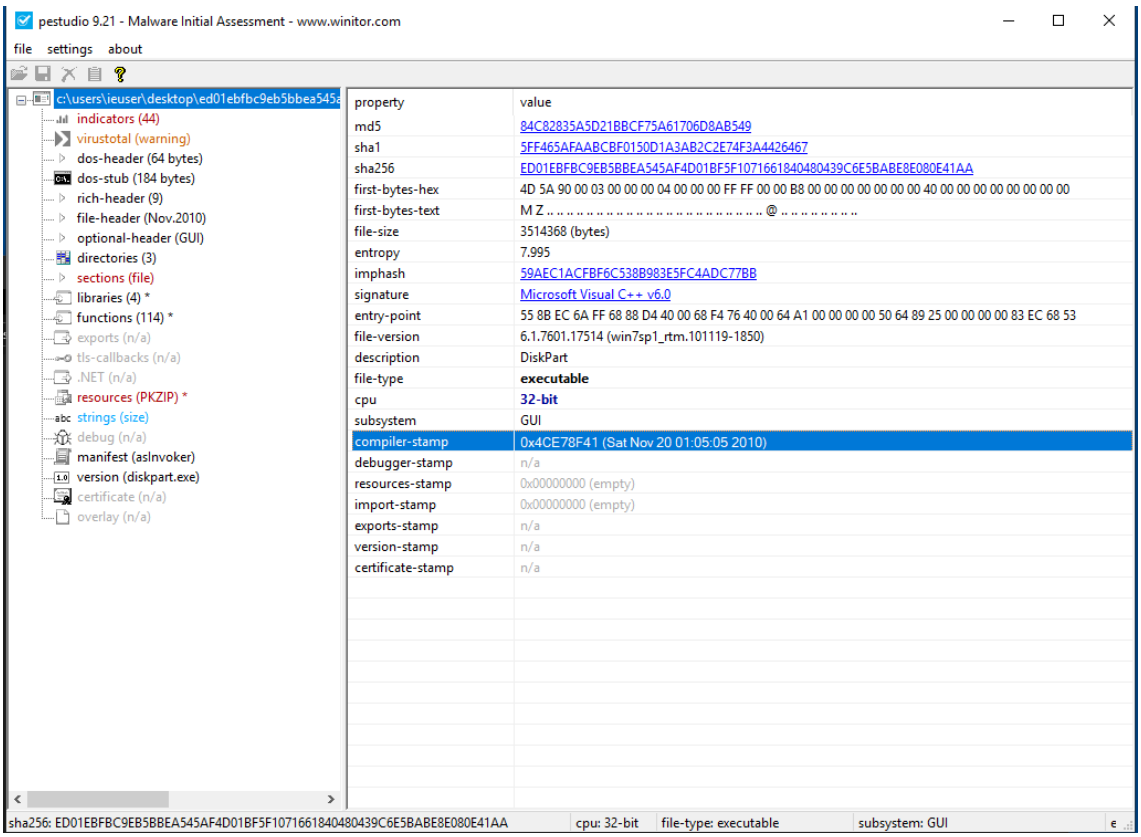
Näytteestä kuitenkin saatiin staattisella analyysilla kerättyä tietoa, jolla ohjelma voitaisiin tunnistaa tulevaisuudessa, tosin jälleen ohjelman iän takia sitä ei varmaankaan tule tapahtumaan – ohjelma toimii tässä vain esimerkkinä.

5.4 Esimerkki 2, WannaCry

Toisena esimerkkinä työssä analysoitiin ”WannaCry” -kiristyshaittaohjelmaa. Haittaohjelmanäyte ladattiin jälleen ”TheZoo” Github-reposta. Ladatun tiedoston nimi oli ”*ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe”.* Ohjelmalla on suoritettavan tiedoston vakioikoni.

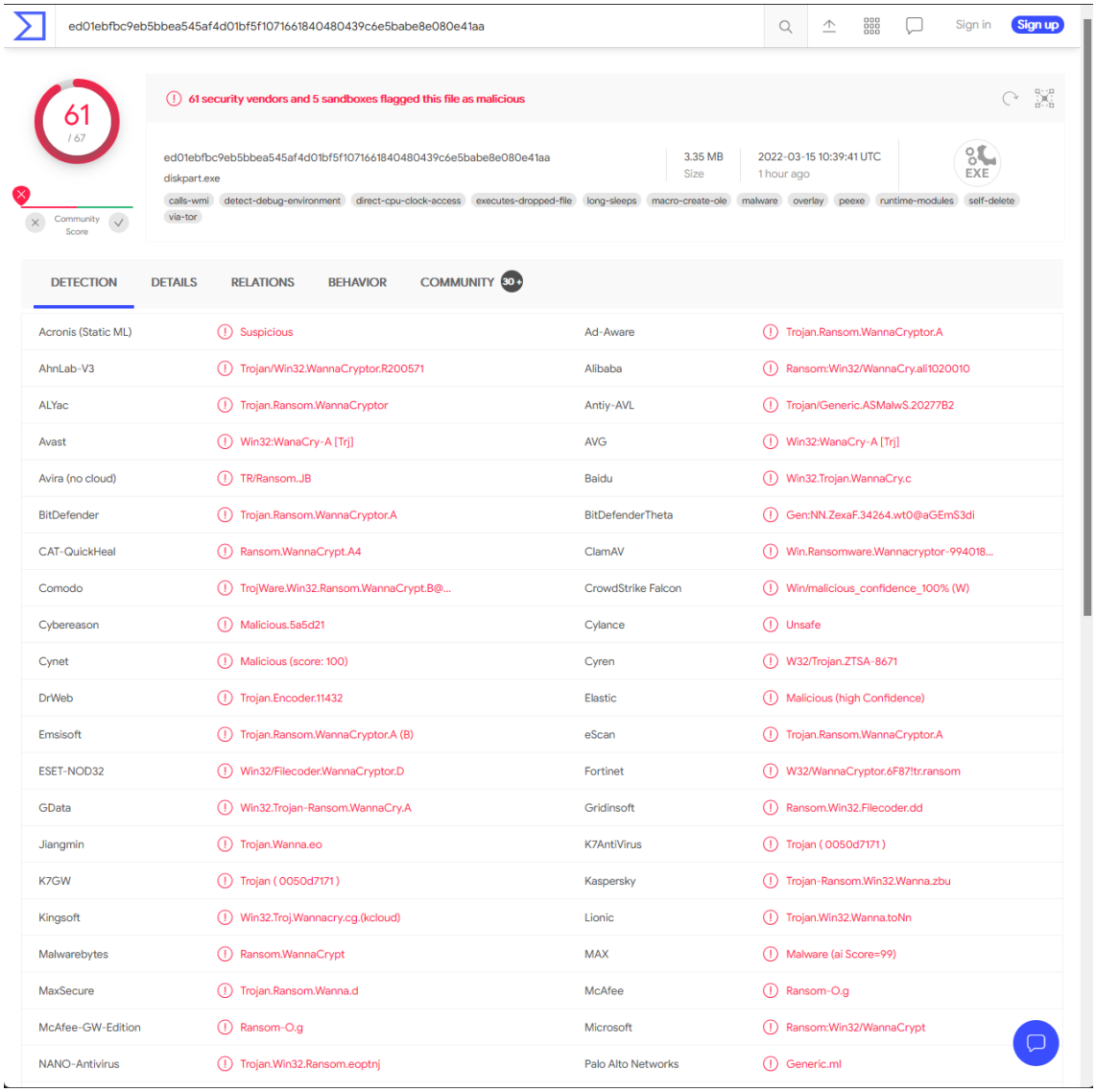
5.4.1 Staattinen analyysi

Toisen näytteen staattinen analyysi suoritettiin samalla tavalla kuin ensimmäisenkin. PEStudiolla tarkkailtuna tiedoston MD5-summa oli ”*84C82835A5D21BBCF75A61706D8AB549”*. Ohjelma oli käännetty 20.11.2010. Tämä on esitetty kuviossa 24.



KUVIO 24. WannaCry-näyte avattuna PEStudiolla.

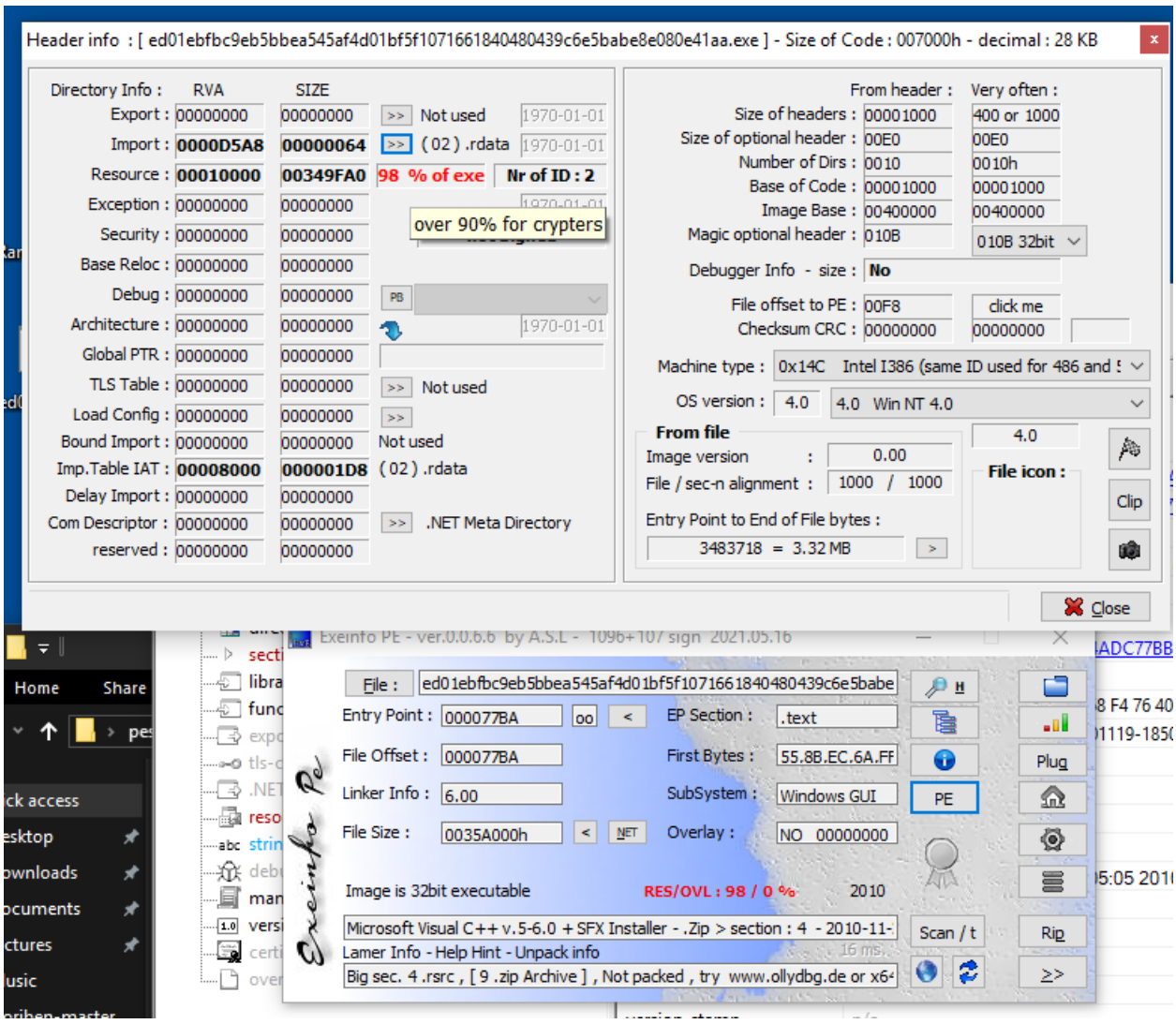
Tiivistesumma syötettiin jälleen Virustotaliin. Tulokset on esitetty kuviossa 25.



KUVIO 25. Virustotalin tulokset WannaCry-näytteelle.

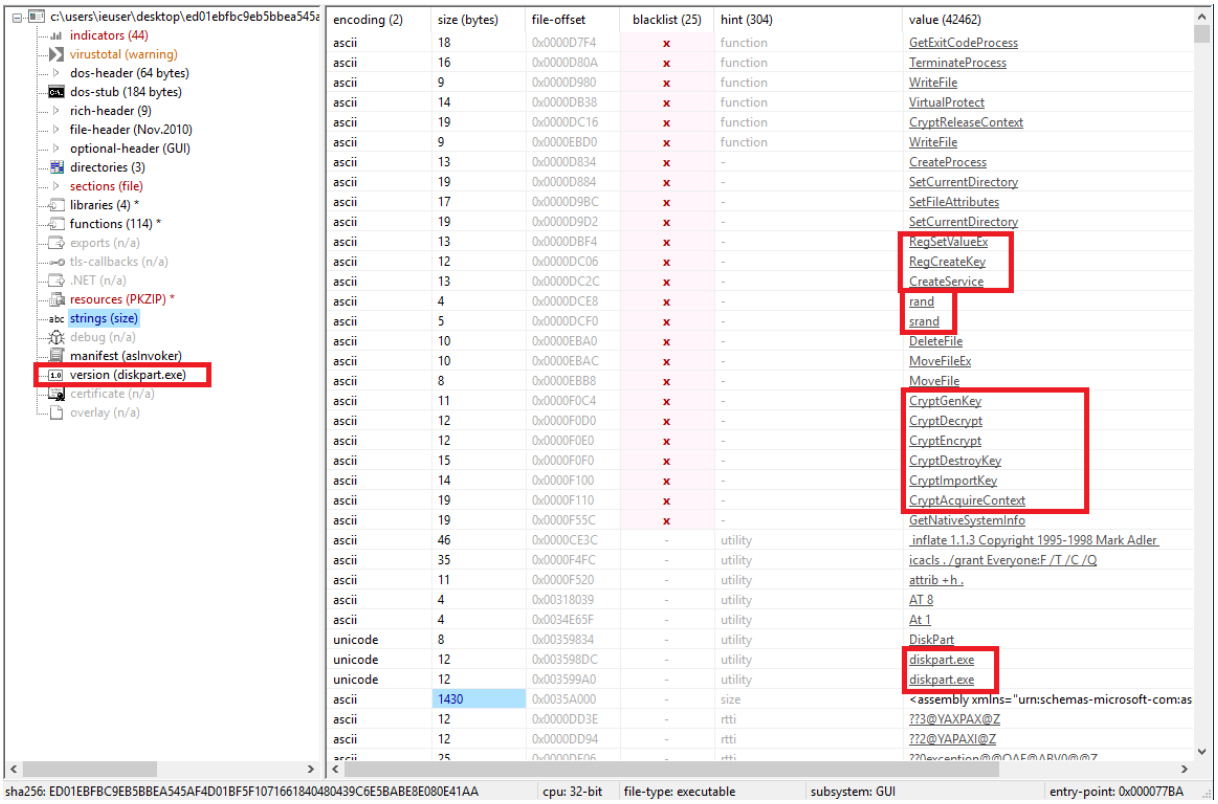
Tuloksista nähtiin, että näytettä on ladattu aikaisemminkin samassa muodossaan tietokantaan. Jälleen jos ei tiedettäisi näytteen haitallisuudesta etukäteen, tämä olisi siitä jo selkeä merkki.

ExeinfoPE-ohjelma ilmoitti tiedoston .resource-osion kattavan yli 90 % tiedoston kokonaiskoosta, joka on ohjelman mukaan yleistä salausohjelmille. Tiedostoa ei näyttänyt olevan pakattu. Tämä on esitetty kuviossa 26.



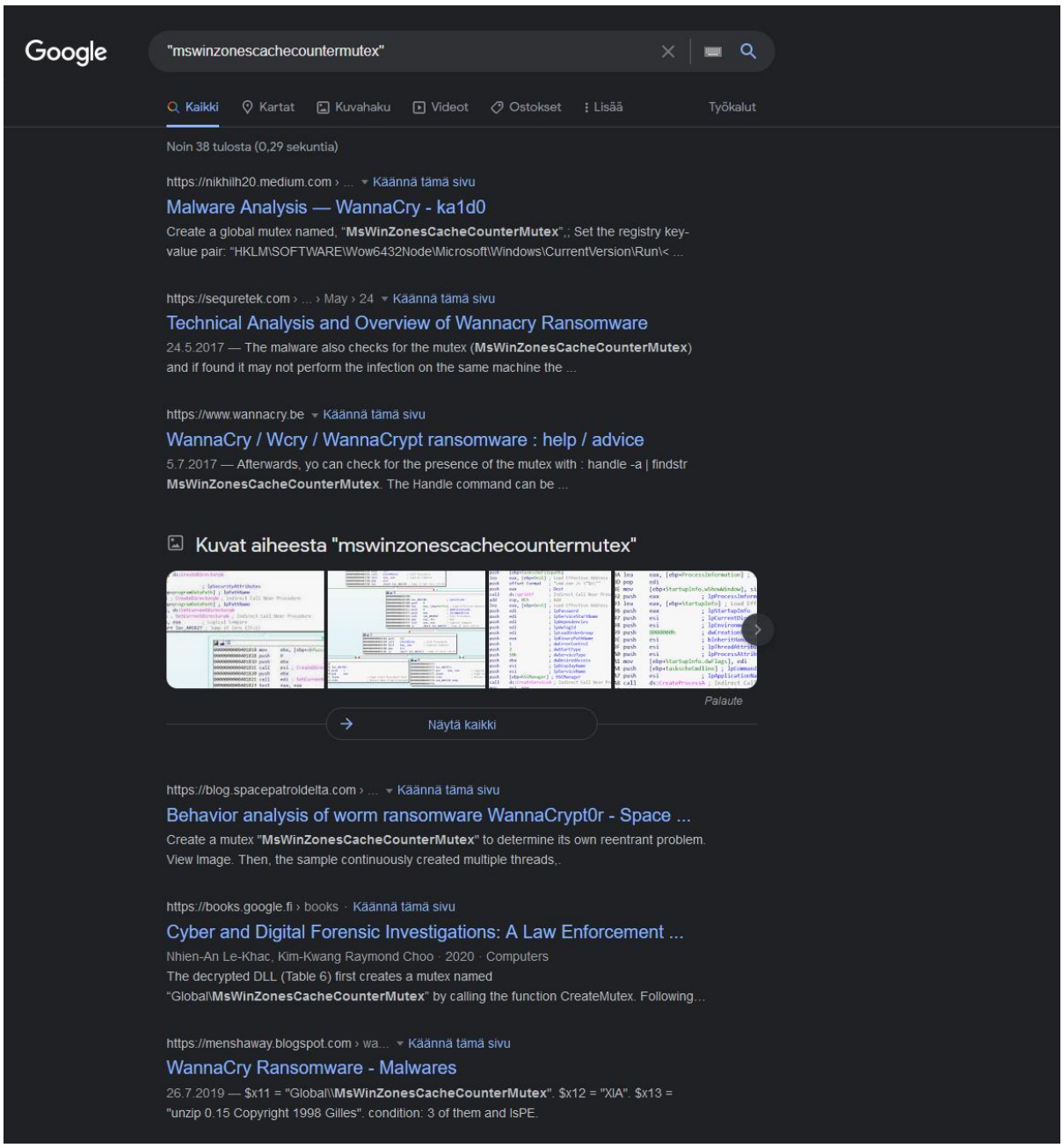
KUVIO 26. WannaCry-haittaohjelmanäyte avattuna ExeinfoPE-ohjelmalla.

Haittaohjelmanäytteen merkkijonoja tarkasteltiin PEStudiolla. Merkkijonoissa huomattiin epäilyttäviä viittauksia *wincrypt.h*:n sisältämiin salaukseen liittyviin funktioihin CryptGenKey, CryptDecrypt, CryptEncrypt, CryptDestroyKey, CryptImportKey, CryptAcguireContext ja CryptReleaseContext. Merkkijonoista nähtiin myös *rand*- ja *srand*-funktiot, joilla luodaan satunnaislukuja, tässä tapauksessa varmaankin salausta varten. Ohjelmanäytteen merkkijonot sisältävät myös rekisteriin ja palveluihin viittaavia RegSetValueEx- ja RegCreateKey, sekä CreateService-funktiokutsut. Nämä viittaavat ohjelman pysyvyyteen. PEStudio ilmoittaa myös ohjelman alkuperäisen version olleen nimeltään *diskpart.exe*, joka on Windowsin oma levynhallintaohjelma. PEStudion näkymä on esitetty kuviossa 27.



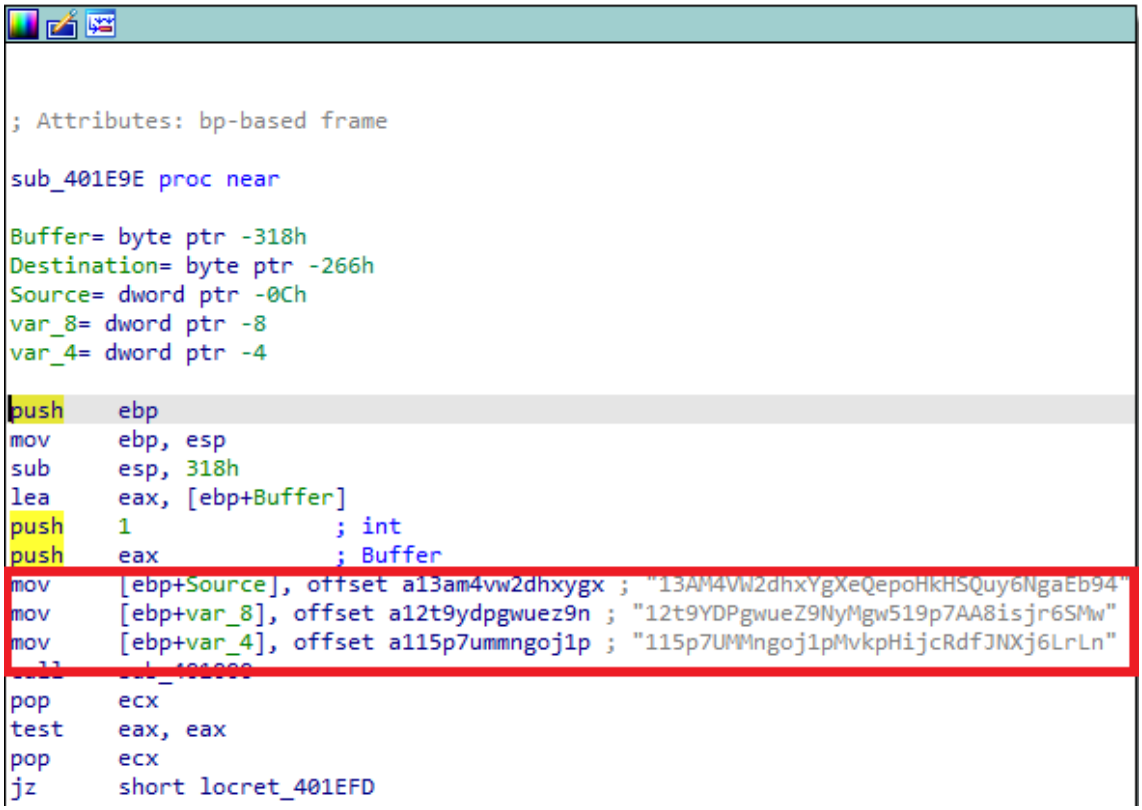
KUVIO 27. WannaCry-näytteen epäilyttäviä merkkijonoja PEStudiolla tarkasteltuna.

Merkkijonoista löydettiin vielä ”WANNACRY!”-merkkijono, viittauksia erilaisiin tiedostopäätteisiin, joita haittaohjelma yrittää salata, sekä mutex-nimike ”MsWinZonesCacheCounterMutex”. Mutex nimi googletettiin, ja tuloksista käy selväksi, että ohjelma on kiristyshaittaohjelma. Jos ko. ohjelmasta ei aikaisemmin olisi kuullut, tuloksista voisi hyvinkin tutustua sen toimintaan. Tulokset on esitetty kuviossa 28.



KUVIO 28. Näytteen sisältämä mutex-nimike syötettynä Googlen hakukoneeseen.

Ohjelman assemblya tarkkailemalla havaittiin myös 3 bitcoin-lompakko-osoitetta, joita ei löydetty merkkijonojen joukosta. Nämä on esitetty kuviossa 29.



```
; Attributes: bp-based frame

sub_401E9E proc near

Buffer= byte ptr -318h
Destination= byte ptr -266h
Source= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 318h
lea     eax, [ebp+Buffer]
push    1               ; int
push    eax             ; Buffer
mov     [ebp+Source], offset a13am4vw2dhxygx ; "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
mov     [ebp+var_8], offset a12t9ydpgwuez9n ; "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
mov     [ebp+var_4], offset a115p7ummngoj1p ; "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
call    sub_401000
pop     ecx
test    eax, eax
pop     ecx
jz      short locret_401EFD
```

KUVIO 29. Ohjelman sisältämät bitcoin-lompakko-osoitteet.

5.4.2 Dynaaminen analyysi

Näytteen dynaaminen analyysi aloitettiin käynnistämällä Linux-virtuaalikone ja varmistamalla, että Wireshark- ja INetsim-ohjelmat olivat käynnissä komennoilla:

$ sudo wireshark

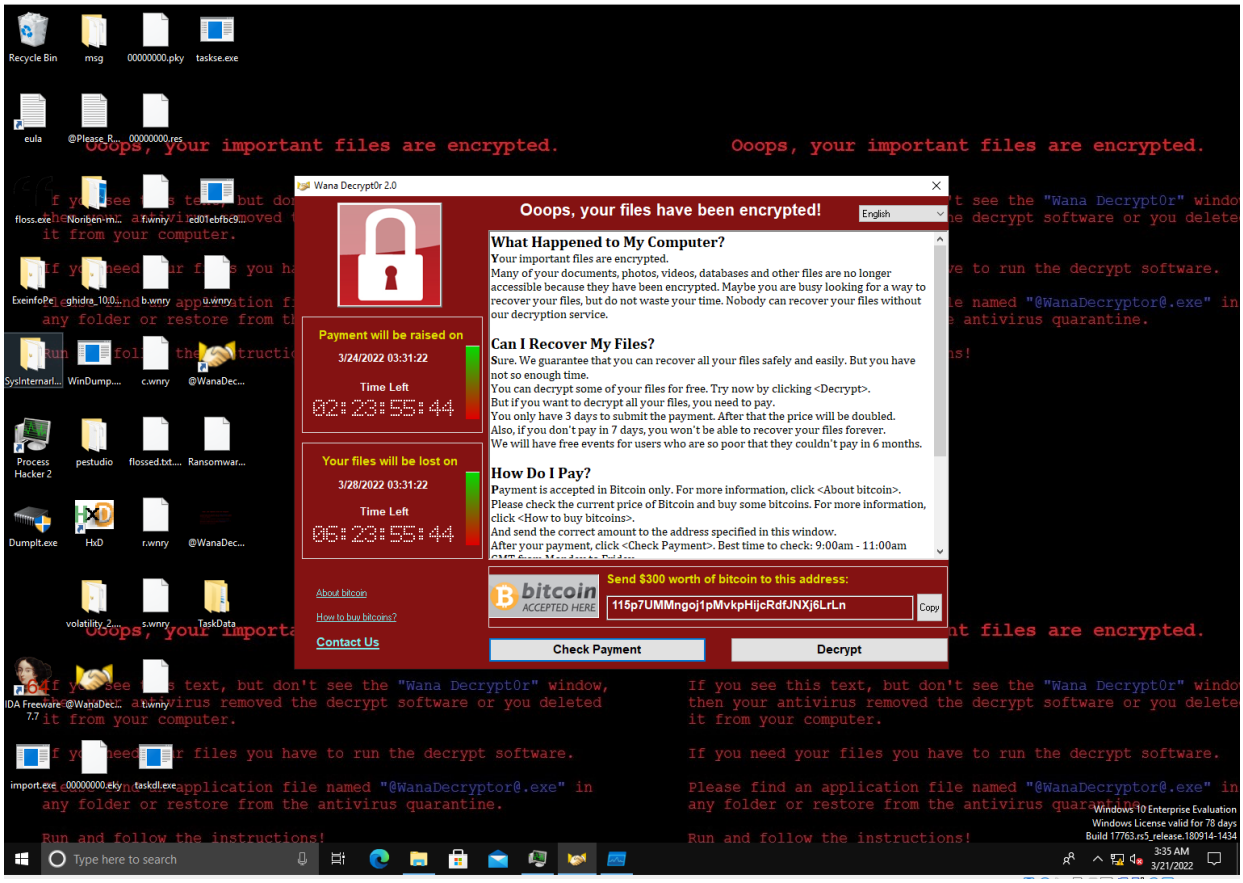
$ sudo inetsim

Wireshark asetettiin tallentamaan käytössä olevan verkkoadapterin verkkoliikennettä.

Windows-virtuaalikoneella käynnistettiin Process Hacker-ohjelma ja suoritettiin Noriben-python-skripti, joka käynnistää SysInteralsSuite:n sisältämän Process Monitor-ohjelman, ja suodattaa sen tuottamia lokitiedostoja helpommin luettavaksi. Python skripti suoritettiin avaamalla komentokehote, navigoimalla kohdekansioon ja suorittamalla komento:

>Python Noriben.py

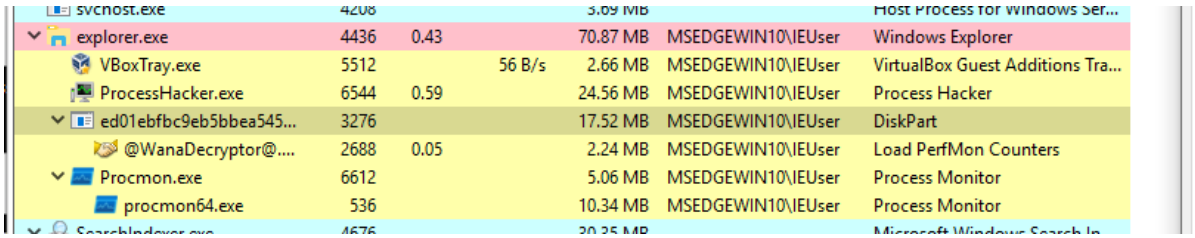
Haittaohjelma suoritettiin, ja hetken kuluttua se oli salannut virtuaalikoneen tiedostot. Kuviossa 30 näkyy koneen työpöytä ohjelman suorittamisen jälkeen.



KUVIO 30. Virtuaalikoneen työpöytä WannaCry-haittaohjelman suorittamisen jälkeen.

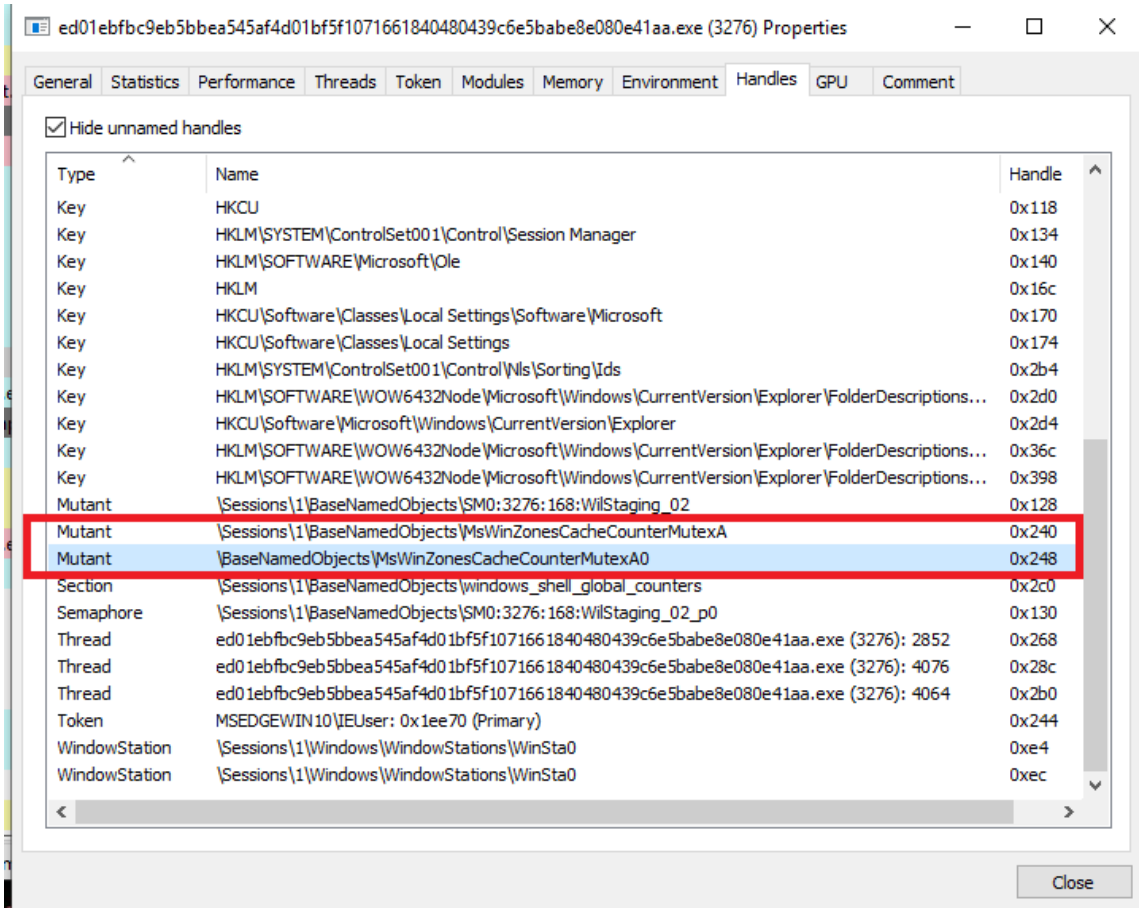
Työpöydälle oli avautunut ohjelma ”Wanna Decrypt0r 2.0”, joka ilmoitti käyttäjälle, että tietokoneen tiedostot olivat nyt salattu, ja niiden palauttaminen vaati maksua ohjelman ilmoittamaan bitcoin-lompakkoon.

ProcessHacker-ohjelmalla nähtiin, että haittaohjelma on luonut uuden prosessin suoritettavasta tiedostosta, jonka emoprosessi on *explorer.exe*. Uusi prosessi on luonut myös oman alaprosessin *@wannaDecryptor@.exe*. Tämä näkyy kuviossa 31.



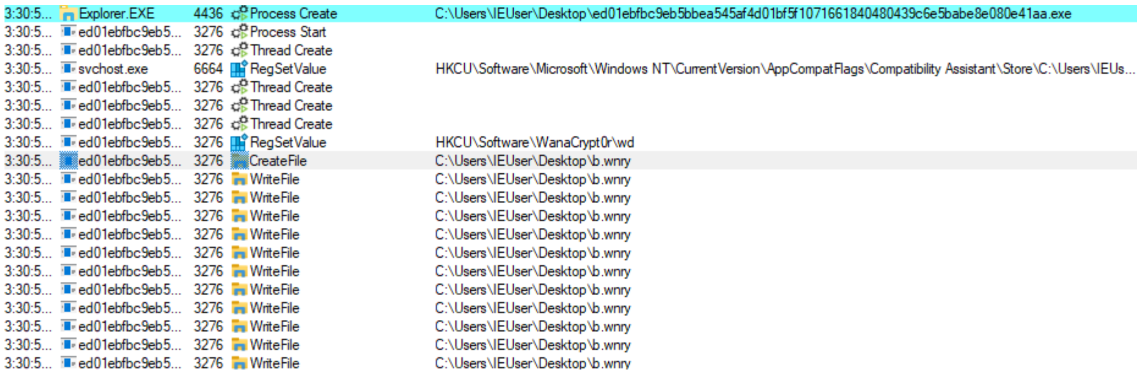
KUVIO 31. Haittaohjelman luomat prosessit.

Uutta prosessia tarkkailemalla huomattiin, että se käyttää aikaisemmin vastaan tullutta mutex-nimikettä. Tämä on esitetty kuviossa 32.



KUVIO 32. Aikaisemmin havaittu mutex-nimike prosessin käytössä.

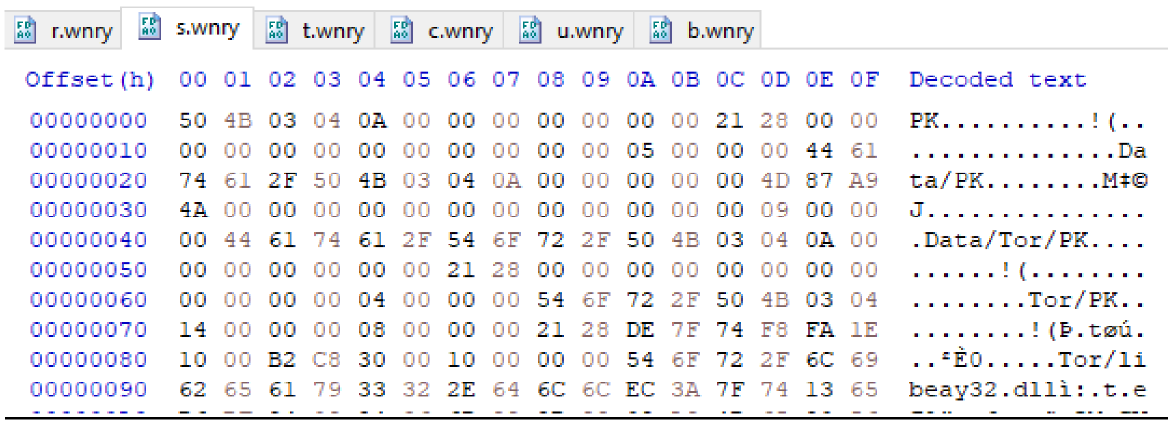
Noribenin luomaa ”.pml”-päätteistä tiedostoa haittaohjelma ei ollut salannut, ja sitä pystyttiin vielä analysoimaan. Kuviossa 33 näkyy, kun ohjelma luo uuden prosessin ja alkaa luomaan tarvitsemiaan tiedostoja.



KUVIO 33. ProcessMonitor-ohjelman näkymä.

Explorer.exe prosessi kutsuu ProcessCreate-funktiota, jonka luoma prosessi alkaa kirjoittamaan *b.wnry*-tiedostoa. Lokia lukemalla nähtiin, että ohjelma luo myös *r.wnry*-, *s.wnry*-, *t.wnry*-, ja *u.wnry*-nimiset tiedostot, sekä lukuisia lokalisaatioon liittyviä .wnry-tiedostoja, kuten esim. *english.wnry*, *german.wnry*, *russian.wnry.* Haittaohjelma on luonut myös *00000000.res*-, *00000000.pky*-, *00000000.dky*- ja *00000000.eky* -nimiset tiedostot työpöydälle.

Tiedostoja analysoitiin HxD-ohjelmalla, ja huomattiin, että *r.wnry* sisältää tekstiä, jota *WannaDecryptor.exe* esittää. Tiedosto *s.wnry* tunnistettiin taikatavujen ”PK” avulla ”.zip”-tiedostoksi, joka sisältää Tor:iin liittyviä tiedostoja. Tämä on esitetty kuviossa 34.



KUVIO 34. *S.wnry*-tiedosto avattuna HxD-ohjelmalla.

t.wnry sisälsi vain yhden luettavan tekstinpätkän ”WANNACRY!”, loppu tiedostosta oli salattua.

u.wnry tunnistettiin suoritettavaksi ohjelmaksi, jälleen taikatavujen ”MZ” avulla. Tämä oli WannaDecryptor:n binääritiedosto, tiedostosta luettavan tekstin perusteella. Tiedosto sisälsi siis merkkijonoja, joita esiintyy WannaDecryptor-ohjelman käyttöliittymässä.

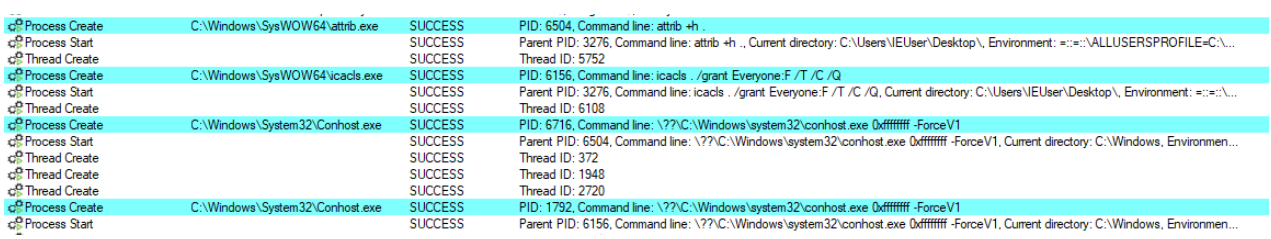
c.wnry-tiedosto sisälsi 5 Tor:n piilopalveluiden .onion-osoitetta, jotka on esitetty taulukossa 1.

TAULUKKO 1. *C.wnry*-tiedoston sisältämät .onion-osoitteet.

	"c.wnry":n sisältämät .onion-osoitteet
1	*gx7ekbenv2riucmf.onion*
2	*57g7spgrzlojinas.onion*
3	*xxlvbrloxvriy2c5.onion*
4	*76jdd2ir2embyv47.onion*
5	*cwwnhwhlz52maqm7.onion*

Tiedostosta löytyi myös viittaus Tor-selaimen lataussivuun.

Haitallinen prosessi avasi *attrib.exe*-, *icacls.exe*-, ja *Conhost.exe*-ohjelmat. Tämä on esitetty kuviossa 35.

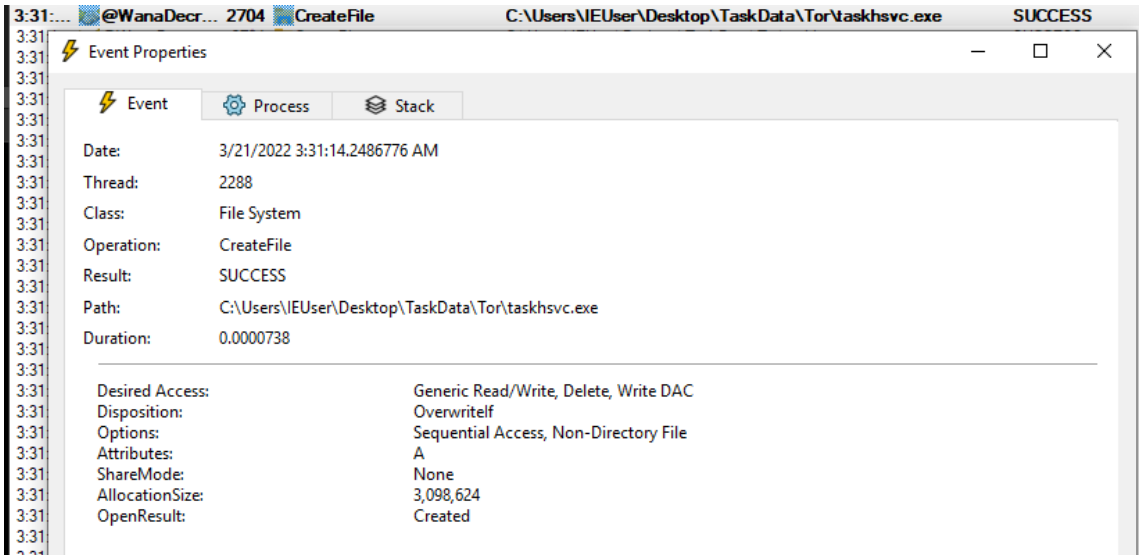


KUVIO 35. Haittaohjelman avaamat prosessit ja niiden suorittamat komennot.

Attrib.exe-ohjelmalla voidaan muuttaa tiedostojen ominaisuuksia ja piilottaa niitä. *Icacls.exe*-ohjelmalla muokataan tiedostojen ja kansioiden käsittelyoikeuksia. *Conhost.exe*-ohjelma liittyy Windowsin komentolinjan oikeaan toimintaan. Tässä tapauksessa haittaohjelma asettaa sen suorituspolun piilotetuksi ”*attrib +h*”-komennolla ja antaa kaikille käyttäjille oikeudet ko. polkuun suorittamalla *Icacls.exe*-ohjelman komennolla *”./grant Everyone:F /T /C /Q”.*

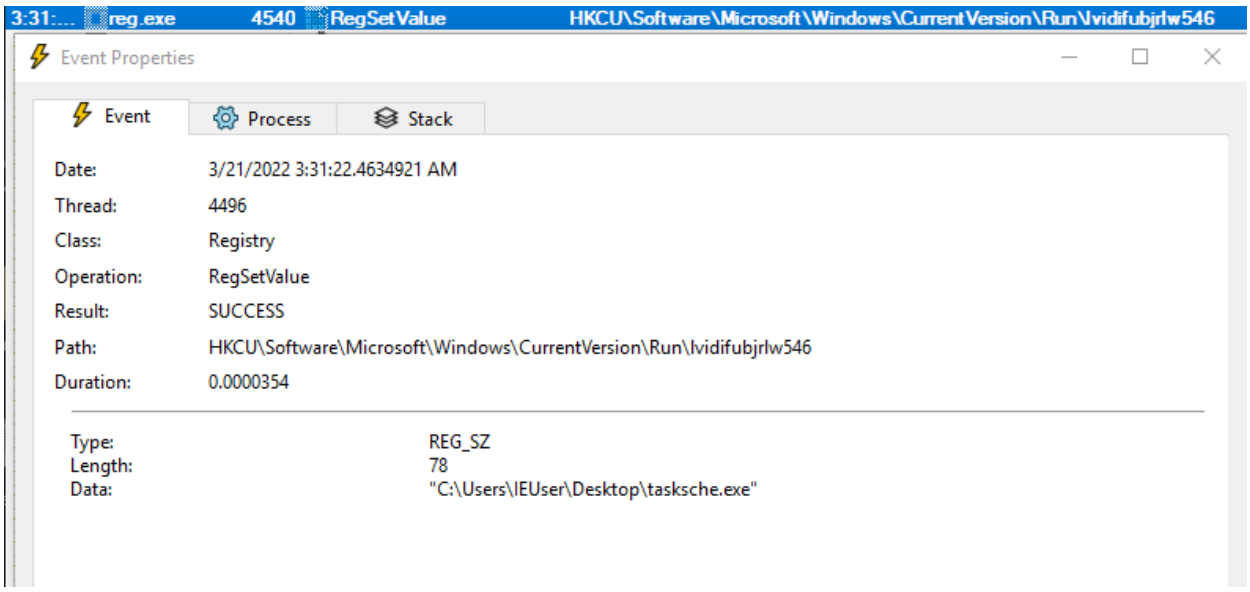
Lokeja seuraamalla nähtiin, että ohjelma alkaa luomaan salattavista tiedostoista *.tmp*-päätteisiä kopioita, jotka muutetaan lopulta salatuiksi *.wncry*- ,tai *.wncryt*-tiedostoiksi.

Lokeista nähtiin, että ohjelma luo suorituspolkuun uuden kansion nimeltään ”TaskData”, johon se purkaa Tor:n suorittamiseen liittyviä tiedostoja, sekä *taskhsvc.exe*-tiedoston. Lokinäkymä on esitetty kuviossa 36.



KUVIO 36. Haittaohjelma luonut uuden kansion ja ”*taskhsvc.exe”*-tiedoston.

Haittaohjelma oli luonut myös rekisteriarvon run-avaimen alle, joka on esitetty kuviossa 37. Rekisteriarvon luominen tarkoittaa, että ohjelma suoritetaan tietokoneen käynnistyessä.



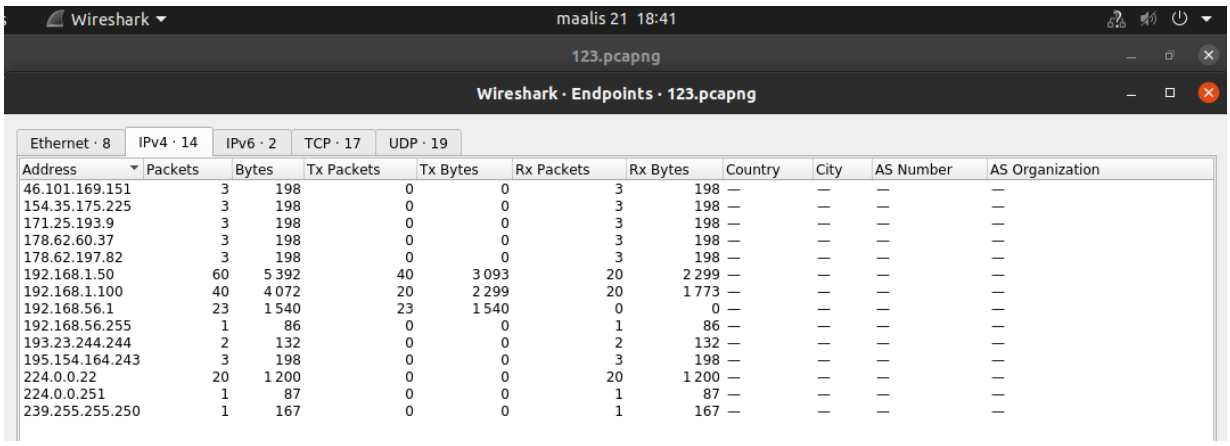
KUVIO 37. ”*Taskhsvc.exe”*-ohjelmalle luotu rekisteriarvo.

Wireshark:n kaappaamaa verkkoliikennettä tarkkailemalla huomattiin, että ohjelma yrittää ottaa yhteyttä IP-osoitteisiin, jotka on esitetty taulukossa 2.

TAULUKKO 2. *Taskhsvc.exe*-prosessin sisältämät IP-osoitteet.

	"Taskhsvc.exe"-prosessin sisältämät IP-osoitteet.
1	*154.35.175.225*
2	*171.25.193.9*
3	*178.62.197.82*
4	*178.62.60.37*
5	*193.23.244.244*
6	*195.154.164.243*
7	*46.101.169.151*

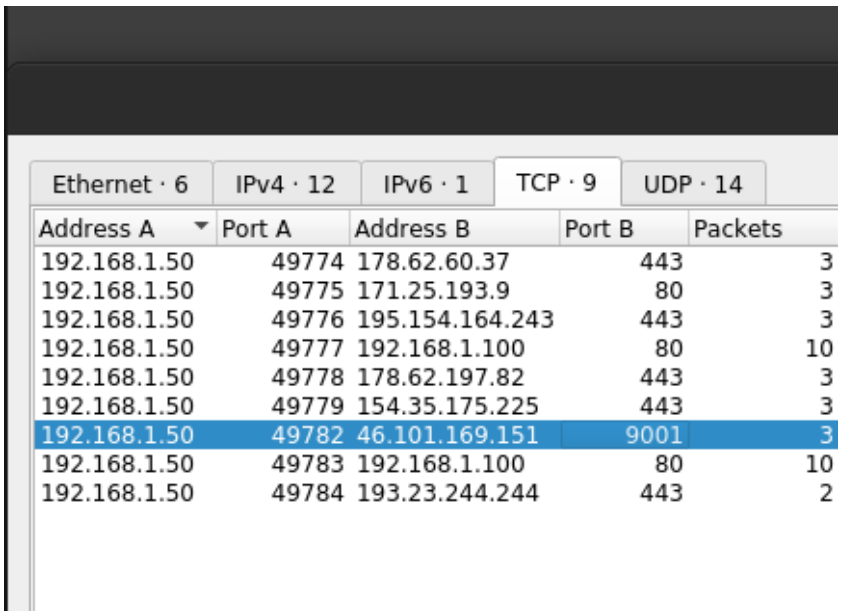
Verkkoliikenteestä kaapatut IP-osoitteet ovat näkyvissä kuviossa 38.



Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
46.101.169.151	3	198	0	0	3	198	—	—	—	—
154.35.175.225	3	198	0	0	3	198	—	—	—	—
171.25.193.9	3	198	0	0	3	198	—	—	—	—
178.62.60.37	3	198	0	0	3	198	—	—	—	—
178.62.197.82	3	198	0	0	3	198	—	—	—	—
192.168.1.50	60	5 392	40	3 093	20	2 299	—	—	—	—
192.168.1.100	40	4 072	20	2 299	20	1 773	—	—	—	—
192.168.56.1	23	1 540	23	1 540	0	0	—	—	—	—
192.168.56.255	1	86	0	0	1	86	—	—	—	—
193.23.244.244	2	132	0	0	2	132	—	—	—	—
195.154.164.243	3	198	0	0	3	198	—	—	—	—
224.0.0.22	20	1 200	0	0	20	1 200	—	—	—	—
224.0.0.251	1	87	0	0	1	87	—	—	—	—
239.255.255.250	1	167	0	0	1	167	—	—	—	—

KUVIO 38. Kaapatussa verkkoliikenteessä esiintyvät IP-osoitteet.

TCP-keskusteluja tarkastelemalla huomattiin, että liikennettä näiden IP-osoitteiden välillä käytiin porttien 80, 443 ja 9001 välillä. Portteja 80 ja 443 käytetään ”tavallisessa” HTTP- ja HTTPS-liikenteessä, mutta kaikkia näitä portteja - ja erityisesti porttia 9001 - käytetään myös Tor-liikenteessä. TCP-keskustelujen portit on esitetty kuviossa 39.



Address A	Port A	Address B	Port B	Packets
192.168.1.50	49774	178.62.60.37	443	3
192.168.1.50	49775	171.25.193.9	80	3
192.168.1.50	49776	195.154.164.243	443	3
192.168.1.50	49777	192.168.1.100	80	10
192.168.1.50	49778	178.62.197.82	443	3
192.168.1.50	49779	154.35.175.225	443	3
192.168.1.50	49782	46.101.169.151	9001	3
192.168.1.50	49783	192.168.1.100	80	10
192.168.1.50	49784	193.23.244.244	443	2

KUVIO 39. TCP-liikenteen portit.

Navigoimalla johonkin erääseen näistä osoitteista selaimella, löydettiin verkkosivu, joka ilmoitti osoitteensa olevan osa Tor-verkkoa. Selainnäkymä on esitetty kuviossa 40.

KUVIO 40. Selainnäkymä eräälle verkkoliikenteessä esiintyneelle IP-osoitteelle.

Haittaohjelma selkeästi kommunikoi Tor-verkon avulla, todennäköisesti komentokeskuksensa kanssa, sekä tarkistaakseen onko käyttäjä maksanut lunnaita. WannaDecryptor-ohjelmassa on ”Contact Us”-painike, jolla käyttäjä voi ottaa yhteyttä hyökkääjiin. Ohjelma ohjaa varmasti tämänkin liikenteen Tor-verkkoon. Voidaan olettaa, että jos näyte suoritettaisiin uudelleen, se varmastikin ottaisi yhteyttä eri IP-soitteisiin.

Yhteenvetona WannaCry-haittaohjelman analyysista saatiin sen verran tietoa, että se voitaisiin tunnistaa tulevaisuudessa. Näitä tiedonpalasia kutustaan indikaattoreiksi (engl. Indicator of Compromise, IOC). IOC:iden avulla voidaan rakentaa haittaohjelmasta profiili, jolla haavoittunut järjestelmä voidaan tunnistaa, esim. YARA-sääntö. WannaCry-näytteitä on tosin analysoitu todella paljon ja syvällisemmin tämän työn ulkopuolella, joten niistä ei lähdetty rakentamaan moisia profiileja. Analyysin aikana kerättyihin indikaattoreihin kuului mm. tiedoston nimi ja eri tiivistesummat, näytteen sisältämät merkkijonot (erityisesti mutex-nimike), sen luomat uudet tiedostot ja rekisteriavaimet ja sen suorittamat komennot, sekä verkkoliikenteessä esiintyvät IP- ja .onion-osoitteet.

6 MUISTIANALYYSI

6.1 Volatility

Työssä suoritettiin muistianalyysia Volatility-nimisellä ohjelmalla, jonka on kehitetty Volatility Foundation:in toimesta vuonna 2007. Ohjelma mahdollistaa tietokoneen suoritushetken analysoinnin tarkastelemalla satunnaismuistin (RAM) sisältämää dataa. Kirjoittamisen hetkellä ohjelman uusin saatavilla oleva versio on 3.0, joka on uudelleenkirjoitettu käyttämään Python-ohjelmointikielen kolmatta julkaisuversiota, joka on epäyhteensopiva aikaisempien python versioiden kanssa. Työssä käytettiin kuitenkin Volatilityn vanhempaa 2.6 versiota, ja siitä saatavilla olevaa itsenäisesti suoritettavaa tyyppiä. Uudessa versiossa on paranneltu ohjelman toimivuutta ja toimintanopeutta suurestikin, mutta kaikkia vanhemman version toimintoja ei ole vielä käännetty uuteen versioon.

Työprosessin helpottamiseksi analysointi suoritettiin Linux-koneella, Volatility 2.6 standalone-versiolla. Ohjelma ladattiin koneelle seuraavasta osoitteesta:

http://downloads.volatilityfoundation.org/releases/2.6/volatility_2.6_lin64_standalone.zip

6.2 Analyysiesimerkki 1, Emotet

Analyysin kohteeksi ladattiin ilmeisesti Emotet-haittaohjelmalla ”tartutettu” Windows käyttöjärjestelmän muistikaappaus. Muistikaappaus olisi voitu luoda myös itse, tartuttamalla Windows virtuaalikone ja suorittamalla *DumpIt.exe*-ohjelma. Kaappauksen siirtäminen virtuaalikoneiden välillä todettiin kuitenkin turhan työlääksi, joten analyysin helpottamiseksi, valmiiksi tartutetun järjestelmän muistikaappaus ladattiin Internetistä suoraan Linux-koneelle. Muistikaappaus ladattiin tryhackme.com-sivuston kautta, seuraavasta osoitteesta:

https://tryhackme.com/room/forensics

Sivuston käyttäminen vaatii käyttäjätunnuksen luomista, ja usein myös kuukausimaksua. Tämä huone ja sen sisältämät resurssit ovat kuitenkin saatavilla ilmaiseksi.

Analyysi aloitettiin selvittämällä muistikaappauksen sisältämiä tietoja. Tämä tehtiin komennolla:

```
$./volatility_2.6_lin64_standalone -f victim.raw imageinfo
```

missä "*./volatility_2.6_lin64_standalone*" suorittaa Volatility-ohjelman, *"-f"* määrittää ohjelmalle kohdetiedoston *victim.raw*, ja *imageinfo* on ohjelman sisäinen liitännäinen (engl. "plugin"), joka esittää tietoa muistikaappaustiedostosta.

Komennon tulokset on esitetty kuviossa 41.

KUVIO 41. Muistikaappaustiedoston sisältämät tiedot esitetty Volatilityn imageinfo-liitännäisellä.

Ohjelman tulostuksesta nähtiin, että se suosittelee käytettäväksi profiiliksi *Win7SP1x64*-profiilia. Muistikaappaus on siis otettu todennäköisesti 64-bittistä Windows 7-käyttöjärjestelmää käyttävästä tietokoneesta. Oikeaa profiilia tarvitaan ohjelman toiminnan takaamiseksi.

Seuraavaksi muistikaappauksesta selvitettiin kaappauksen aikana käynnissä olleet prosessit komennolla:

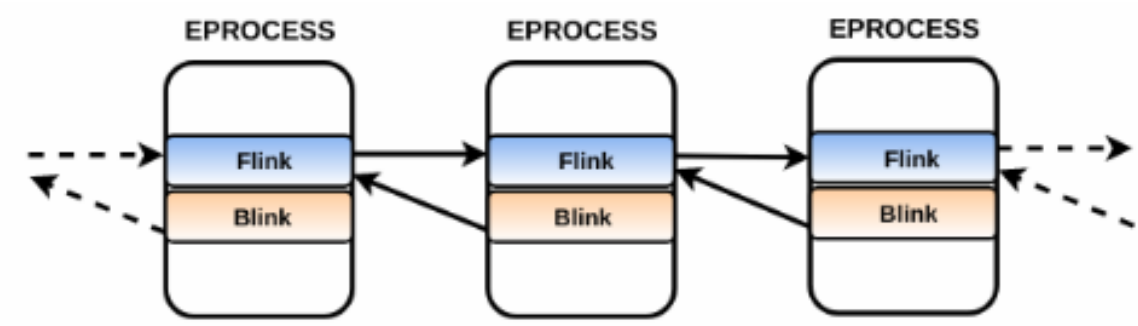
```
$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 pslist
```

Komennon tulokset on esitetty kuviossa 42.

```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------
0xfffffa8001252040 System                    4      0     88      624 ------      0 2019-05-03 06:32:24 UTC+0000
0xfffffa800234d8a0 smss.exe                268      4      2       29 ------      0 2019-05-03 06:32:24 UTC+0000
0xfffffa8002264550 csrss.exe               360    352      9      363      0      0 2019-05-03 06:32:34 UTC+0000
0xfffffa80027d67d0 csrss.exe               408    400      7      162      1      0 2019-05-03 06:32:35 UTC+0000
0xfffffa8002b601c0 wininit.exe             416    352      3       76      0      0 2019-05-03 06:32:35 UTC+0000
0xfffffa8002b71680 winlogon.exe            444    400      3      111      1      0 2019-05-03 06:32:35 UTC+0000
0xfffffa8002c69b30 services.exe            504    416      6      184      0      0 2019-05-03 06:32:36 UTC+0000
0xfffffa80027d9b30 lsass.exe               512    416      6      534      0      0 2019-05-03 06:32:37 UTC+0000
0xfffffa80027d81f0 lsm.exe                 520    416     10      143      0      0 2019-05-03 06:32:37 UTC+0000
0xfffffa80029cd3e0 svchost.exe             628    504      9      345      0      0 2019-05-03 06:32:48 UTC+0000
0xfffffa8002d38b30 VBoxService.ex          688    504     12      135      0      0 2019-05-03 06:32:48 UTC+0000
0xfffffa8002a1bb30 svchost.exe             752    504      7      235      0      0 2019-05-02 18:02:51 UTC+0000
0xfffffa8002d70650 svchost.exe             852    504     22      473      0      0 2019-05-02 18:02:51 UTC+0000
0xfffffa8002d9c780 svchost.exe             892    504     17      427      0      0 2019-05-02 18:02:51 UTC+0000
0xfffffa8002dbe9e0 svchost.exe             920    504     29      878      0      0 2019-05-02 18:02:51 UTC+0000
0xfffffa8002e3db30 svchost.exe             400    504     10      281      0      0 2019-05-02 18:02:56 UTC+0000
0xfffffa8002e57890 svchost.exe            1004    504     20      379      0      0 2019-05-02 18:02:56 UTC+0000
0xfffffa8002dfdab0 spoolsv.exe            1140    504     12      279      0      0 2019-05-02 18:02:57 UTC+0000
0xfffffa8002f2cb30 svchost.exe            1268    504     17      297      0      0 2019-05-02 18:02:59 UTC+0000
0xfffffa8002f81460 svchost.exe            1368    504     20      295      0      0 2019-05-02 18:02:59 UTC+0000
0xfffffa8003148b30 taskhost.exe           1788    504      8      159      1      0 2019-05-02 18:03:09 UTC+0000
0xfffffa8003172b30 explorer.exe           1860   1756     19      645      1      0 2019-05-02 18:03:09 UTC+0000
0xfffffa800315eb30 dwm.exe                1896    892      3       69      1      0 2019-05-02 18:03:09 UTC+0000
0xfffffa800300d700 VBoxTray.exe           1600   1860     13      141      1      0 2019-05-02 18:03:25 UTC+0000
0xfffffa8003367060 SearchIndexer.         2180    504     11      629      0      0 2019-05-02 18:03:32 UTC+0000
0xfffffa80033f6060 WmiPrvSE.exe           2876    628      5      113      0      0 2019-05-02 18:03:55 UTC+0000
0xfffffa8003162060 svchost.exe            1820    504     11      317      0      0 2019-05-02 18:05:09 UTC+0000
0xfffffa8003371540 wmpnetwk.exe           2464    504     14      440      0      0 2019-05-02 18:05:10 UTC+0000
0xfffffa80014eeb30 taskhost.exe           1148    504      8      176      0      0 2019-05-02 18:09:58 UTC+0000
malwarelab@malwarelabvm:~/Desktop/Volatility$
```

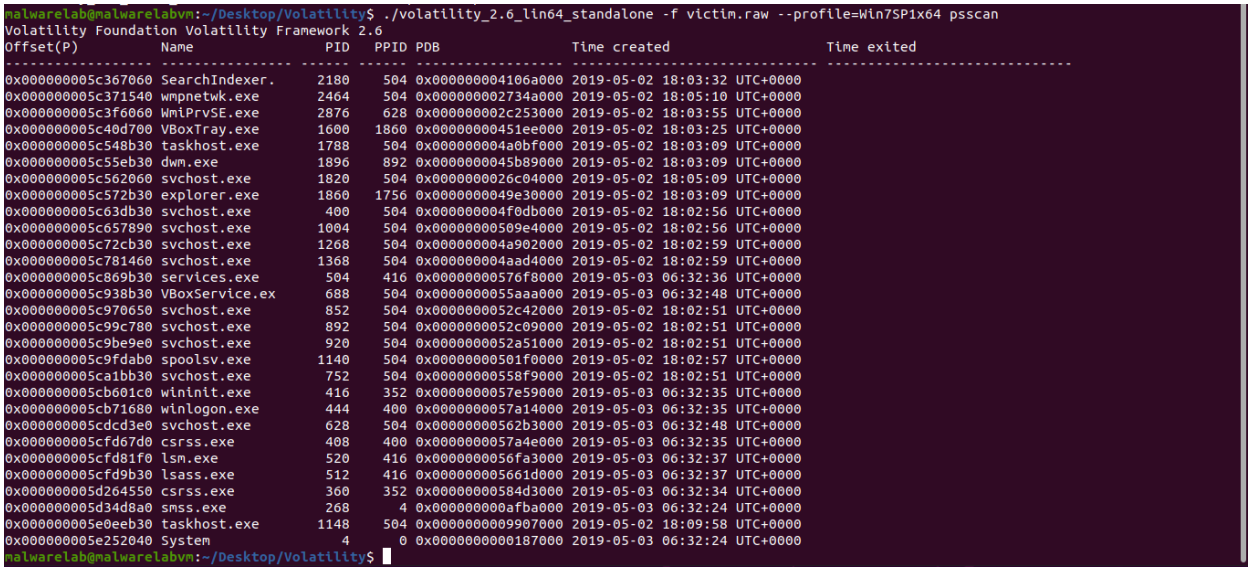
KUVIO 42. ”pslist”-liitännäisen tulokset, eli järjestelmässä kaappaushetkellä käynnissä olleet prosessit.

Volatilityn 2.6-versiossa prosesseja listaava ”pslist”-liitännäinen etsii prosesseja lukemalla kernel-tilasta löytyviä ”*_EPROCESS*”-tietueita, jota myös Windows käyttöjärjestelmä käyttää prosessien esittämisessä. Nämä tietueet on yhdistetty toisiinsa linkitettyyn listaan, joten jokaisesta tietueesta selviää aikaisemman ja seuraavan tietueen osoite. (Monnappa 2018, 10.) Tämä on esitetty kuviossa 43.



KUVIO 43. *EPROCESS*-tietueiden linkitetty lista (Monnappa 2018, 10).

Ohjelman ”psscan”-liitännäisen avulla voidaan tunnistaa prosesseja, jotka on piilotettu, tai jotka suljettu. Psscan-liitännäinen tekee tämän etsimällä muistialueesta ”*_POOL_HEADER*”-tietueiden sisältämiä ”PoolTag”-merkintöjä, jotka käyttöjärjestelmä luo kaikille objekteille, kun ne kirjotetaan muistiin. (Monnappa 2018, 10.) ”Psscan”-liitännäisen tulokset on esitetty kuviossa 44.



```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 psscan
Volatility Foundation Volatility Framework 2.6
Offset(P)          Name                PID   PPID PDB               Time created                   Time exited
------------------ ---------------- ------ ------ ----------------- ------------------------------ ------------------------------
0x000000005c367060 SearchIndexer.     2180    504 0x000000004106a000 2019-05-02 18:03:32 UTC+0000
0x000000005c371540 wmpnetwk.exe       2464    504 0x000000002734a000 2019-05-02 18:05:10 UTC+0000
0x000000005c3f6060 WmiPrvSE.exe       2876    628 0x000000002c253000 2019-05-02 18:03:55 UTC+0000
0x000000005c40d700 VBoxTray.exe       1600   1860 0x000000000451ee000 2019-05-02 18:03:25 UTC+0000
0x000000005c548b30 taskhost.exe       1788    504 0x000000004a0bf000 2019-05-02 18:03:09 UTC+0000
0x000000005c55eb30 dwm.exe            1896    892 0x0000000045b89000 2019-05-02 18:03:09 UTC+0000
0x000000005c562060 svchost.exe        1820    504 0x0000000026c04000 2019-05-02 18:05:09 UTC+0000
0x000000005c572b30 explorer.exe       1860   1756 0x0000000049e30000 2019-05-02 18:03:09 UTC+0000
0x000000005c63db30 svchost.exe         400    504 0x000000004f0db000 2019-05-02 18:02:56 UTC+0000
0x000000005c657890 svchost.exe        1004    504 0x00000000509e4000 2019-05-02 18:02:56 UTC+0000
0x000000005c72cb30 svchost.exe        1268    504 0x000000004a902000 2019-05-02 18:02:59 UTC+0000
0x000000005c781460 svchost.exe        1368    504 0x000000004aad4000 2019-05-02 18:02:59 UTC+0000
0x000000005c869b30 services.exe        504    416 0x00000000576f8000 2019-05-03 06:32:36 UTC+0000
0x000000005c938b30 VBoxService.ex      688    504 0x0000000055aaa000 2019-05-03 06:32:48 UTC+0000
0x000000005c970650 svchost.exe         852    504 0x0000000052c42000 2019-05-02 18:02:51 UTC+0000
0x000000005c99c780 svchost.exe         892    504 0x0000000052c09000 2019-05-02 18:02:51 UTC+0000
0x000000005c9be9e0 svchost.exe         920    504 0x0000000052a51000 2019-05-02 18:02:51 UTC+0000
0x000000005c9fdab0 spoolsv.exe        1140    504 0x00000000501f0000 2019-05-02 18:02:57 UTC+0000
0x000000005ca1bb30 svchost.exe         752    504 0x00000000558f9000 2019-05-02 18:02:51 UTC+0000
0x000000005cb601c0 wininit.exe         416    352 0x0000000057e59000 2019-05-03 06:32:35 UTC+0000
0x000000005cb71680 winlogon.exe        444    400 0x0000000057a14000 2019-05-03 06:32:35 UTC+0000
0x000000005cdcd3e0 svchost.exe         628    504 0x00000000562b3000 2019-05-03 06:32:48 UTC+0000
0x000000005cfd67d0 csrss.exe           408    400 0x0000000057a4e000 2019-05-03 06:32:35 UTC+0000
0x000000005cfd81f0 lsm.exe             520    416 0x0000000056fa3000 2019-05-03 06:32:37 UTC+0000
0x000000005cfd9b30 lsass.exe           512    416 0x000000005661d000 2019-05-03 06:32:37 UTC+0000
0x000000005d264550 csrss.exe           360    352 0x00000000584d3000 2019-05-03 06:32:34 UTC+0000
0x000000005d34d8a0 smss.exe            268      4 0x000000000afba000 2019-05-03 06:32:24 UTC+0000
0x000000005e0eeb30 taskhost.exe       1148    504 0x0000000009907000 2019-05-02 18:09:58 UTC+0000
0x000000005e252040 System                4      0 0x0000000000187000 2019-05-03 06:32:24 UTC+0000
malwarelab@malwarelabvm:~/Desktop/Volatility$
```

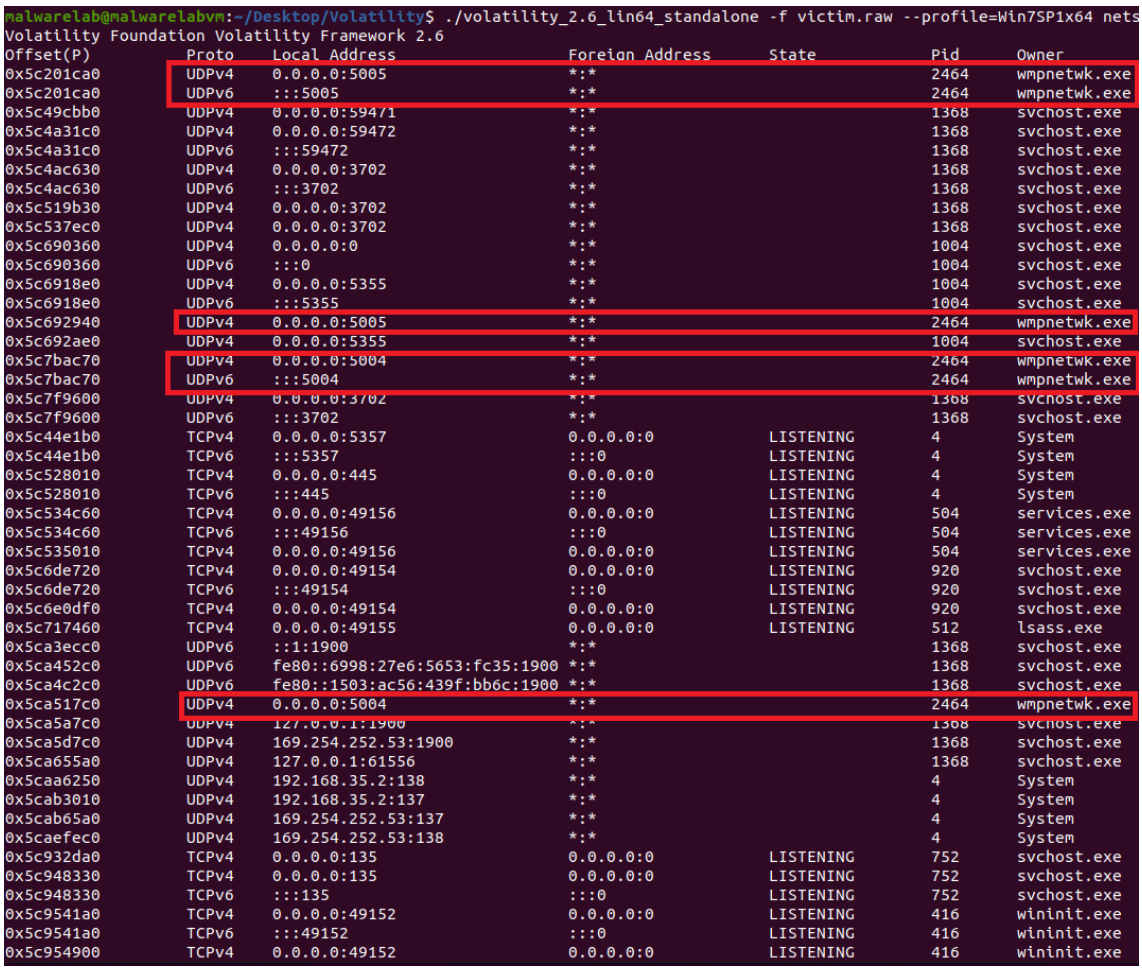
KUVIO 44. Psscan-liitännäisen esittämät prosessit.

Tuloksista huomattiin, että *explorer.exe*-, *csrss.exe*- ja *wininit.exe*-prosesseilla oli PPID (emoprosessin ID), jota ei näkynyt listassa. Yleensä tämä olisi huomioita herättävää, mutta se on näille prosesseille normaalia. Pelkästään prosesseja tarkastelemalla ei nähty mitään epäilyttävää.

Muistin sisältämää verkkoliikenteeseen liittyvää dataa tarkasteltiin seuraavaksi komennolla:

```
$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 netscan
```

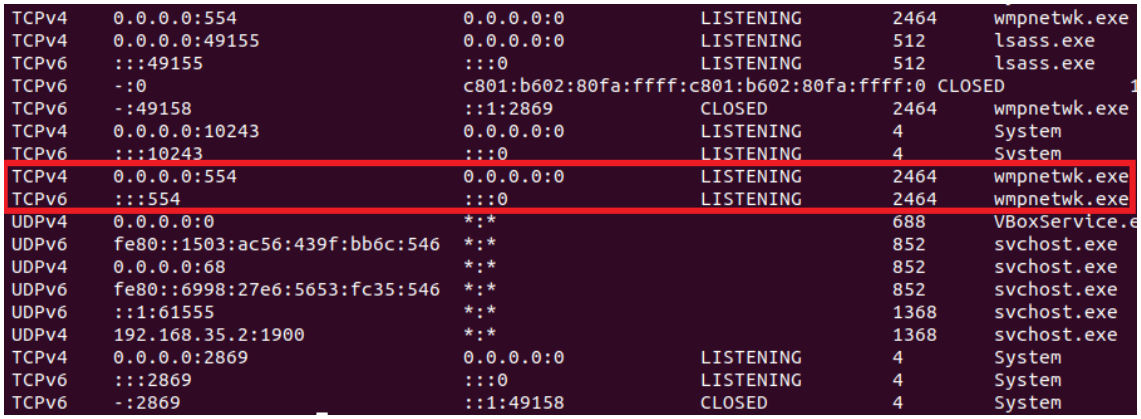
Komennon suorituksen tulokset näkyvät osittain kuviossa 45.



```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 nets
Volatility Foundation Volatility Framework 2.6
Offset(P)          Proto    Local Address                  Foreign Address      State            Pid      Owner
0x5c201ca0         UDPv4    0.0.0.0:5005                   *:*                                   2464     wmpnetwk.exe
0x5c201ca0         UDPv6    :::5005                        *:*                                   2464     wmpnetwk.exe
0x5c49cbb0         UDPv4    0.0.0.0:59471                  *:*                                   1368     svchost.exe
0x5c4a31c0         UDPv4    0.0.0.0:59472                  *:*                                   1368     svchost.exe
0x5c4a31c0         UDPv6    :::59472                       *:*                                   1368     svchost.exe
0x5c4ac630         UDPv4    0.0.0.0:3702                   *:*                                   1368     svchost.exe
0x5c4ac630         UDPv6    :::3702                        *:*                                   1368     svchost.exe
0x5c519b30         UDPv4    0.0.0.0:3702                   *:*                                   1368     svchost.exe
0x5c537ec0         UDPv4    0.0.0.0:3702                   *:*                                   1368     svchost.exe
0x5c690360         UDPv4    0.0.0.0:0                      *:*                                   1004     svchost.exe
0x5c690360         UDPv6    :::0                           *:*                                   1004     svchost.exe
0x5c6918e0         UDPv4    0.0.0.0:5355                   *:*                                   1004     svchost.exe
0x5c6918e0         UDPv6    :::5355                        *:*                                   1004     svchost.exe
0x5c692940         UDPv4    0.0.0.0:5005                   *:*                                   2464     wmpnetwk.exe
0x5c692ae0         UDPv4    0.0.0.0:5355                   *:*                                   1004     svchost.exe
0x5c7bac70         UDPv4    0.0.0.0:5004                   *:*                                   2464     wmpnetwk.exe
0x5c7bac70         UDPv6    :::5004                        *:*                                   2464     wmpnetwk.exe
0x5c7f9600         UDPv4    0.0.0.0:3702                   *:*                                   1368     svchost.exe
0x5c7f9600         UDPv6    :::3702                        *:*                                   1368     svchost.exe
0x5c44e1b0         TCPv4    0.0.0.0:5357                   0.0.0.0:0            LISTENING        4        System
0x5c44e1b0         TCPv6    :::5357                        :::0                 LISTENING        4        System
0x5c528010         TCPv4    0.0.0.0:445                    0.0.0.0:0            LISTENING        4        System
0x5c528010         TCPv6    :::445                         :::0                 LISTENING        4        System
0x5c534c60         TCPv4    0.0.0.0:49156                  0.0.0.0:0            LISTENING        504      services.exe
0x5c534c60         TCPv6    :::49156                       :::0                 LISTENING        504      services.exe
0x5c535010         TCPv4    0.0.0.0:49156                  0.0.0.0:0            LISTENING        504      services.exe
0x5c6de720         TCPv4    0.0.0.0:49154                  0.0.0.0:0            LISTENING        920      svchost.exe
0x5c6de720         TCPv6    :::49154                       :::0                 LISTENING        920      svchost.exe
0x5c6e0df0         TCPv4    0.0.0.0:49154                  0.0.0.0:0            LISTENING        920      svchost.exe
0x5c717460         TCPv4    0.0.0.0:49155                  0.0.0.0:0            LISTENING        512      lsass.exe
0x5ca3ecc0         UDPv6    ::1:1900                       *:*                                   1368     svchost.exe
0x5ca452c0         UDPv6    fe80::6998:27e6:5653:fc35:1900 *:*                                   1368     svchost.exe
0x5ca4c2c0         UDPv6    fe80::1503:ac56:439f:bb6c:1900 *:*                                   1368     svchost.exe
0x5ca517c0         UDPv4    0.0.0.0:5004                   *:*                                   2464     wmpnetwk.exe
0x5ca5a7c0         UDPv4    127.0.0.1:1900                 *:*                                   1368     svchost.exe
0x5ca5d7c0         UDPv4    169.254.252.53:1900            *:*                                   1368     svchost.exe
0x5ca655a0         UDPv4    127.0.0.1:61556                *:*                                   1368     svchost.exe
0x5caa6250         UDPv4    192.168.35.2:138               *:*                                   4        System
0x5cab3010         UDPv4    192.168.35.2:137               *:*                                   4        System
0x5cab65a0         UDPv4    169.254.252.53:137             *:*                                   4        System
0x5caefec0         UDPv4    169.254.252.53:138             *:*                                   4        System
0x5c932da0         TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING        752      svchost.exe
0x5c948330         TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING        752      svchost.exe
0x5c948330         TCPv6    :::135                         :::0                 LISTENING        752      svchost.exe
0x5c9541a0         TCPv4    0.0.0.0:49152                  0.0.0.0:0            LISTENING        416      wininit.exe
0x5c9541a0         TCPv6    :::49152                       :::0                 LISTENING        416      wininit.exe
0x5c954900         TCPv4    0.0.0.0:49152                  0.0.0.0:0            LISTENING        416      wininit.exe
```

KUVIO 45. ”Netscan”-liitännäisen osittaiset tulokset.

Tuloksista nähtiin, että prosessi *wmpnetwk.exe* - jonka PID on 2464 - luo järjestelmässä monia eri yhteyksiä. Kuviossa 46 on esitetty tulos, josta selvisi, että kyseinen prosessi kuuntelee yhteyksiä kaikista osoitteista, portin 554 kautta. Prosessi on käyttänyt myös porttia 2869, mutta yhteys oli suljettu.



```
TCPv4    0.0.0.0:554                    0.0.0.0:0                LISTENING        2464     wmpnetwk.exe
TCPv4    0.0.0.0:49155                  0.0.0.0:0                LISTENING        512      lsass.exe
TCPv6    :::49155                       :::0                     LISTENING        512      lsass.exe
TCPv6    -:0                            c801:b602:80fa:ffff:c801:b602:80fa:ffff:0 CLOSED
TCPv6    -:49158                        ::1:2869                 CLOSED           2464     wmpnetwk.exe
TCPv4    0.0.0.0:10243                  0.0.0.0:0                LISTENING        4        System
TCPv6    :::10243                       :::0                     LISTENING        4        System
TCPv4    0.0.0.0:554                    0.0.0.0:0                LISTENING        2464     wmpnetwk.exe
TCPv6    :::554                         :::0                     LISTENING        2464     wmpnetwk.exe
UDPv4    0.0.0.0:0                      *:*                                       688      VBoxService.e
UDPv6    fe80::1503:ac56:439f:bb6c:546  *:*                                       852      svchost.exe
UDPv4    0.0.0.0:68                     *:*                                       852      svchost.exe
UDPv6    fe80::6998:27e6:5653:fc35:546  *:*                                       852      svchost.exe
UDPv6    ::1:61555                      *:*                                       1368     svchost.exe
UDPv4    192.168.35.2:1900              *:*                                       1368     svchost.exe
TCPv4    0.0.0.0:2869                   0.0.0.0:0                LISTENING        4        System
TCPv6    :::2869                        :::0                     LISTENING        4        System
TCPv6    -:2869                         ::1:49158                CLOSED           4        System
```

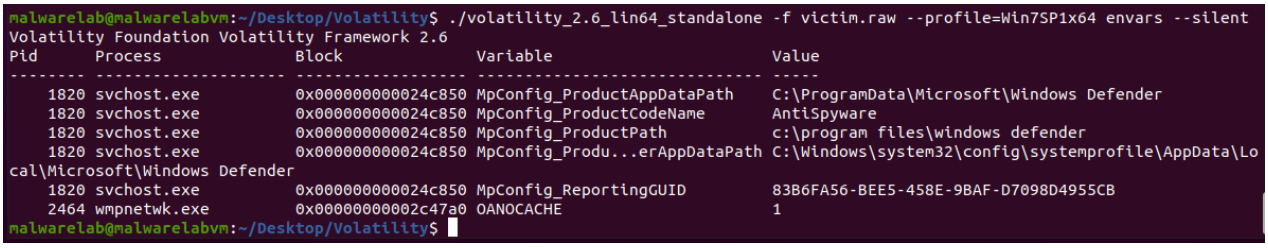
KUVIO 46. Netscan-liitännäisen tuloksia.

Lyhyellä tiedonhaulla selvitettiin, että näiden porttien käyttö tälle ohjelmalle on normaalia.

Volatilityn avulla voidaan näyttää prosessien ympäristömuuttujia (engl. ”environment variable”) ”envars”-liitännäisellä. Syöttämällä sille ”–silent” parametrin, ohjelma esittää vain yleisistä poikkeavat tulokset. Analyysissa suoritettiin seuraavaksi komento:

```
$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 envars --silent
```

Suorituksen tulokset on esitetty kuviossa 47.



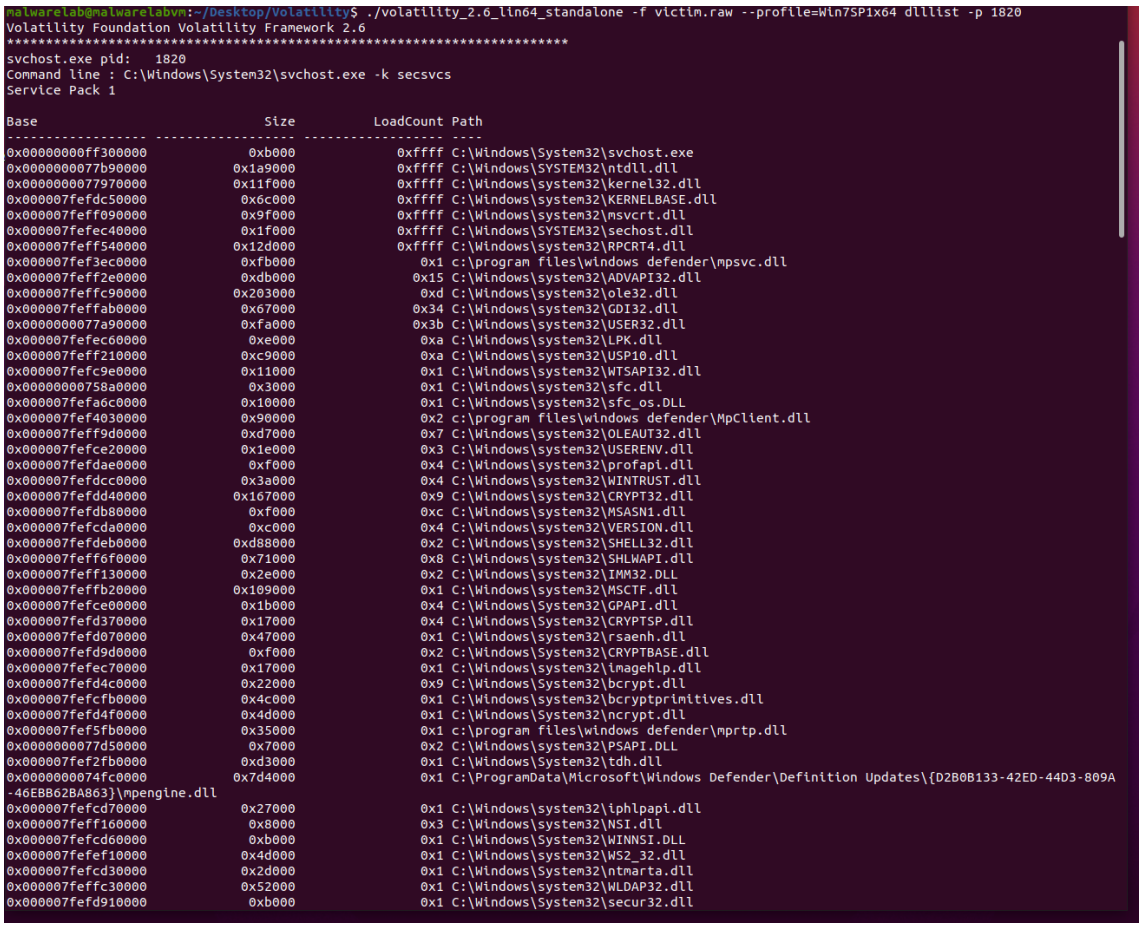
KUVIO 47. Yleisistä poikkeavat ympäristömuuttujat.

Tuloksissa esiintyi jo aikaisemmin tunnistettu 2464 PID:n prosessi, mutta myös toinen 1820 PID:n omaava ”*svchost.exe*”-prosessi. ”*Svchost.exe”*-prosessi on Windowsille ominainen, ja se isännöi useita käyttöjärjestelmän palveluita. Haittaohjelmat voivat kuitenkin kaapata ko. prosessin saamalla sen suorittamaan omia palveluitaan.

Prosessien käyttämät .dll tiedostot listattiin ”dlllist”-liitännäisen avulla, seuraavalla komennolla:

```
$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 dlllist -p 1820,2464
```

Liitännäisen tulokset on esitetty kuvioissa 48 & 49.

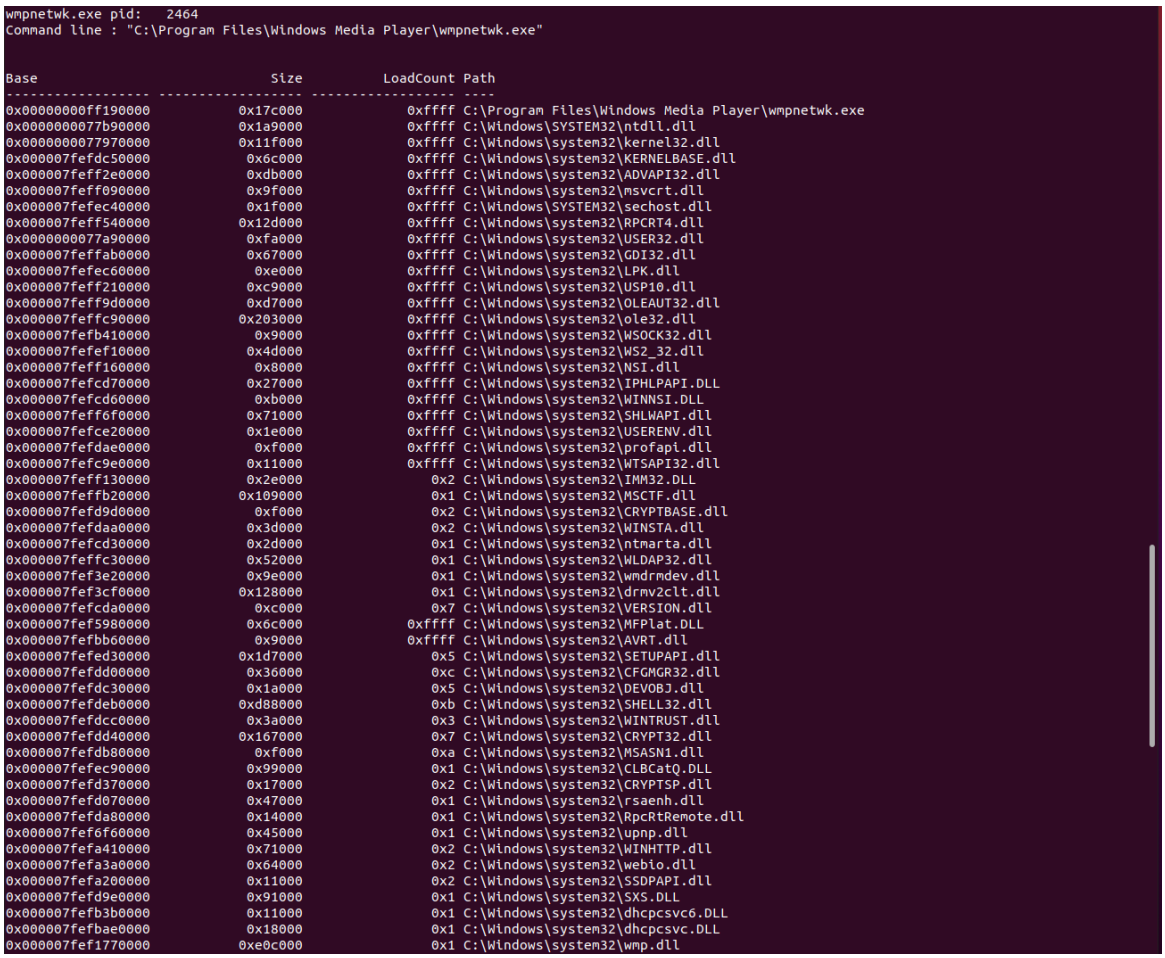


```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 dlllist -p 1820
Volatility Foundation Volatility Framework 2.6
************************************************************************
svchost.exe pid:   1820
Command line : C:\Windows\System32\svchost.exe -k secsvcs
Service Pack 1

Base                             Size          LoadCount Path
------------------ ------------------ ------------------ ----
0x00000000ff300000             0xb000             0xffff C:\Windows\System32\svchost.exe
0x0000000077b90000           0x1a9000             0xffff C:\Windows\SYSTEM32\ntdll.dll
0x0000000077970000           0x11f000             0xffff C:\Windows\system32\kernel32.dll
0x000007fefdc50000            0x6c000             0xffff C:\Windows\system32\KERNELBASE.dll
0x000007feff090000            0x9f000             0xffff C:\Windows\system32\msvcrt.dll
0x000007fefec40000            0x1f000             0xffff C:\Windows\SYSTEM32\sechost.dll
0x000007feff540000           0x12d000             0xffff C:\Windows\system32\RPCRT4.dll
0x000007fef3ec0000            0xfb000                0x1 c:\program files\windows defender\mpsvc.dll
0x000007feff2e0000            0xdb000               0x15 C:\Windows\system32\ADVAPI32.dll
0x000007feffc90000           0x203000                0xd C:\Windows\system32\ole32.dll
0x000007feffab0000            0x67000               0x34 C:\Windows\system32\GDI32.dll
0x0000000077a90000            0xfa000               0x3b C:\Windows\system32\USER32.dll
0x000007fefec60000             0xe000                0xa C:\Windows\system32\LPK.dll
0x000007feff210000            0xc9000                0xa C:\Windows\system32\USP10.dll
0x000007fefc9e0000            0x11000                0x1 C:\Windows\system32\WTSAPI32.dll
0x00000000758a0000             0x3000                0x1 C:\Windows\system32\sfc.dll
0x000007fefa6c0000            0x10000                0x1 C:\Windows\system32\sfc_os.DLL
0x000007fef4030000            0x90000                0x2 c:\program files\windows defender\MpClient.dll
0x000007feff9d0000            0xd7000                0x7 C:\Windows\system32\OLEAUT32.dll
0x000007fefce20000            0x1e000                0x3 C:\Windows\system32\USERENV.dll
0x000007fefdae0000             0xf000                0x4 C:\Windows\system32\profapi.dll
0x000007fefdcc0000             0x3a000               0x4 C:\Windows\system32\WINTRUST.dll
0x000007fefdd40000           0x167000                0x9 C:\Windows\system32\CRYPT32.dll
0x000007fefdb80000             0xf000                0xc C:\Windows\system32\MSASN1.dll
0x000007fefcda0000             0xc000                0x4 C:\Windows\system32\VERSION.dll
0x000007fefdeb0000            0xd88000               0x2 C:\Windows\system32\SHELL32.dll
0x000007feff6f0000            0x71000                0x8 C:\Windows\system32\SHLWAPI.dll
0x000007feff130000            0x2e000                0x2 C:\Windows\system32\IMM32.DLL
0x000007feffb20000           0x109000                0x1 C:\Windows\system32\MSCTF.dll
0x000007fefce00000             0x1b000               0x4 C:\Windows\System32\GPAPI.dll
0x000007fefd370000            0x17000                0x4 C:\Windows\System32\CRYPTSP.dll
0x000007fefd070000            0x47000                0x1 C:\Windows\system32\rsaenh.dll
0x000007fefd9d0000             0xf000                0x2 C:\Windows\System32\CRYPTBASE.dll
0x000007fefec70000            0x17000                0x1 C:\Windows\system32\imagehlp.dll
0x000007fefd4c0000            0x22000                0x9 C:\Windows\System32\bcrypt.dll
0x000007fefcfb0000             0x4c000               0x1 C:\Windows\system32\bcryptprimitives.dll
0x000007fefd4f0000             0x4d000               0x1 C:\Windows\System32\ncrypt.dll
0x000007fef5fb0000            0x35000                0x1 c:\program files\windows defender\mprtp.dll
0x0000000077d50000             0x7000                0x2 C:\Windows\system32\PSAPI.DLL
0x000007fef2fb0000             0xd3000               0x1 C:\Windows\System32\tdh.dll
0x0000000074fc0000           0x7d4000                0x1 C:\ProgramData\Microsoft\Windows Defender\Definition Updates\{D2B0B133-42ED-44D3-809A
-46EBB62BA863}\mpengine.dll
0x000007fefcd70000            0x27000                0x1 C:\Windows\system32\iphlpapi.dll
0x000007feff160000             0x8000                0x3 C:\Windows\system32\NSI.dll
0x000007fefcd60000             0xb000                0x1 C:\Windows\System32\WINNSI.DLL
0x000007fefef10000             0x4d000               0x1 C:\Windows\system32\WS2_32.dll
0x000007fefcd30000             0x2d000               0x1 C:\Windows\System32\ntmarta.dll
0x000007feffc30000             0x52000               0x1 C:\Windows\system32\WLDAP32.dll
0x000007fefd910000             0xb000                0x1 C:\Windows\System32\secur32.dll
```

KUVIO 48. *svchost.exe*-prosessin osittain lataamat .dll-tiedostot.

Svchost.exe-prosessi isännöi tässä tapauksessa Windows Defender:n palvelua. Listasta nähtiin, että prosessi lataa kuitenkin mm. *RpcRtRemote.dll*-tiedostoa, joka liittyy etäproseduurikutsujen suorittamiseen. Tiedonhaku ei todistanut, että Windows Defender käyttäisi RPC-protokollaa. Toiminta voitaisiin siis todeta epäilyttäväksi.



```
wmpnetwk.exe pid:   2464
Command line : "C:\Program Files\Windows Media Player\wmpnetwk.exe"


Base                            Size          LoadCount Path
------------------ ------------------ ------------------ ----
0x00000000ff190000           0x17c000             0xffff C:\Program Files\Windows Media Player\wmpnetwk.exe
0x0000000077b90000           0x1a9000             0xffff C:\Windows\SYSTEM32\ntdll.dll
0x0000000077970000           0x11f000             0xffff C:\Windows\system32\kernel32.dll
0x000007fefdc50000            0x6c000             0xffff C:\Windows\system32\KERNELBASE.dll
0x000007feff2e0000            0xdb000             0xffff C:\Windows\system32\ADVAPI32.dll
0x000007feff090000            0x9f000             0xffff C:\Windows\system32\msvcrt.dll
0x000007fefec40000            0x1f000             0xffff C:\Windows\SYSTEM32\sechost.dll
0x000007feff540000           0x12d000             0xffff C:\Windows\system32\RPCRT4.dll
0x0000000077a90000            0xfa000             0xffff C:\Windows\system32\USER32.dll
0x000007feffab0000            0x67000             0xffff C:\Windows\system32\GDI32.dll
0x000007fefec60000             0xe000             0xffff C:\Windows\system32\LPK.dll
0x000007feff210000            0xc9000             0xffff C:\Windows\system32\USP10.dll
0x000007feff9d0000            0xd7000             0xffff C:\Windows\system32\OLEAUT32.dll
0x000007feffc90000           0x203000             0xffff C:\Windows\system32\ole32.dll
0x000007fefb410000             0x9000             0xffff C:\Windows\system32\WSOCK32.dll
0x000007fefef10000            0x4d000             0xffff C:\Windows\system32\WS2_32.dll
0x000007feff160000             0x8000             0xffff C:\Windows\system32\NSI.dll
0x000007fefcd70000            0x27000             0xffff C:\Windows\system32\IPHLPAPI.DLL
0x000007fefcd60000             0xb000             0xffff C:\Windows\system32\WINNSI.DLL
0x000007feff6f0000            0x71000             0xffff C:\Windows\system32\SHLWAPI.dll
0x000007fefce20000            0x1e000             0xffff C:\Windows\system32\USERENV.dll
0x000007fefdae0000             0xf000             0xffff C:\Windows\system32\profapi.dll
0x000007fefc9e0000            0x11000             0xffff C:\Windows\system32\WTSAPI32.dll
0x000007feff130000            0x2e000                0x2 C:\Windows\system32\IMM32.DLL
0x000007feffb20000           0x109000                0x1 C:\Windows\system32\MSCTF.dll
0x000007fefd9d0000             0xf000                0x2 C:\Windows\system32\CRYPTBASE.dll
0x000007fefdaa0000            0x3d000                0x2 C:\Windows\system32\WINSTA.dll
0x000007fefcd30000            0x2d000                0x1 C:\Windows\system32\ntmarta.dll
0x000007feffc30000            0x52000                0x1 C:\Windows\system32\WLDAP32.dll
0x000007fef3e20000            0x9e000                0x1 C:\Windows\system32\wmdrmdev.dll
0x000007fef3cf0000           0x128000                0x1 C:\Windows\system32\drmv2clt.dll
0x000007fefcda0000             0xc000                0x7 C:\Windows\system32\VERSION.dll
0x000007fef5980000            0x6c000             0xffff C:\Windows\system32\MFPlat.DLL
0x000007fefbb60000             0x9000             0xffff C:\Windows\system32\AVRT.dll
0x000007fefed30000           0x1d7000                0x5 C:\Windows\system32\SETUPAPI.dll
0x000007fefdd00000            0x36000                0xc C:\Windows\system32\CFGMGR32.dll
0x000007fefdc30000            0x1a000                0x5 C:\Windows\system32\DEVOBJ.dll
0x000007fefdeb0000           0xd88000                0xb C:\Windows\system32\SHELL32.dll
0x000007fefdcc0000            0x3a000                0x3 C:\Windows\system32\WINTRUST.dll
0x000007fefdd40000           0x167000                0x7 C:\Windows\system32\CRYPT32.dll
0x000007fefdb80000             0xf000                0xa C:\Windows\system32\MSASN1.dll
0x000007fefec90000            0x99000                0x1 C:\Windows\system32\CLBCatQ.DLL
0x000007fefd370000            0x17000                0x2 C:\Windows\system32\CRYPTSP.dll
0x000007fefd070000            0x47000                0x1 C:\Windows\system32\rsaenh.dll
0x000007fefda80000            0x14000                0x1 C:\Windows\system32\RpcRtRemote.dll
0x000007fef6f60000            0x45000                0x1 C:\Windows\system32\upnp.dll
0x000007fefa410000            0x71000                0x2 C:\Windows\system32\WINHTTP.dll
0x000007fefa3a0000            0x64000                0x2 C:\Windows\system32\webio.dll
0x000007fefa200000            0x11000                0x2 C:\Windows\system32\SSDPAPI.dll
0x000007fefd9e0000            0x91000                0x1 C:\Windows\system32\SXS.DLL
0x000007fefb3b0000            0x11000                0x1 C:\Windows\system32\dhcpcsvc6.DLL
0x000007fefbae0000            0x18000                0x1 C:\Windows\system32\dhcpcsvc.DLL
0x000007fef1770000           0xe0c000                0x1 C:\Windows\system32\wmp.dll
```

KUVIO 49. *wmpnetwk.exe*-prosessin osittain lataamat .dll-tiedostot.

wmpnetwk.exe-prosessi on Windows Media Player:iin liittyvä median jakamiseen tarkoitettu ohjelma. Sen lataamia .dll-tiedostoja tarkkailemalla ei huomattu mitään erikoista, useat niistä olivat verkkoliikenteeseen liittyviä, mutta se on varmaankin odotettavissa median jakoon tarkoitetulla ohjelmalla.

Lopuksi suoritettiin vielä ”malfind”-liitännäinen, joka etsii muistikaappauksesta koodi-injektioita.

Se löytää niitä skannaamalla prosessien muistialueista löytyviä VAD (Virtual Address Descriptor) -tietueita. Näistä tietueista selviää mm. prosessin muistialueen suojausasetus. Normaalisti luodulla prosessilla muistialueen suojaus on yleensä ”*PAGE_EXECUTE_WRITECOPY*”, joka tarkoittaa, että muistialueen voi suorittaa, mutta se on kirjoitussuojattu, eli read-only-tilassa. Muistialueelle on asetettu myös ”copy-on-write”-asetus. Koodi-injektiot voidaan lähes aina havaita suojauksesta ”*PAGE_EXECUTE_READWRITE*”, jolloin muistialuetta voi-

daan suorittaa, lukea ja siihen voidaan kirjoittaa. (Ligh, Case, Levy, & Walters 2014, s. 198, 203, 253.)

Liitännäisen osittaiset tulokset on esitetty kuviossa 50.

```
Process: svchost.exe Pid: 1820 Address: 0x4d90000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 256, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x04d90000  20 00 00 00 e0 ff 0f 00 0c 00 00 00 01 00 05 00   ................
0x04d90010  00 42 00 50 00 30 00 70 00 60 00 00 00 00 00 00   .B.P.0.p.`......
0x04d90020  ba fc ff ff ff 03 55 20 03 55 5c b9 04 00 1a 00   ......U..U\.....
0x04d90030  4c 8b c5 ff 95 e0 37 00 00 8b 4d 24 89 08 48 8d   L.....7...M$..H.

0x04d90000 2000             AND [EAX], AL
0x04d90002 0000             ADD [EAX], AL
0x04d90004 e0ff             LOOPNZ 0x4d90005
0x04d90006 0f000c00         STR WORD [EAX+EAX]
0x04d9000a 0000             ADD [EAX], AL
0x04d9000c 0100             ADD [EAX], EAX
0x04d9000e 0500004200       ADD EAX, 0x420000
0x04d90013 50               PUSH EAX
0x04d90014 0030             ADD [EAX], DH
0x04d90016 007000           ADD [EAX+0x0], DH
0x04d90019 60               PUSHA
0x04d9001a 0000             ADD [EAX], AL
0x04d9001c 0000             ADD [EAX], AL
0x04d9001e 0000             ADD [EAX], AL
0x04d90020 bafcffffff       MOV EDX, 0xfffffffc
0x04d90025 035520           ADD EDX, [EBP+0x20]
0x04d90028 03555c           ADD EDX, [EBP+0x5c]
0x04d9002b b904001a00       MOV ECX, 0x1a0004
0x04d90030 4c               DEC ESP
0x04d90031 8bc5             MOV EAX, EBP
0x04d90033 ff95e0370000     CALL DWORD [EBP+0x37e0]
0x04d90039 8b4d24           MOV ECX, [EBP+0x24]
0x04d9003c 8908             MOV [EAX], ECX
0x04d9003e 48               DEC EAX
0x04d9003f 8d               DB 0x8d
```

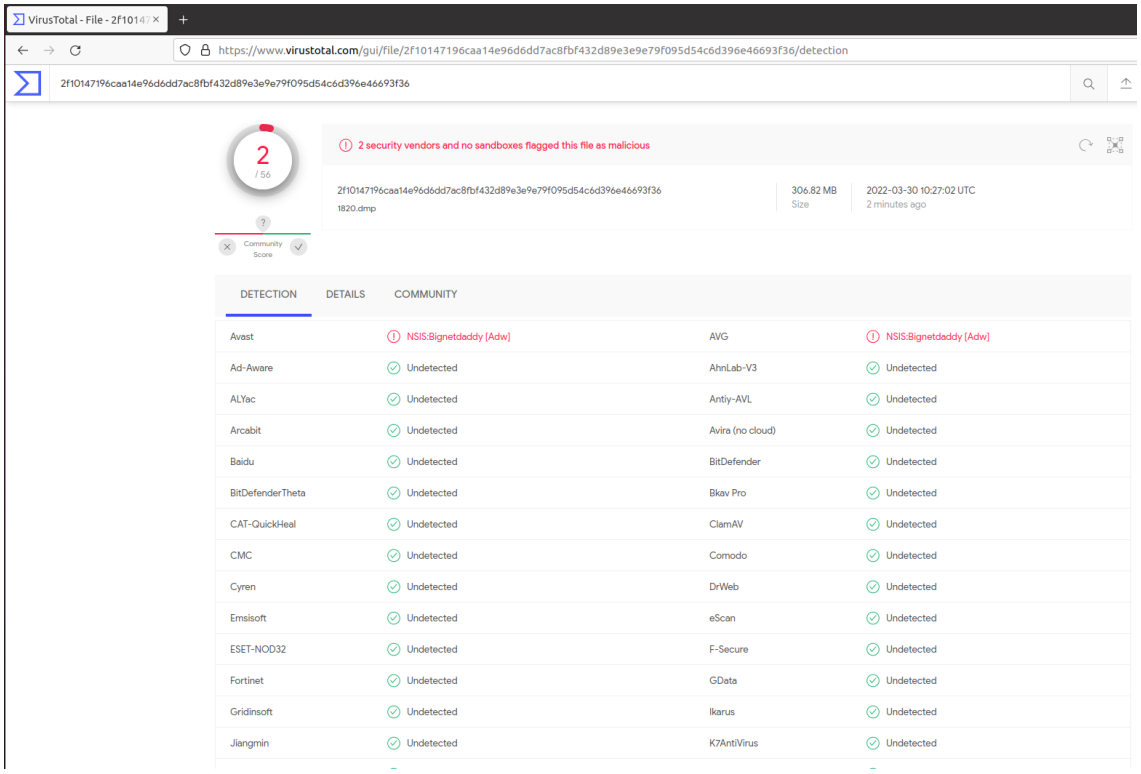
KUVIO 50. ”Malfind”-liitännäisen tuloksia svchost.exe-prosessista.

Liitännäinen esitti viiden eri muistialueen omaavan *”PAGE_EXECUTE_READWRITE”*-muistisuojauksen. Nämä muistialueet kuuluivat prosesseille *explorer.exe*, *svchost.exe* ja *wmpnetwk.exe*. Injektoidut koodit eivät sisältäneet suoritettavia ohjelmia, mutta niiden assembly esitykset näyttivät kuitenkin sisältävän ”oikeanlaisia” prosessorikäskyjä. Näytteen haitallisuudesta ei voitu olla täysin varmoja, mutta jonkinlaista koodia tai komentoja näihin prosesseihin oli kuitenkin injektoitu.

Viimeiseksi yksi prosesseista (*svchost.exe*) dumpattiin kokonaisuudessaan levylle komennolla:

```
$ ./volatility_2.6_lin64_standalone -f victim.raw --profile=Win7SP1x64 memdump -p 1820
-D 1820
```

Prosessista luotu tiedosto ladattiin Virustotaliin skannausta varten. Tulokset on esitetty kuviossa 51.



KUVIO 51. Virustotal skannaustulokset svchost.exe-prosessista.

Tuloksista nähtiin, että 2 antivirusskanneria tunnisti prosessin sisältävän haitallisen ohjelman NSIS:Bignetdaddy [Adw]-mainoshaittaohjelman.

Lopulta tällä analyysilla ei voitu täydellä varmuudella sanoa, sisältääkö muistikaappaus haitallista koodia, vai ei. Emotet on ilmeisesti hankala havaita ja on myös täysin mahdollista, että se on tässä tapauksessa onnistunut välttämään havaitsemista. Mitään kyseiseen haittaohjelmaan viittaavaa ei kuitenkaan löydetty.

6.3 Analyysiesimerkki 2, Cridex

Muistianalyysin avulla tarkasteltiin vielä toista muistikaappausta, joka oli tällä kertaa ilmeisesti tartutettu ”Cridex”-haittaohjelmalla. Muistikaappaus ladattiin Volatility:n Github-tietovarastosta, seuraavasta osoitteesta:

https://github.com/volatilityfoundation/volatility/wiki/Memory-Samples

Näytteen analyysi suoritettiin samoilla metodeilla kuin aikaisempikin, ensiksi suorittamalla imageinfo-liitännäinen. Tuloksista kävi selväksi, että näyte oli kaapattu todennäköisimmin 32-bittistä Windows XP-käyttöjärjestelmää käyttäneestä tietokoneesta. Muistikappauksesta selvitettiin jälleen järjestelmässä käynnissä olleet prosessit, tällä kertaa ”psxview”-liitännäisellä. Tämä liitännäinen etsii käynnissä olevia prosesseja 7:llä eri metodilla, ja sillä voidaan havaita prosesseja, jotka yrittävät piilotella muiden prosessilistausliitännäisien skannausmetodeilta. Liitännäisen tulokset on esitetty kuviossa 52.

```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f cridex.vmem psxview
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x02498700 winlogon.exe            608 True   True   True     True   True  True    True
0x02511360 svchost.exe             824 True   True   True     True   True  True    True
0x022e8da0 alg.exe                 788 True   True   True     True   True  True    True
0x020b17b8 spoolsv.exe            1512 True   True   True     True   True  True    True
0x0202ab28 services.exe            652 True   True   True     True   True  True    True
0x02495650 svchost.exe            1220 True   True   True     True   True  True    True
0x0207bda0 reader_sl.exe          1640 True   True   True     True   True  True    True
0x025001d0 svchost.exe            1004 True   True   True     True   True  True    True
0x02029ab8 svchost.exe             908 True   True   True     True   True  True    True
0x023fcda0 wuauclt.exe            1136 True   True   True     True   True  True    True
0x0225bda0 wuauclt.exe            1588 True   True   True     True   True  True    True
0x0202a3b8 lsass.exe               664 True   True   True     True   True  True    True
0x023dea70 explorer.exe           1484 True   True   True     True   True  True    True
0x023dfda0 svchost.exe            1056 True   True   True     True   True  True    True
0x024f1020 smss.exe                368 True   True   True     True   False False   False
0x025c89c8 System                    4 True   True   True     True   False False   False
0x024a0598 csrss.exe               584 True   True   True     True   False True    True
```

KUVIO 52. Psxview-liitännäisen tulokset.

Tuloksista nähtiin, että mikään prosessi ei ole yrittänyt piilottaa itseään toisilta prosessilistausliitännäisiltä, sillä ohjelma ilmoittaa kaikkien prosessien löytyvän pslist- ja psscan-liitännäisten avulla. Tämä selviää ”True”-merkinnästä kahdessa ensimmäisessä sarakkeessa. Tuloksia tarkkailemalla ainoaksi huomiota herättäväksi prosessiksi merkittiin *reader_sl.exe*-prosessi – prosessi ID:llä 1640 -, jonka on avannut 1484 PID:n omaava ”*explorer.exe*”-prosessi. ”*Reader_sl.exe*” on näistä ainoa Windowsille ei-ominainen prosessi.

Suorittamalla ”connections”-liitännäinen, saatiin selville, että muistikaappaushetkellä järjestelmässä oli käynnissä yksi yhteys, joka oli avattu 1484 PID:n omaavan prosessin, eli ”*explorer.exe”*:n puolesta. Yhteys oli avattu osoitteeseen 41.168.5.140, käyttäen http-liikenteen yleistä porttia 8080. Tämä on hyvinkin epäilyttävää, sillä ”*explorer.exe”*-prosessi on Windowsille ominainen, graafisen käyttöliittymän toimintaan liittyvä prosessi, eikä sen tulisi avata verkkoyhteyksiä. Tämä on esitetty kuviossa 53.

```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f cridex.vmem connections
Volatility Foundation Volatility Framework 2.6
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
0x81e87620 172.16.112.128:1038      41.168.5.140:8080         1484
```

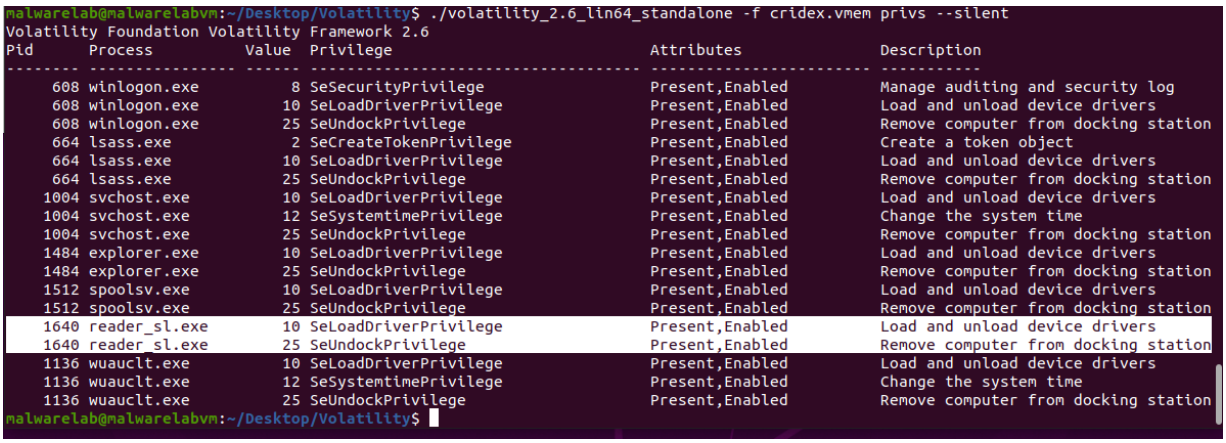
KUVIO 53. Muistikaappauksen hetkellä auki olleet yhteydet.

Seuraavaksi suoritettiin ”connscan”-liitännäinen, jolla nähtiin, että sama prosessi oli avannut toisenkin yhteyden eri osoitteeseen (125.19.103.198) ja sulkenut sen, jolloin sitä ei näkynyt aikaisemman liitännäisen tuloksissa. Tämä on esitetty kuviossa 54.

```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f cridex.vmem connscan
Volatility Foundation Volatility Framework 2.6
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
0x02087620 172.16.112.128:1038      41.168.5.140:8080         1484
0x023a8008 172.16.112.128:1037      125.19.103.198:8080       1484
malwarelab@malwarelabvm:~/Desktop/Volatility$
```

KUVIO 54. ”Connscan”-liitännäisen tulokset.

Suorittamalla ”privs”-liitännäinen ”*–silent*” parametrillä saatiin selville, että aikaisemmin epäilyttäväksi merkitty ”*reader_sl.exe”*-prosessi oli asettanut itselleen oikeudet ladata muistiin laiteajureita. Tämä on esitetty kuviossa 55.



```
malwarelab@malwarelabvm:~/Desktop/Volatility$ ./volatility_2.6_lin64_standalone -f cridex.vmem privs --silent
Volatility Foundation Volatility Framework 2.6
Pid      Process          Value  Privilege                         Attributes              Description
-------- ---------------- ------ --------------------------------- ----------------------- -----------
     608 winlogon.exe          8 SeSecurityPrivilege               Present,Enabled         Manage auditing and security log
     608 winlogon.exe         10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
     608 winlogon.exe         25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
     664 lsass.exe             2 SeCreateTokenPrivilege            Present,Enabled         Create a token object
     664 lsass.exe            10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
     664 lsass.exe            25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
    1004 svchost.exe          10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
    1004 svchost.exe          12 SeSystemtimePrivilege             Present,Enabled         Change the system time
    1004 svchost.exe          25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
    1484 explorer.exe         10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
    1484 explorer.exe         25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
    1512 spoolsv.exe          10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
    1512 spoolsv.exe          25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
    1640 reader_sl.exe        10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
    1640 reader_sl.exe        25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
    1136 wuauclt.exe          10 SeLoadDriverPrivilege             Present,Enabled         Load and unload device drivers
    1136 wuauclt.exe          12 SeSystemtimePrivilege             Present,Enabled         Change the system time
    1136 wuauclt.exe          25 SeUndockPrivilege                 Present,Enabled         Remove computer from docking station
malwarelab@malwarelabvm:~/Desktop/Volatility$
```

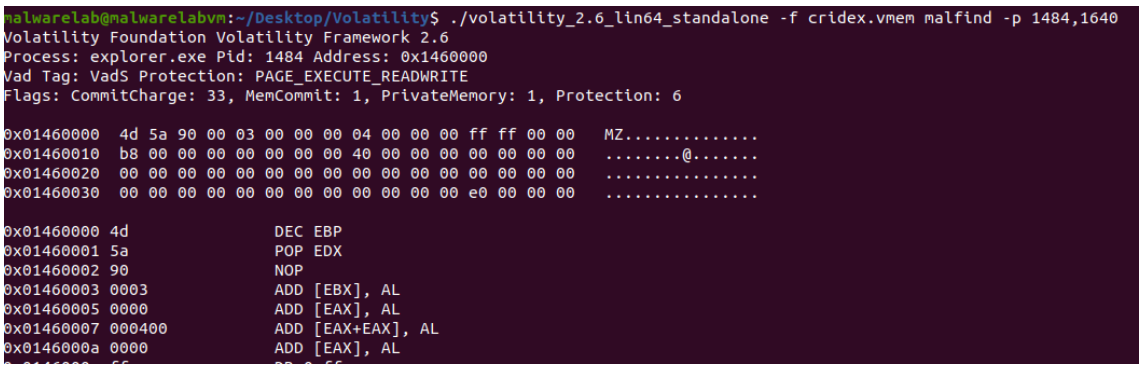
KUVIO 55. ”Privs”-liitännäisen tulokset.

Lyhyellä tiedonhaulla selvisi, että ko. prosessin tulisi olla Adoben Acrobat-dokumenttilukijaan liittyvä prosessi, sen ei varmaankaan tulisi ladata muistiin laiteajureita.

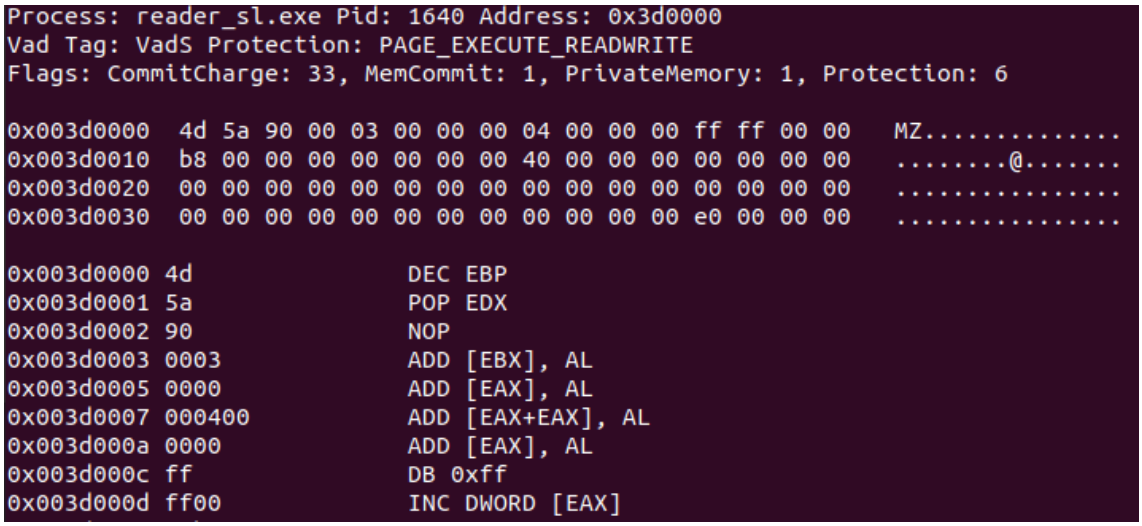
Käytössä olleet ajurit listattiin ”driverscan”-liitännäisellä, mutta niiden joukosta ei huomattu mitään huomioita herättävää.

Seuraavaksi suoritettiin ”mutantscan”-liitännäinen, jolla saatiin selville, että molemmat epäilyttävät prosessit olivat luoneet mutex:eja järjestelmään. ”*explorer.exe”*-prosessi oli luonut mutex:it nimillä: *XMM000005CC*, *XMS8149A9A8* ja *XMQ8149A9A8*. ”*reader_sl.exe”*-prosessi oli taas luonut yhden mutex:in nimellä: *XM00000668*.

Etsimällä koodi-injektioita ”malfind”-liitännäisellä selvitettiin, että molemmat epäilyttävät prosessit olivat injektoineet koodia, jotka tunnistettiin suoritettaviksi ohjelmiksi taikatavujen ”MZ” avulla. Tulokset on esitetty kuvioissa 56 & 57.



KUVIO 56. ”malfind”-liitännäisen tuloksia ”explorer.exe”-prosessista.



KUVIO 57. ”malfind”-liitännäisen tuloksia ”reader_sl.exe”-prosessista.

Nämä injektoidut ohjelmat dumpattiin omiin tiedostoihinsa syöttämällä ”vaddump”-liitännäiselle niiden muistiosoitteiden alut, jotka saatiin äskeisitä tuloksista. Kokonainen komento on seuraavan näköinen:

```
$ mkdir 1484 && ./volatility_2.6_lin64_standalone -f victim.raw --profile=WinXPSP2x86 vaddump -b 0x1460000 -D 1484
```

Jossa ”*mkdir 1484*” luo kansion nimeltään 1484, *”-b”* ilmoittaa liitännäiselle, mistä osoitteesta halutaan luoda kaappaus, *“0x1460000*” on dumpattavan muistialueen alkuosoite ja *”-D 1484*” ilmoittaa mihin polkuun dumpattu tiedosto luodaan. Sama toistettiin toiselle prosessille.

Prosesseista luotiin myös kokonaiset kaappaustiedostot aikaisemmin käytetyllä ”memdump”-liitännäisellä.

Seuraavaksi tarkkailtiin prosessien sisältämiä merkkijonoja Linuxin ”strings”-komennolla. Merkkijonot tallennettiin kaappaustiedostoista, jotka sisälsivät prosessit kokonaisuudessaan, omaan tekstitiedostoihinsa seuraavasti:

$strings 1640.dmp > 1640.txt

Komento toistettiin myös toiselle .dmp-tiedostolle.

”reader_sl.exe”-prosessin muistialueen merkkijonoista löytyi mm. viittauksia eri pankkisivustoihin ja HTML-dataa, jota ohjelma ilmeisesti yrittää tarjota ”authorize.net”-sivustolle. Nämä ovat esitetty kuvioissa 58 & 59.

```
*treasurypathways.com*
*CorporateAccounts*
*weblink.websterbank.com*
*secure7.onlineaccess1.com*
*trz.tranzact.org*
*onlineaccess1.com*
*secureport.texascapitalbank.com*
*/Authentication/zbf/k/*
*ebc_ebc1961*
*tdbank.com*
*online.ovcb.com*
*ebanking-services.com*
*schwab.com*
*billmelater.com*
*chase.com*
*bankofamerica.com*
*pnc.com*
*suntrust.com*
*wellsfargo.com*
*ibanking-services.com*
*bankonline.umpquabank.com*
*servlet/teller*
*nsbank.com*
*securentry.calbanktrust.com*
*securentry*
*/Common/SignOn/Start.asp*
*telepc.net*
*enterprise2.openbank.com*
*BusinessAppsHome*
*global1.onlinebank.com*
*webexpress*
*/sbuser/*
```

KUVIO 58. Eri pankkisivustoja, joita löydettiin prosessin merkkijonoista.

KUVIO 59. Prosessin muistialueesta löydettyä HTML-dataa, joka viittaa authorize.net-sivustoon.

Kuviossa 59 on merkattu myös epäilyttävä merkkijono, joka sisältää web-palvelimen IP-osoitteen 188.40.0.138 ja polun .php-tiedostoon.

Merkkijonoista etsittiin viittauksia aikaisemmin nähtyyn IP-osoitteeseen 41.168.5.140 komennolla:

$grep 41.168.5.140 -C 10 1640.txt

”Grep” on Linux-komento, jolla voidaan etsiä jotain tiettyä merkkiä, tai merkkijonoa sille annetusta syötteestä, tässä tapauksessa yllä näkyvä IP-osoite. ”-C 10” parametrillä tulostetaan osuman sisältäneen rivin lisäksi 10 aikaisempaa ja 10 seuraavaa riviä kokonaisuudessaan. Tulokset on esitetty kuviossa 60.

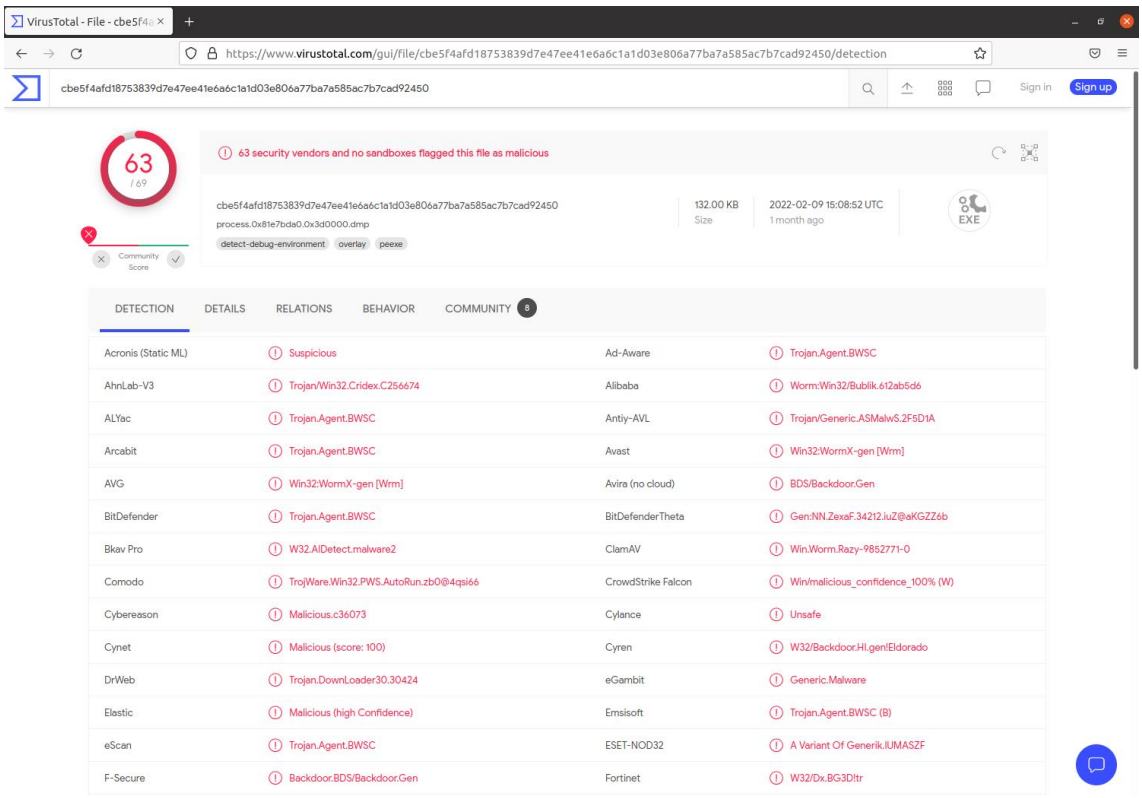
KUVIO 60. Aikaisemmin havaittu IP-osoite merkkijonoissa.

Tuloksista voitiin epäillä, että ohjelma lähettää IP-osoitteeseen POST-metodilla dataa, polkuun *"/zb/v_01_a/in/".* Toiseen aikaisemmin havaittuun IP-osoitteeseen ei löydetty viittauksia.

Merkkijonoista löytyi monia viittauksia API-kutsuihin, joita ohjelma käyttää toimintansa suorittamiseen ja tiedon keräämiseen tietokoneelta.

Toisen prosessin merkkijonoista löydettiin viittaus IP-osoitteeseen 125.19.103.198, joka oli identtinen kuviossa 60 esitetyn tuloksen kanssa. Tämä prosessi sisälsi kaikki "*reader_sl.exe*"-prosessin muistialueen merkkijonot, mutta ei paljoa uutta tietoa esittävää.

Seuraavaksi *"reader_sl.exe"*-prosessin muistialueeseen injektoidusta koodista luotu .dmp-tiedosto ladattiin Virustotaliin skannausta varten. Tulokset on esitetty kuviossa 61.



KUVIO 61. Injektoidun ohjelman Virustotal tulokset.

Tulokset olivat hyvinkin selkeät, 63 skanneria 69:stä tunnistivat muistialueeseen injektoidun ohjelman haitalliseksi, tosin eri nimikkeillä. Troijalainen ja ”downloader” näytti olevan yleisin merkintä, mutta yksi skanneri tunnisti sen ”Cridex”-haittaohjelmaksi.

”explorer.exe”-prosessiin injektoidusta koodista luotu .dmp-tiedosto ladattiin myös Virustotaliin, ja se esittikin samanlaisia tuloksia, tällä kertaa useampikin skanneri tunnisti näytteen nimikkeellä ”Cridex”.

Yhteenvetona analyysillä saatiin kerättyä hyvinkin paljon tietoa ohjelmasta, joka voitiin lopulta tunnistaakin pankkitroijalaiseksi. Näytteen IOC:eihin kuuluuivat mm. IP-osoitteet 41.168.5.140, 125.19.103.198 ja 188.40.0.138, lukuisat pankkisivustojen nimet merkkijonoissa, *polku ”/zb/v_01_a/in/”*, sekä sen piiloutuminen *”reader_sl.exe”*-prosessin ”taakse”.

7 POHDINTA

Opinnäytetyön tavoitteena oli kerätä tietoa haittaohjelmista, niiden toiminnasta ja vaikutuksesta ja niihin liittyvistä puolustusmetodeista, sekä suorittaa haittaohjelmien staattista ja dynaamista analyysia omassa virtuaalisessa laboratorioympäristössä, erityisesti suorittaen muistianalyysiä. Työn teoriaosuudesta tulikin jokseenkin laaja ja ehkä hieman pitkähkö, mutta yleisesti onnistunut tietopaketti haittaohjelmista ja niiden analysoinnista. Teoriaosuuden lähteinä käytettiin enimmäkseen aiheeseen liittyvää kirjallisuutta.

Työn analysointiosiossa suoritettiin analyysia omassa virtuaaliympäristössä kahdelle eri ns. ”oikealle” haittaohjelmanäytteelle staattisin ja dynaamisin metodein. Ensimmäisen analyysin kohdalla ilmeni ongelmia näytteen dynaamisessa analyysissa, mutta kokonaisuudessaan molemmista näytteistä saatiin kerättyä hyvin informaatiota käyttäen teoriaosuudessa avattuja analyysitapoja.

Seuraavaksi suoritettiin muistianalyysia Volatility-ohjelmalla kahdelle muistikaappaukselle, jotka olivat kaapattu tartunnan saaneista järjestelmistä. Analyysin helpottamiseksi näytteet ladattiin Internetistä, eikä kaappauksia otettu omista virtuaalikoneista, vaikka sekin olisi ollut mahdollista. Ensimmäisen näytteen kohdalla ei ollut varmaa oliko järjestelmä oikeastaan saanut tartunnan, vai ei, vaikka itse analyysi oli onnistunut hyvin. Epävarmuutta loi tilanteeseen se, että muistikaappauksen järjestelmä oli väitetysti tartutettu ”Emotet”-haittaohjelmalla, joka on hyvinkin vaikea havaita. Toisessa analyysissä ei esiintynyt ongelmia ja näytteestä kävi selväksi, että järjestelmä oli tartutettu ”Cridex”-pankkiiritroijalaisella.

Työn aihe oli itselleni uusi, ja varsinkin alkuvaiheissa suhteellisen haastava informaation määrän vuoksi. Hankaluutta lisäsi se, että alkuvaiheessa oli vaikea tietää, mihin tietoon tulisi keskittyä. Työn tavoitteet kuitenkin enimmäkseen täyttyivät ja lopullista tulosta voidaan pitää onnistuneena. Työn aihe valittiin kiinnostuksesta kyberturvallisuuteen.

Virtuaaliympäristön kanssa syntyi eniten ongelmia, koneet toimivat alussa hitaasti ja silloin tällöin ne saattoivat jumittua kokonaan, ongelmat saatiin tosin korjattua säätämällä virtuaalikoneiden asetuksia Virtualbox:ssa.

Työssä oli alun perin tarkoitus myös käsitellä koodianalyysiä tarkemmin, mutta se todistautui liian työlääksi, ottaen huomioon työn pituuden ja laajuuden ilmankin sen käsittelyä. Siksi työn jatkoksi voitaisiin ehdottaa haittaohjelmanäytteiden analysointia koodianalyysia käyttäen, keskittyen enimmäkseen assemblyyn ja takaisinmallinnusmenetelmiin. Jos tekisin työn uudestaan, rajaisin sen aluetta paremmin, keskittyen enimmäkseen muistianalyysiin ja varsinkin yrittäisin tuottaa vielä tarkempia kuvauksia käsiteltyjen haittaohjelmien toiminnasta.

LÄHTEET

AV-ATLAS. 2022. Malware & PUA overview. Distribution of malware and PUA by operating system. Verkkosivu. Päivitetty 2022. Viitattu 18.2.2022. https://portal.av-atlas.org/malware

Corfield, G. 2020. Doppelpaymer ransomware crew fingered for attack on German hospital that caused death of a patient. The Register artikkeli. Viitattu 11.1.2022. https://www.theregister.com/2020/09/23/doppelpaymer_german_hospital_ransomware/

Europol. 2021. World's most dangerous malware EMOTET disrupted through global action. Verkkosivu. Päivitetty 18.11.2021. Viitattu 13.1.2022. https://www.europol.europa.eu/media-press/newsroom/news/world%E2%80%99s-most-dangerous-malware-emotet-disrupted-through-global-action

Fortinet. n.d. Trojan Horse Virus. Verkkosivu. Viitattu 6.1.2022. https://www.fortinet.com/resources/cyberglossary/trojan-horse-virus

F-secure. n.d. Trojan-Dropper:W32/Stuxnet. Verkkosivu. Viitattu 8.1.2022. https://www.f-secure.com/v-descs/trojan-dropper_w32_stuxnet.shtml

Hosseini, A. 2017. Ten process injection techniques: A technical survey of common and trending process injection techniques. Julkaistu 18.7.2017 Verkkosivu. Viitattu 22.2.2022. https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process

Johansen, G. 2020. Digital Forensics and Incident Response – Second Edition. Kappale 4. Specialist Topics. YARA. Birmingham: Packt Publishing Ltd. Viitattu 31.1.2022.

Kaspersky. 2021. DarkChronicles: the consequences of the Colonial Pipeline attack. Verkkosivu. Julkaistu 21.5.2021. Viitattu 18.1.2022. https://ics-cert.kaspersky.com/publications/reports/2021/05/21/darkchronicles-the-consequences-of-the-colonial-pipeline-attack/

Kaspersky. n.d. What Is a Drive by Download. Verkkosivu. Viitattu 9.1.2022. https://www.kaspersky.com/resource-center/definitions/drive-by-download

Ligh, M., Case, A., Levy, J. & Walters, A. 2014. The art of memory forensics. 1. painos. Introduction. Indianapolis: John Wiley & Sons, Inc.

Loman, M., Gallagher, S. & Ajjan, A. 2021. Independence Day: REvil uses supply chain exploit to attack hundreds of businesses. Artikkeli. Julkaistu 4.7.2021. Viitattu 10.1.2022. https://news.sophos.com/en-us/2021/07/04/independence-day-revil-uses-supply-chain-exploit-to-attack-hundreds-of-businesses/

Malwarebytes. n.d. a Malware. What are the most common forms of malware? Verkkosivu. Viitattu 3.1.2022.
https://www.malwarebytes.com/malware

Malwarebytes. n.d. b Exploits. What is a zero-day exploit? Verkkosivu. Viitattu 8.3.2022.
https://www.malwarebytes.com/exploits

Martinez, W. 2021. Major U.S. Pipeline hit by ransomware attack. Kuvio. Poimittu CBC uutisartikkelista. Julkaistu 12.5.2021. Viitattu 20.1.2022.
https://www.cbc.ca/news/business/fuel-shortages-southeastern-us-pipeline-1.6023362

Mohanta, A. & Saldanha, A. 2020. Malware analysis and detection engineering. New York City: Apress Media LLC. Viitattu 11.1.2022.

Moir, R. 2009. Defining Malware: FAQ. Verkkosivu. Viitattu 2.1.2022.
https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10)?redirectedfrom=MSDN

Monnappa, K. 2018. Learning Malware Analysis. Birmingham: Packt Publishing Ltd. Viitattu 3.1.2022.

Morrison, S. 2021. How a major oil pipeline got held for ransom. Vox artikkeli. Päivitetty 8.7.2021. Viitattu 10.1.2022.
https://www.vox.com/recode/22428774/ransomeware-pipeline-colonial-darkside-gas-prices

Morse, A. 2017. Investigation: WannaCry cyber attack and the NHS. PDF-dokumentti. Julkaistu 24.10.2017. Viitattu 11.1.2022.
https://www.nao.org.uk/wp-content/uploads/2017/10/Investigation-WannaCry-cyber-attack-and-the-NHS.pdf

Osborne, C. 2021. Updated Kaseya ransomware attack FAQ: What we know now. ZDNet artikkeli. Julkaistu 23.7.2021. Viitattu 10.1.2022.
https://www.zdnet.com/article/updated-kaseya-ransomware-attack-faq-what-we-know-now/

Owasp. n.d Buffer overflow. Verkkosivu. Viitattu 8.1.2022.
https://owasp.org/www-community/vulnerabilities/Buffer_Overflow#

Sikorksi M. & Honig A. 2012. Practical Malware Analysis. San Francisco: No Starch Press. Viitattu 6.1.2022.

Sophos. 2021. The state of ransomware 2021. PDF-Dokumentti. Julkaistu 19.4.2021. Viitattu 6.1.2022.
https://secure2.sophos.com/en-us/medialibrary/pdfs/whitepaper/sophos-state-of-ransomware-2021-wp.pdf

Tolomei, G. n.d. Computer science, research, data, and code. Virtual memory, paging, and swapping. Wordpress blogi. Viitattu 18.2.2022.

https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/virtual-memory-paging-and-swapping/

Tutorialspoint. n.d. Cryptography Hash functions. Verkkosivu. Viitattu 30.1.2022. https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

Yehoshua, N. & Kosayev, U. 2021. Antivirus Bypass Techniques. Birmingham: Packt Publishing Ltd. Viitattu 20.1.2022.

Zscaler. 2021. Return of Emotet: Malware Analysis. Verkkosivu. Julkaistu 13.12.2021. Viitattu 13.1.2022. https://www.zscaler.com/blogs/security-research/return-emotet-malware-analysis

LIITTEET

Liite 1. Virtuaaliympäristön asennus.

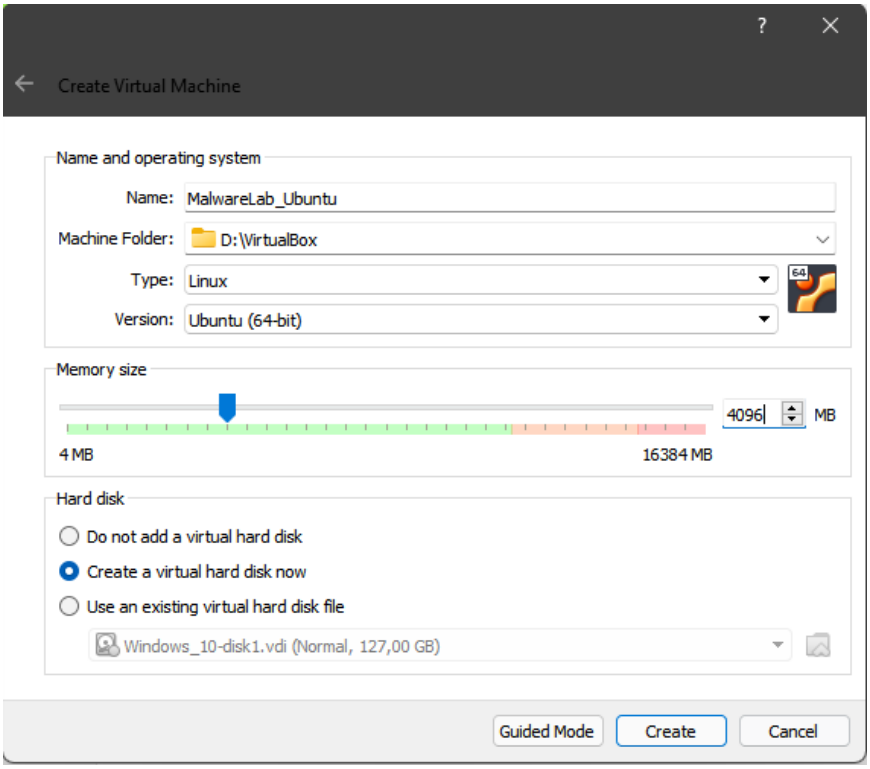
Laboratorioympäristön asennus aloitettiin lataamalla VirtualBoxin & Ubuntun uusimmat versiot työn suorituksen hetkellä, alla olevista osoitteista:
https://releases.ubuntu.com/20.04/

https://www.virtualbox.org/

Windows 10 -virtuaalikoneen kuva ladattiin osoitteesta:

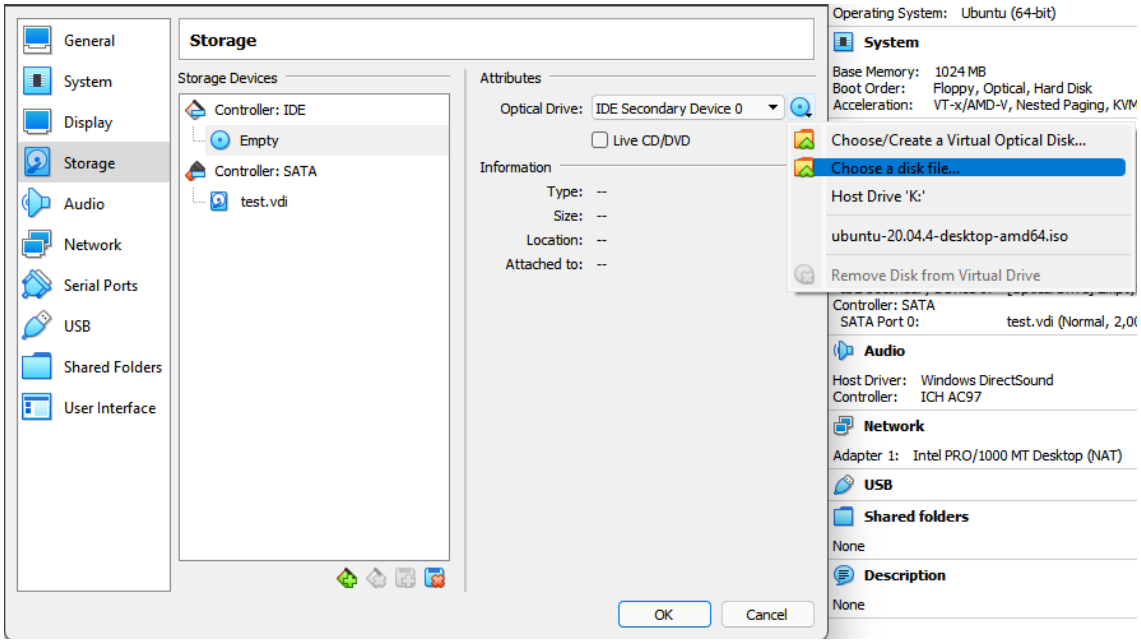
https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

Virtualboxin asentamisen jälkeen ohjelmassa luotiin Ubuntun asennusta varten uusi virtuaalikone valitsemalla *Machine* > *New*. Virtuaalikoneelle annettiin nimi, kohdekansio, sekä valittiin käyttöjärjestelmän tyyppi. Tämä on esitetty kuviossa 62.



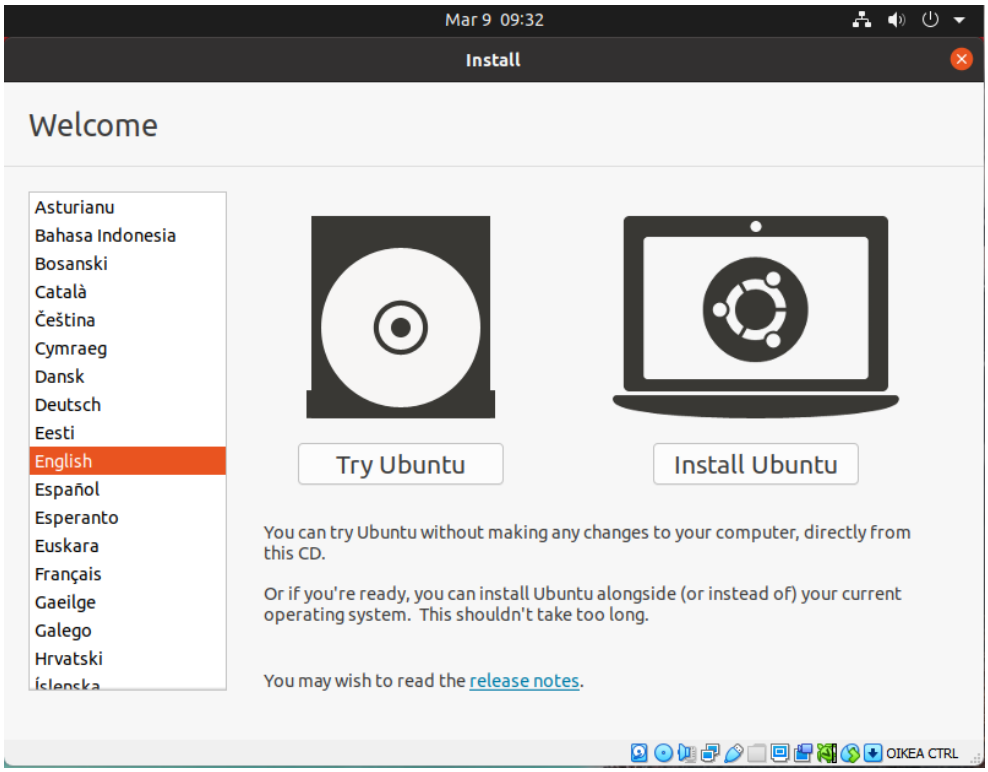
KUVIO 62. Uuden virtuaalikoneen luominen VirtualBoxissa.

Koneelle valittiin haluttu määrä muistia, ja sille luotiin virtuaalikovalevy. Virtuaalikoneen luomisen jälkeen Ubuntun *.iso* asennuskuva ladattiin koneeseen valikoista *Settings* > *Storage* > *Optical Drive* > *Choose a disk file…* Valikko on esitetty kuviossa 63.



KUVIO 63. Ubuntun asennuskuvan lataaminen virtuaalikoneeseen.

Käynnistämällä virtuaalikone ja valitsemalla ”*Install Ubuntu”* päästiin käyttöjärjestelmän asennusprosessiin. Valikko on esitetty kuviossa 64.



KUVIO 65. Ubuntun asennusvalikko.

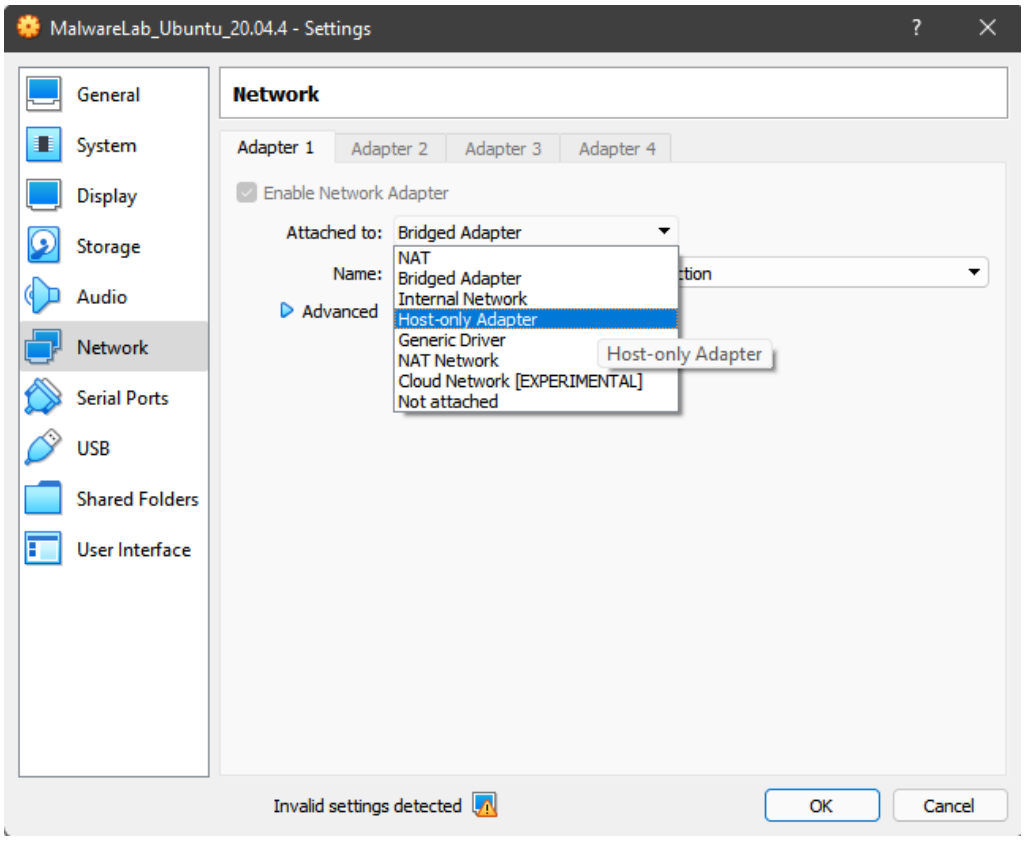
Asennusprosessin jälkeen käyttöjärjestelmälle luotiin käyttäjä ja asennettiin analysointiprosessissa tarvittavia työkaluja.

Verkkopalveluita simuloiva INetSim-, sekä verkkoliikennettä tarkkaileva Wireshark-ohjelmat asennettiin seuraavilla komennoilla:

$ sudo apt-get install inetsim

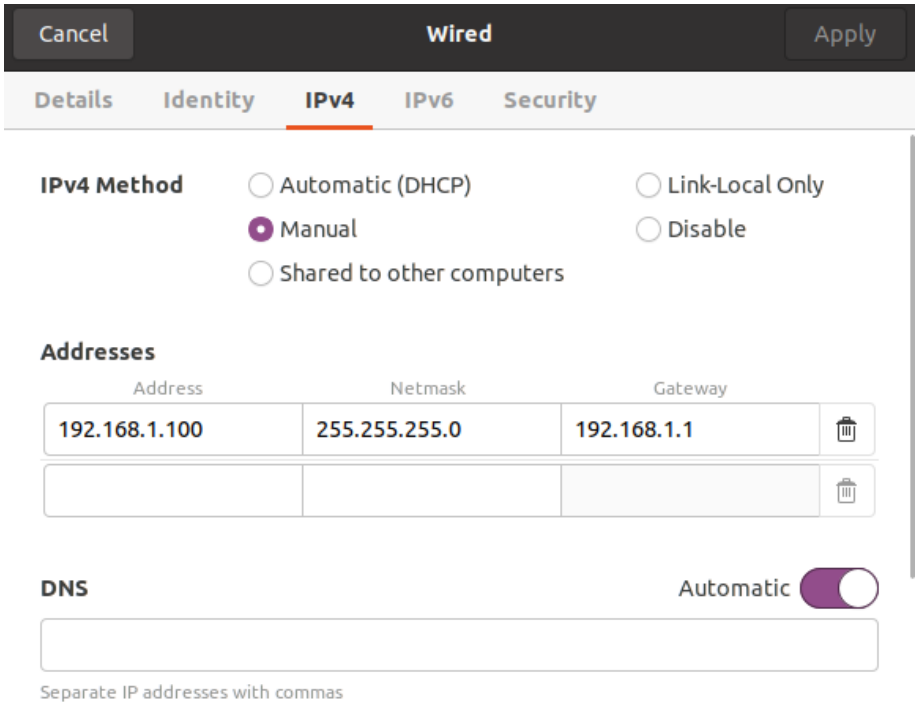
$ sudo apt-get install wireshark

Kun halutut ohjelmat saatiin asennettua, virtuaalikone irrotettiin verkosta asettamalla verkkoadapteriin *"Host-only"*-asetus valikoista *Devices* > *Network* > *Network Settings.* "Host-only"-asetus mahdollistaa useamman virtuaalikoneen yhdistämisen samaan Host:lle asennettuun virtuaaliseen verkkoadapteriin ilman, että ne ovat yhteydessä Internettiin, tai Host-koneen verkkoon. Valikko on esitetty kuviossa 66.



KUVIO 66. Virtuaalikoneen verkkoadapterin asettaminen *"Host-only"-tilaan.*

Virtuaalikoneille luotiin oma verkko asettamalla Linux-koneen IP-osoitteeksi *192.168.1.100, /24 – eli 255.255.255.0* - verkkomaskilla. IP-asetukset on esitetty kuviossa 67.



KUVIO 67. Linux koneen IP-asetukset.

Kun Linux-kone oli isoloitu verkosta, *INetSim* konfiguroitiin käyttämään virtuaalikoneen IP-osoitetta palveluiden simuloinnissa. *INetSim*:n konfiguraatiotiedostoa päästiin muokkaamaan seuraavalla komennolla:

$ sudo gedit /etc/inetsim/inetsim.conf

Simuloidut verkkopalvelut ohjattiin käytössä olevaan IP-osoitteeseen lisäämällä konfiguraatiotiedostoon *service_bind_address*-osion alle rivi:

service_bind_address 192.168.1.100

Ohjelma konfiguroitiin vastaamaan DNS-kyselyihin Linux-koneen IP-osoitteella lisäämällä *dns_default_ip*-osion alle rivi:

dns_default_ip 192.168.1.100

Nämä konfiguraatiotiedoston muutokset on esitetty kuvioissa 68 ja 69.

```
60 ##########################################
61 # service_bind_address
62 #
63 # IP address to bind services to
64 #
65 # Syntax: service_bind_address <IP address>
66 #
67 # Default: 127.0.0.1
68 #
69 #service_bind_address 10.10.10.1
70 service_bind_address 192.168.1.100
71
72 ##########################################
```

KUVIO 68. Inetsim:n konfiguraatiotiedoston service_bind_address-osio muutettuna.

```
198 ##########################################
199 # dns_default_ip
200 #
201 # Default IP address to return with DNS replies
202 #
203 # Syntax: dns_default_ip <IP address>
204 #
205 # Default: 127.0.0.1
206 #
207 #dns_default_ip 10.10.10.1
208 dns_default_ip 192.168.1.100
209
210 ##########################################
```

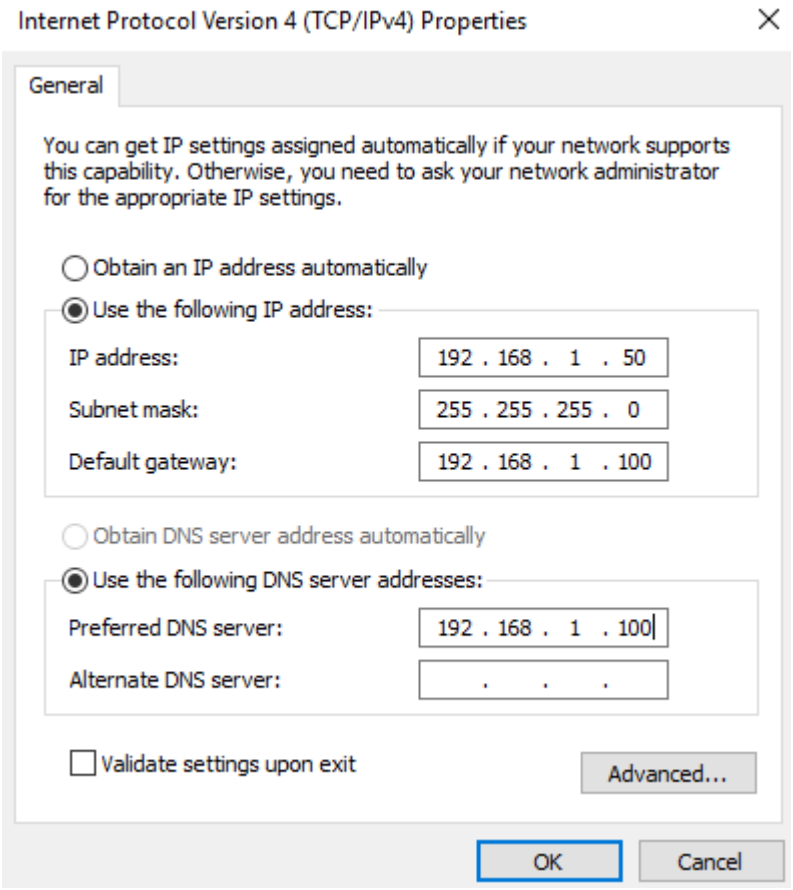
KUVIO 69. Inetsim:n konfiguraatiotiedoston dns_default_ip muutettuna.

INetSim ohjelma käynnistettiin komennolla:

$ sudo inetsim

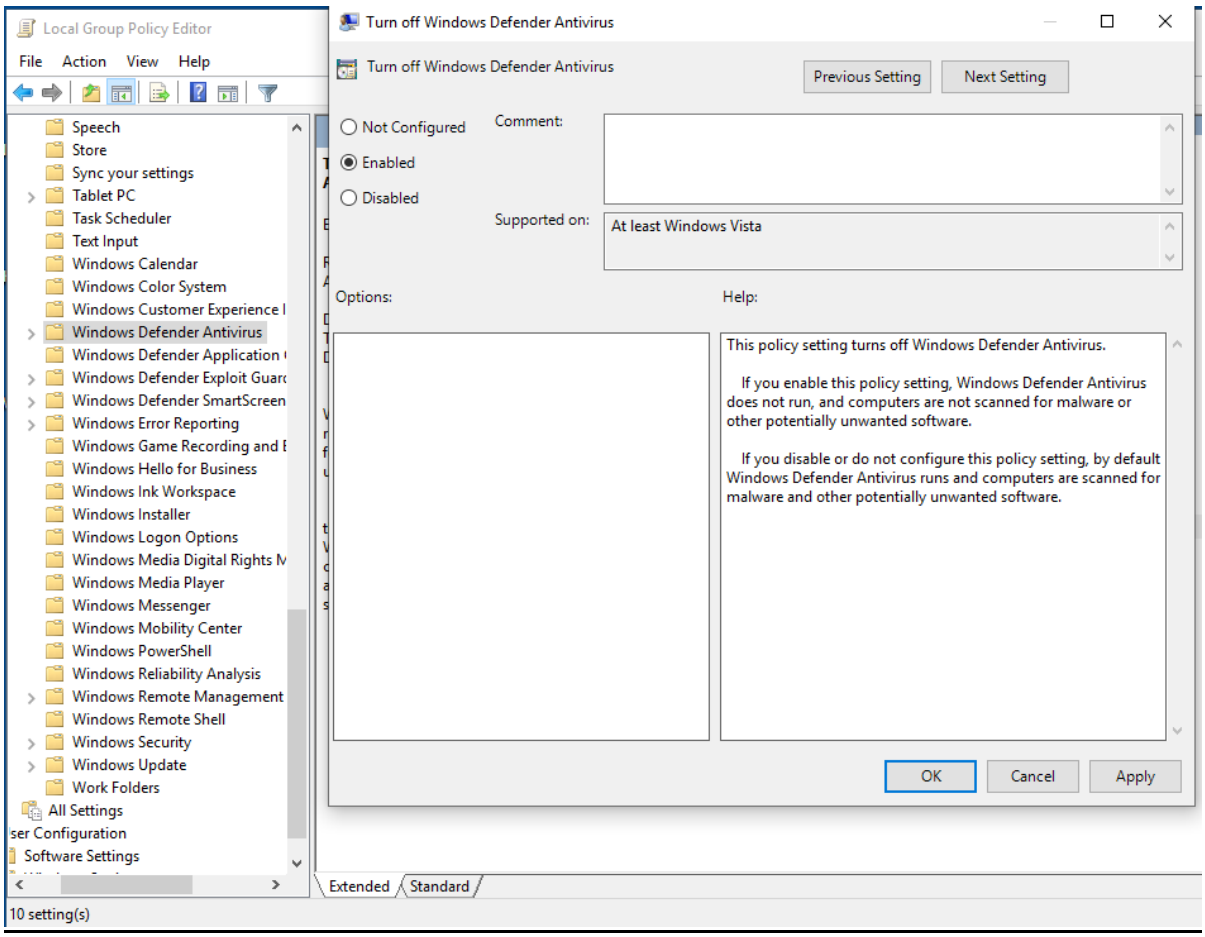
Windows 10 virtuaalikone asennettiin valitsemalla VirtualBox:n asetuksista *File > Import Virtual Appliance* ja valitsemalla ladattu ”*.ova”*-tiedosto. Tässä tapauksessa tiedosto sisältää virtuaalisen kovalevyn, johon on jo valmiiksi asennettu Windows 10 käyttöjärjestelmä ja luotu testausta varten käyttäjä *IEUser*. Käyttäjän salasana on ”*Passw0rd!*”.

Kun Windows-virtuaalikoneelle saatiin asennettua halutut työkalut, se isoloitiin verkosta samalla tavalla kuin Linux-virtuaalikone. Sen IP-osoitteeksi asetettiin *192.168.1.50, /24*-verkkomaskilla, ja oletusyhdyskäytäväksi, sekä DNS palvelimeksi Linux-koneen IP-osoite *192.168.1.100.* VirtualBoxin asetuksista verkkoadapteri asetettiin ”*Host-only”*-tilaan. Windows-koneen IP-asetukset on esitetty kuviossa 70.



KUVIO 70. Windows-koneen IP-asetukset.

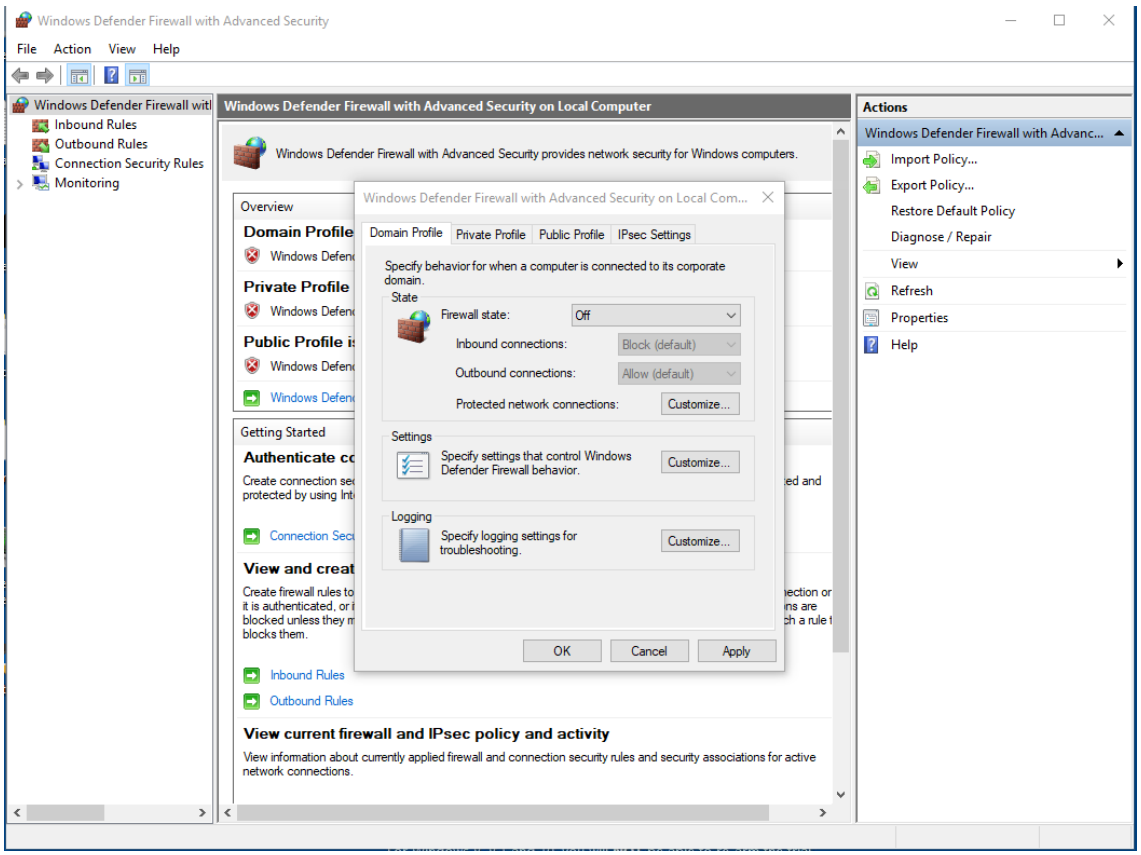
Analyysia varten Windows-koneesta otettiin pois käytöstä Windows Defenderin tarjoamat toiminnot. Tämä tehtiin avaamalla *Local Group Policy Editor* ja valitsemalla valikosta *Computer Configuration* > *Administrative Templates* > *Windows Components* > *Windows Defender Antivirus*. Avautuneista asetuksista valittiin *Turn off Windows Defender Antivirus* ja se asetettiin *Enabled*-tilaan. Tämä on esitetty kuviossa 71.



KUVIO 71. Windows Defenderin antivirus ominaisuuden käytöstä poistaminen.

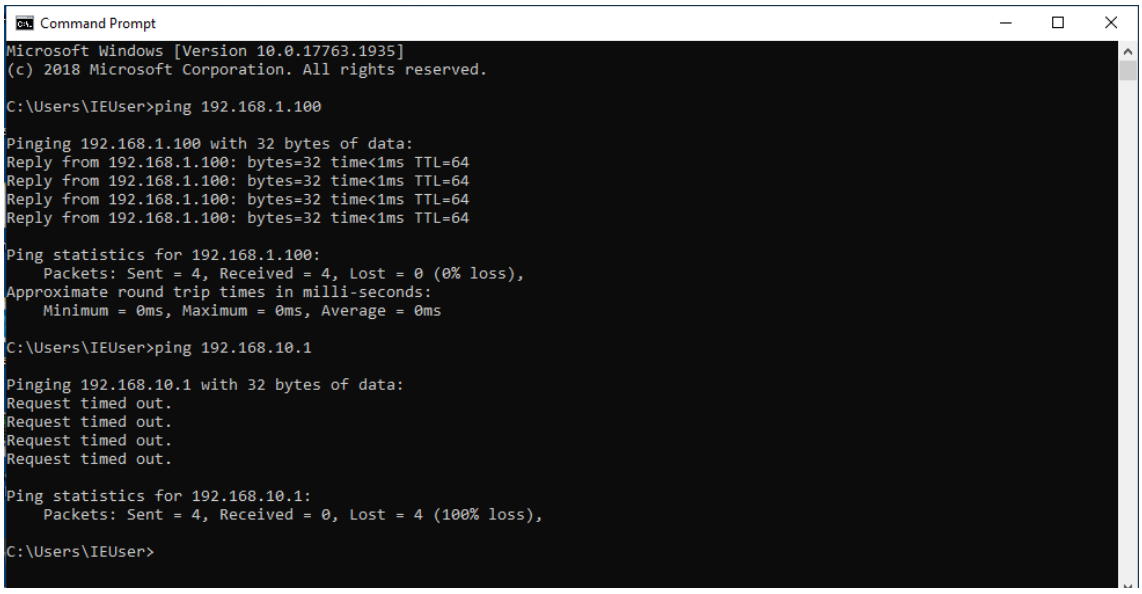
Sama toimenpide toistettiin *Windows Defender Application Smart Guard*-asetukselle.

Myös Windowsin oma palomuurit poistettiin käytöstä valikoista *Settings* > *Network & Security* > *Windows Firewall* > *Advanced Settings* > *Properties* ja asettamalla ”*Firewall state*”-asetus “off”-tilaan, jokaiselle profiilille. Tämä on esitetty kuviossa 72.



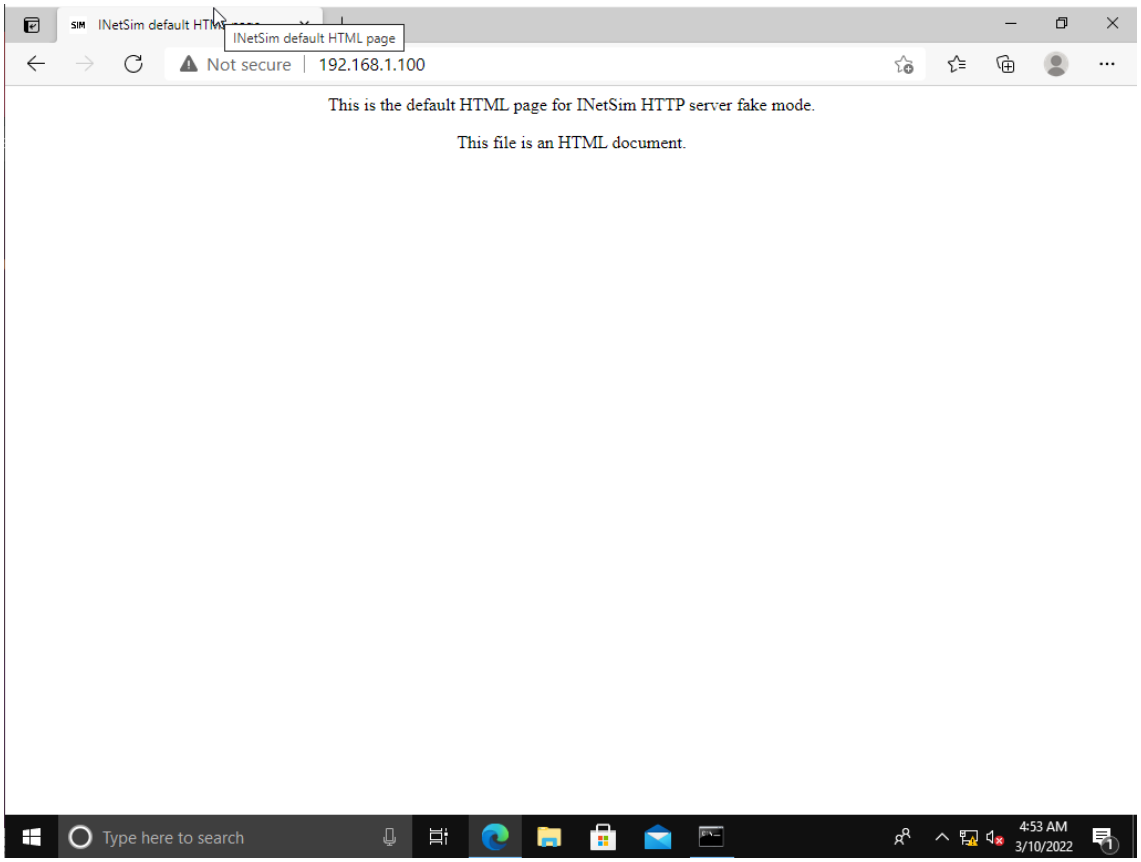
KUVIO 72. Windowsin palomuurin asetukset.

Virtuaalikoneiden välisen yhteyden toimivuus testattiin pingaamalla Linux-konetta Windows-koneesta. Samalla varmistettiin, ettei virtuaalikoneelta kyetty pingaamaan host-koneen verkkoa. Tämä on esitetty kuviossa 73.



KUVIO 73. Windows-koneelta pingattu Linux-koneen ja host-koneen verkkoihin.

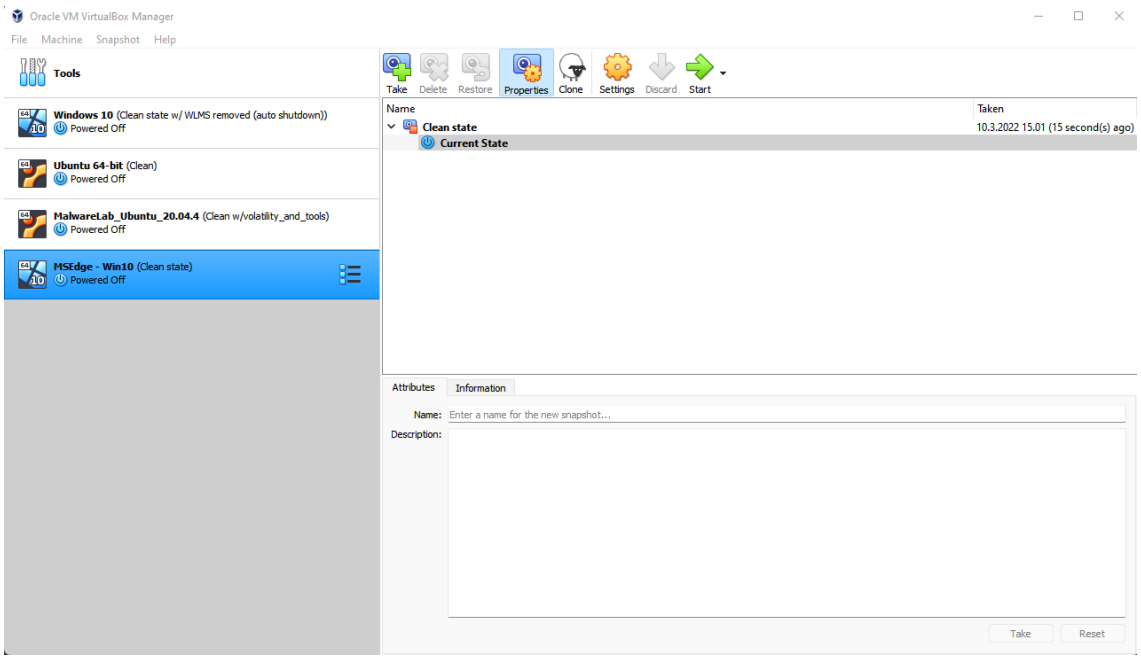
Linux-koneella pyörivän INetSim:n toiminta todistettiin avaamalla Windows-koneella selain, ja navigoimalla sillä Linux-koneen IP-osoitteeseen. INetSim simuloi muiden palveluiden ohessa myös http-palvelinta. Http-palvelin palauttaa pyydettäessä minkä tahansa siltä pyydetyn tiedoston. Kun osoitteeseen navigoidaan selaimella, palvelin lähettää vastaukseksi HTML-tiedoston, joka on esitetty kuviossa 74.



KUVIO 74. INetSim:n avulla simuloitu http-palvelin.

Windows virtuaalikoneesta poistettiin vielä lopuksi VirtualBox:n mukana tulleet *”Guest Addon*” toiminnot.

Nyt kun laboratorioympäristön virtuaalikoneet olivat halutussa tilassa, niistä otettiin kopiot VirtualBox:n snapshot toiminnolla. Tämä onnistui valikoista *Snapshot > Take.* Tämä on esitetty kuviossa 75.



KUVIO 75. VirtualBox:n snapshot toiminnon valikko.

Virtuaalikoneet palautettiin puhtaaseen tilaan samasta valikosta, jokaisen analyysin jälkeen.