Alexander San Miguel

Takaisinmallintamistyökalujen vertailu haittaohjelma-analyysiin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

31.10.2024

Tiivistelmä

Tekijä:	Alexander San Miguel
Otsikko:	Takaisinmallintamistyökalujen vertailu haittaohjelma-analyysiin
Sivumäärä:	41 sivua
Aika:	31.10.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Vanhempi luennoitsija, Amir Dirin

Tämän opinnäytetyön tavoitteena oli tarkastella kahta koodin IDA Freen ja Ghidran avulla ja tutkia niiden käytettävyyttä haittaohjelmien analysoinnissa toisiinsa verrattuna. Arviointi perustui aitoon haittaohjelmanäytetteeseen, joka analysoitiin molempia ohjelmia käyttäen.

Haittaohjelmanäytteen analysointia varten käytettiin virtuaalikonetta. Vieraskäyttöjärjestelmänä käytettiin REMnuxia, jota käytetään yleisesti kyberturvallisuusalalla ohjelmien takaisinmallintamiseen. Virtuaaliympäristö on ihanteellinen haittaohjelmien analysointiin, eristyksen ja ympäristön hallinnan takia.

Prosessiin sisältyi haittaohjelmanäyteen lataaminen sekä IDA Freehen että Ghidraan, minkä jälkeen jatkettiin staattista analysointia käyttämällä ohjelmien sisäänrakennettua automatisoitua työkalua, manuaalista analyysiä, sekä takaisinmallintamistekniikoita haittaohjelmanäytteen käyttämien ominaisuuksien ja salauksen tutkimiseen.

Tutkimus osoitti, sekä IDA:n että Ghidran olevan hyödyllisiä työkaluja ja vaikka IDA on paremmin vakiintunut työkalusarja, Ghidra on nykyaikaisempi, avoimempi ja monin tavoin aloittelijaystävällisempi ohjelmisto haittaohjelmien takaisinmallintamiseen ja niiden tutkimiseen.

Avainsanat: Haittaohjelmat, takaisinmallinnus, Ghidra, IDA

Abstract

Author:	Alexander San Miguel
Title:	Comparison of reverse engineering tools for use in malware analysis
Number of Pages:	41 pages
Date:	31 October 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Amir Dirin, Senior Lecturer

The objective of this thesis was to offer a look at two disassembling tools and compare their usability in analyzing malware. To assist with this evaluation, a real piece of live malware was procured and analyzed using both tools.

A virtual machine was set up for the purpose of analyzing the malware sample. The guest Operating System used was REMnux, which is a purpose-built virtual machine commonly used in the cyber security industry for the purposes of reverse engineering software. The virtual environment is ideal for analyzing malware, due to the inherent isolation and control over the environment itself.

The process involved loading the malware sample into both IDA Free and Ghidra, and then proceeding to statically analyze both using built in automated tooling, manual enumeration, as well as reverse engineering techniques to ascertain capabilities and encryption used by the malware sample.

This study showed that both IDA Free and Ghidra are useful tools, and while the former is a better-established toolset, Ghidra is a more modern, faster and generally more beginner friendly software for the purposes of malware reverse engineering and analysis.

Keywords: Malware, reverse engineering, Ghidra, IDA

Sisällys

Lyhenteet

C2: *Command and Control*. Tietoliikenneinfrastruktuuri, jota hyökkääjät käyttävät verkon vaarantuneiden järjestelmien hallinnassa. Kun haittaohjelma tarttuu laitteeseen, tämä usein muodostaa yhteyden C2-palvelimeen, jonka avulla hyökkääjä voi antaa manuaalisia tai automatisoituja etäkomentoja. Nämä komennot voivat sisältää laitteen tietojen suodattamista tai lisähaittaohjelmien asentamista.

RE: *Reverse Engineering*. Takaisinmallintaminen on analyysimetodi, jolla pyritään ymmärtämään, miten tarkkailtu ohjelmisto tai järjestelmä toimii.

EDR: *Endpoint Detection Response*. Päätepisteiden tunnistaminen ja reagointi uhkiin. Lyhenne viittaa automatisoituihin työkaluihin, jotka valvovat tietoturvauhkia työntekijöiden tietokoneilla, sekä niillä päätepisteillä, joihin ne on asennettu. Näillä työkaluilla pyritään ennaltaehkäisemään tietoturvauhkia, kuten haittaohjelmien lataus-, asennus- ja käyttöönottohetkiä.

IOC: *Indicators of Compromise*. Vaarantumisindikaattori. Tämä on todiste, kuten tietty tiedosto, epätavallinen verkkoliikenne, tai IP-osoite, joka osoittaa tietoturvaloukkauksen tai haitallisen toiminnan tapahtuneen järjestelmässä tai verkossa. Nämä indikaattorit ovat ratkaiseva tekijä kyberuhkien tunnistamisessa, tutkimisessa ja lieventämisessä.

Keylogger: *Keystroke logger.* Haittaohjelma, joka tallentaa tietokoneen näppäinpainallukset.

Malware: *Malicious Software*. Haittaohjelma, joka pyrkii aiheuttamaan vahinkoa uhrille/laitteelle/järjestelmälle.

Trojan: Troijalainen virus on haittaohjelmatyyppi, joka naamioituu toiseksi ohjelmaksi tai tiedostoksi huijatakseen käyttäjää lataamaan ja asentamaan sen laitteellensa.

VM: *Virtual Machine*. Virtuaalikone on fyysisen tietokoneen ohjelmistopohjainen emulointi, joka käyttää käyttöjärjestelmää ja sovelluksia itsenäisesti, jolloin useat virtuaalikoneet voivat toimia yhdessä fyysisessä koneessa.

1 Johdanto

Takaisinmallintamistyökalut ovat pitkään olleet olennainen osa laajempaa kyberturvallisuuden alaa. Nämä työkalut antavat haittaohjelmatutkijoille mahdollisuuden tarkastella haittaohjelmien sisäistä logiikkaa, jonka pohjalta voidaan luoda tehokkaampia vastakeinoja näitä vastaan tunnistamalla mahdollisia vaarantumisindikaattoreita, tai uusia menetelmiä, joita haittaohjelmien tekijät käyttävät.

Takaisinmallintamistyökaluja on monia, mutta erityiseen suosioon on kasvanut sekä IDA (*Interactive Disassembler*) että Ghidra. Molempia käytetään ohjelmien staattiseen analysointiin. Tätä prosessia kutsutaan ohjelman disassembloinniksi tai purkamiseksi, johon kuuluu konekoodin muuntaminen takaisin ihmisen luettavaksi kokoonpanokieleksi, Assembly-ohjelmointikieleksi. Tätä prosessia käyttämällä pyritään tutkimaan, kuinka ohjelma toimii alhaisella tasolla, varsinkin kun tutkittavan ohjelman lähdekoodia ei ole saatavilla. Tämä voi olla välttämätön tehtävissä kuten virheenkorjauksessa, ohjelmiston tietoturva-aukkojen analysoinnissa sekä etenkin ohjelmiston takaisinmallintamisessa.

Tämän opinnäytetyön tarkoituksena on esittää tapaustutkimus, jossa verrataan IDA Freen ja Ghidran välisiä toimintoja sekä kyvykkyyksiä tutkia haittaohjelmia aloittelevan tutkijan näkökulmasta. Opinnäytetyössä keskitytään staattisen analyysin työkaluihin, jolla pyritään minimoimaan tartuntariskeja ja mahdollistamaan turvallisen ja tehokkaan tavan analysoida haittaohjelmanäytettä. Tästä johtuen staattisen analyysin menetelmä on erityisen hyödyllinen haittaohjelmakoodin tunnistamiseen, haitallisten koodirakenteiden ymmärtämiseen ja alustavien vaarantumisindikaattorien tutkimiseen.

Keskittymällä staattisiin analyysitekniikoihin tutkimuksessa arvioidaan molempien työkalujen vahvuuksia ja pyritään antamaan alustava näkemys niiden ominaisuuksista suhteessa toisiinsa. Nämä ohjelmat valittiin erityisesti niiden yleisen saavutettavuuden takia. Molempia pidetään laajasti aloittelijaystävällisinä ja

molemmilla ohjelmistoilla on laaja yhteisö, joka ylläpitää niitä sekä auttaa aloittelevia haittaohjelmatutkijoita.

Jotta vertailu pohjautuu realistiseen tapaukseen, tutkimuksessa käytetään IllusionBot -haittaohjelmanäytettä, joka löytyy sivustolta “https://github.com/ytisf/theZoo/tree/master". Tämä näyte toimii perustana tutkimukselle, jossa korostetaan, miten kukin työkalu käsittelee todellisen haittaohjelman monimutkaisuutta ja haasteita. Tutkimuksessa nähdään yleisiä menetelmiä, joita haittaohjelmien tekijät käyttävät ohjelmakoodin hämärtämiseen sekä kuinka turvallisesti staattista analyysiä voidaan suorittaa todellisessa laboratorioympäristössä. Tutkimuksen avulla myös tarjotaan alustava ymmärrys haittaohjelmien takaisinmallinnuksesta kiinnostuneille ja sitä opiskeleville.

Tutkimuksen tavoitteet ovat seuraavat:

- Tutkimus vertaa IDA Freen ja Ghidran staattisen analyysin ominaisuuksia.
- Tutkimus pyrkii tunnistamaan kunkin työkalun vahvuuksia ja rajoituksia haittaohjelmanäytettä analysoitaessa.
- Tutkimus tarjoaa käytännön näkemyksiä ja suosituksia haittaohjelmien analyytikoille, mitkä koskevat näiden työkalujen käyttöä reaalimaailmassa.

2 Staattisen analyysin tarkoitus haittaohjelmatutkimuksessa

Staattisella analyysillä tarkoitetaan menetelmää, jossa ohjelmakoodia tarkastellaan ja analysoidaan sitä suorittamatta. (Sikorski ja Honig, 2012: 9). Tämä tarkoittaa, että ohjelman toimintaa ja rakennetta tutkitaan pelkän lähdekoodin tai muun koodimuodon avulla. Staattisen analyysin vastakohta on dynaaminen analyysi, jolla tarkoitetaan haittaohjelman suorittamista hallitussa ympäristössä. Tällä pyritään tarkkailla sen käyttäytymistä käytännössä, joka voi usein olla hyödyksi haittaohjelmaa tutkiessa. Tällöin tutkijat kykenevät näkemään reaaliajassa mitä haittaohjelma tekee kohdekoneella, kuten esimerkiksi haitallisten tiedostojen luonnin ja laukaisun, palomuurien asetuksien muutokset, tai muutokset rekisteriin. (Shanice Jones, 2023).

Haittaohjelmia takaisinmallintaessa on otettava huomioon mallinnuksen tarkoitus. Useasti tarkoituksena on erilaisten vaarantumisindikaattorien löytäminen ja tunnistaminen, jolloin tutkijan päämääränä on löytää haittaohjelmasta ne komponentit, joita voidaan käyttää tietyn haittaohjelman tunnistamiseen tulevaisuudessa (Sikorski ja Honig, 2012: 2).

Tästä syystä on tärkeää kaventaa tavoitteita tarkasteltaessa haittaohjelmia. Nämä tavoitteet voivat sisältää vaarantumisindikaattoreita (eng. *Indicators of Compromise, IOC*), jotka kertovat tutkijoilla mitä muutoksia haittaohjelma mahdollisesti tekee laittelle, tai miltä kyseisen haittaohjelman tuottava tietoliikenne näyttää (Fortinet).

Tutkijoita voi myös kiinnostaa haittaohjelman käyttämät pysyvyysmekanismit (eng. *persistence mechanism*), jolla kyseinen ohjelma piilottaa itsensä tarttuneen laitteen sisälle ja jatkaa toimintaansa ilman, että uhri tai laitteen oma EDR huomaa tartunnan tapahtuneen. Näitä mekanismeja voivat olla esimerkiksi erilaisten takaporttien asennus laitteeseen tai Windows-käyttöjärjestelmän rekisteriavainten muutokset, joilla haittaohjelma luo automatisoidun tavan käynnistää itsensä uudelleen laitetta uudelleenkäynnistäessä. (Sikorski ja Honig, 2012: 241).

On myös tärkeää ottaa huomioon itse ohjelmiston käytettävyys ja saavutettavuus. Etenkin tapa jolla takaisinmallinnusohjelmisto käsittelee automatisoitua analysointia, sekä mitä lisäinformaatiota ohjelmisto mahdollisesti antaa epäilyttävien toimintojen korostamiseen on tärkeä. Tutkimuksessa myös otetaan huomioon ohjelmiston käyttökokemus, johon sisältyy haittaohjelmanäytteen purku ja projektin sisällä navigointiin.

2.1 Takaisinmallintamisen tarkoitus

Haittaohjelmia tutkiessa voidaan mieltää yhden tai useamman tavoitteen takaisinmallinnuksen mahdolliseksi kohteeksi. Usein tavoitteena voi olla kyvykkyyksien tunnistus, jolloin näytettä tutkiessa pyritään ymmärtämään, mitä toimintoja

haittaohjelma sisältää. Esimerkiksi tiettyjen verkkoliikenneprotokollien käyttö, kuten yleisesti käytössä oleva *Hypertext Transfer Protocol* (HTTP), voi kertoa paljon haittaohjelman toiminnallisuudesta.

Toisena kohteena saattaa olla vaarantumisindikaattorien tunnistus. Näitä tarkastellessa tutkijat pyrkivät tunnistamaan, mitä muutoksia ohjelma tekee kohdelaitteelle, kuten haitallisten tiedostojen luonnin, tai tietyn verkkoliikenteen tunnistaminen. Usein myös Windowsin rekisteriavainten lisäys tai muokkaus voi olla merkki haitallisesta toiminnallisuudesta (Sikorski ja Honig 2012: 139). Näissä tapauksissa on erityisen tärkeää tunnistaa eri IP-osoitteita, joilla voidaan tunnistaa haitalliset palvelimet myös tulevaisuudessa. Nämä vaarantumisindikaattorit tulee dokumentoida tarkasti jatkotutkimuksia varten.

Haitallisista tiedostoista voidaan myös luoda hajautusarvot (eng. *hashing*). Pienetkin muutokset tiedostoon johtavat täysin erilaiseen hajautusarvoon, mikä tekee siitä hyödyllisen tiedostojen nopeassa tunnistamisessa ja vertailussa. Haittaohjelmatutkimuksessa tiedostojen hajautus auttaa tunnistamaan tunnetut haittaohjelmat vertaamalla hajautustietoja tietokantoihin, varmistamaan tiedostojen eheyden havaitsemalla luvattomat muutokset ja vertailemaan tehokkaasti suuria tietomääriä tarkastamatta koko tiedostoa, mikä auttaa haittaohjelmien luokittelussa ja jäljittämisessä. Moni moderni viruksentorjuntajärjestelmä käyttää hajautusarvoja tunnistamaan haitallisia tiedostoja.

2.2 Disassembloija-työkalut

Disassembloijat ovat koodin purkamiseen tarkoitettuja ohjelmistoja. Näiden tarkoitus on purkaa binäärinäytteet, jotka sisältävät tietokoneen suorittimen suorittamia perusohjeita. Tämä on hyödyllistä silloin, kun ohjelmiston alkuperäinen lähdekoodi ei ole tutkittavissa. Operaattorikoodi (eng. *Operation code, opcode*) sisältää aritmeettisia laskuja, tiedonsiirtoa tai ohjausvirtaa. Konekielessä toimintakoodit esitetään binääri- tai heksadesimaaliarvoina, jotka suoritin purkaa ja suorittaa. Ne muodostavat ohjelman perusosia, jotka mahdollistavat ylemmän

tason ohjelmointikielen kääntämisen suoritettavaksi konekoodiksi. (Eagle 2008: 4.)

Koodin purkaminen näiden ohjelmien avulla tarkoittaa tätä konekoodin kääntämistä takaisin ihmisluettavaan muotoon, yleisesti Assembly-ohjelmointikieleen. Moni disassembloija sisältää kyvykkyyden muuttaa Assembly-ohjelmointikielen ylemmän tason ohjelmointikieleksi käyttämällä *decompile*-toiminnallisuutta. Tyypillisesti tämä johtaa disassembloijan tuottavan C-ohjelmointikielitulkinnan Assembly-kielestä. On silti tärkeä pitää mielessä, että automatisoitu prosessi voi tehdä tämän käännöksen väärin, jolloin perusymmärrys alkuperäisestä Assembly-ohjelmointikielestä auttaa tulkitsemisessa.

2.2.1 IDA Free

IDA Free on vapaaversio IDA Pro:sta, jota Hex-Rays myy. Tätä ohjelmistoa on käytetty vuodesta 1991 asti, minkä takia sen tarjoamat toiminnot ovat laajat.

Hex-Rays tuottaa myös maksullisen version Interactive Disassembler –ohjelmistosta. (Hex-Rays, 2024). Kalleimmillaan tämä lisenssi maksaa 1975 euroa per lisenssi. IDA Free ei sisällä kaikkia IDA Pron tarjoamia ominaisuuksia, kuten yhteystyö-, python-skriptaus- tai binäärin lokaalia takaisinkääntöominaisuuksia. IDA Free myös tukee vain Portable Executable- (PE), Executable and Linkable Format- (ELF) ja Mach Object- (Mach-O) tiedostomuotojen disassemblaatiota, verrattuna IDA Pron yli 45 eri tiedostomuotoon. Nämä ovat binääritiedostomuotoja, joita käytetään suoritettaville tiedostoille, objektikoodille ja jaetuille kirjastoille eri käyttöjärjestelmissä. PE on Windows-järjestelmien vakiomuoto, jota käytetään sekä 32- että 64-bittisissä sovelluksissa (Microsoft, 2024). ELF on Unix-pohjaisten järjestelmien ensisijainen suoritusmuoto, mikä mahdollistaa joustavuuden eri arkkitehtuurien välillä (Debian, 2017). Mach-O:ta käytetään macOS- ja iOS-järjestelmissä (Apple, 2014). Jokainen formaatti sisältää tietoa ohjelman koodista, datasta ja tarvittavat ohjeet ohjelman lataamiseen ja suorittamiseen, mutta niiden rakenteet vaihtelevat järjestelmävaatimusten ja suunnittelufilosofian mukaan.

2.2.2 Ghidra

Ghidra on Javalla kirjoitettu avoimen lähdekoodin ohjelma, joka julkaistiin vuonna 2019 Yhdysvaltojen Kansallisen turvallisuusviraston (NSA) toimesta. Alunperin Ghidra oli NSA:n sisäiseen käyttöön tarkoitettu ohjelmisto, mutta vuosi vuonna 2014, jonka jälkeen se julkaistiin avoimen lähdekoodin muodossa vuonna 2019. (Cimpanu, 2019.) Koska Ghidra on avoimen lähdekoodin ohjelma, voimakkaasti laajennettava ja täysin ilmainen, siitä tuli nopeasti suosittu työkalu takaisinmallintamisyhteisön sisällä.

Ghidran tärkeimpiin ominaisuuksiin kuuluu ohjelmabinäärien purkaminen, kokoaminen ja komentosarjojen käyttö takaisinmallintamisen helpottamiseksi. Ghidra tukee monia erilaisia prosessorin käskysarjoja ja suoritettavia muotoja, ja sitä voidaan käyttää sekä käyttäjän interaktiivisessa että automatisoidussa tilassa. Käyttäjät voivat myös kehittää omia Ghidra-laajennuskomponentteja ja/tai komentosarjoja käyttämällä Javaa tai Pythonia.

2.2.3 Muita takaisinmallintamiseen käytettäviä työkaluja

Ghidran sekä IDA:n lisäksi löytyy toki muita erityisen suosittuja työkaluja. Haitta-ohjelmien takaisinmallinnusta avustavat ohjelmistot sisältävät esimerkiksi Radare2:n ja Binary Ninjan.

Radare2 on suosittu takaisinmallintamiseen käytetty ohjelmisto, joka tarjoaa suuren määrän hyödyllisiä ominaisuuksia. Tämä tukee kattavan määrän eri arkkitehtuureja, lisäosia, sekä staattisen että dynaamisen analyysin työvälineet. (Pancake, 2016). Vaikka Radare2 tarjoaa kattavan dokumentaation, se ei silti välttämättä ole täysin aloittelijaystävällinen. Oletuksena Radare2 toimii täysin tekstipohjaisen käyttöliittymän kautta, joka saattaa olla aloittelevalle tutkijalle hankala käyttää. Radare2:lle on olemassa graafinen käyttöliittymä, *iaito*, jonka käyttäjäystävällisyys on parempi, mutta vaatii käyttäjältä enemmän asennus- ja harjoitteluaikaa. (Radare, 2024).

Toinen usein käytetty työkalu ohjelmistojen takaisinmallinnukseen on Binary Ninja. Tämä on vuonna 2016 julkaistu työkalu, joka myös tukee monia eri arkkitehtuureita, mutta kyseessä on täysin kaupallinen tuote. Vapaaversio on mahdollista ottaa käyttöön, mutta tämä rajoittaa ominaisuuksia huomattavasti. Arkkitehtuureita on vähemmän, laajennuksia ei voida asentaa, ja koodin automaattinen analyysi ei sisälly vapaaversioon. (Binary Ninja, 2024).

3 Tutkimusympäristö

Tässä osassa kuvataan ympäristöä, jolla tutkimus suoritettiin. Tutkimuksessa käytettiin virtuaalikoneympäristöä, johon asennettiin sekä IDA Free, että Ghidra -ohjelmat.

3.1 Virtuaaliympäristö

Haittaohjelmanäytteen turvallista tutkimista varten laajasti suositellaan virtuaalikoneen käyttöä, sillä laitteen virtualisointi tarjoaa monia etuja haittaohjelmien tutkimuksessa.

Virtuaalikonetta käyttäessä vierasjärjestelmä voi tallentaa tilansa käyttämällä tilannekuvia, jolloin tutkijat voivat vertailla tartunnan aikaisia muutoksia, kuten esimerkiksi vasta luotuja tiedostoja tai rekisteriavainmuutoksia. Tällä voidaan myös palauttaa vierasjärjestelmä aiempaan tilaan, mikäli haittaohjelma tuhoaa tai estää pääsyn järjestelmäresursseihin. On myös mahdollista, että tutkija käsittelee kiristyshaittaohjelmistoa, jonka tavoitteena on salata osa järjestelmää, jolloin näytteen vahingollinen laukaisu voisi mahdollisesti tuhota järjestelmän.

Virtuaalikone mahdollistaa myös järjestelmän verkon eristämisen, jolloin tutkittavien haittaohjelmien on mahdotonta levitä isäntäjärjestelmään tai sen kautta ulkoisiin resursseihin, kuten haittaohjelmia levittävän johto- ja valvontajärjestelmään. Näitä kutsutaan englanniksi “Command and Control” -resursseiksi tai viitataan lyhenteellä C2. Nämä C2-resurssit yleisesti tallentavat haittaohjelmien sieppaamia tietoja, kuten luottokorttitietoja, salasanoja, tai muuta

herkkäluontoista dataa. C2-järjestelmät voivat myös automatisoida haitallisia toimintoja, kuten ladata lisää haittaohjelmistoja tai antaa käskyjä komentojen suoritukseen kohdelaitteella. (Calsapeu Casagrande Adria, 2020: 36).

Virtualisoinnissa "isäntäkoneella" tarkoitetaan fyysistä tietokonetta, joka käyttää virtualisointiohjelmistoa, joka hallitsee virtuaalikoneiden luomista ja toimintaa. Tutkimuksessa käytetään Oraclen VirtualBox-ohjelmistoa. "Vieraskone" on tässä ympäristössä toimiva virtuaalikone (eng. *Virtual Machine, VM*), joka toimii itsenäisenä tietokoneena omalla käyttöjärjestelmällään ja sovelluksilla. Isäntä tarjoaa taustalla olevat resurssit, kuten suorittimen, muistin ja tallennustilan, kun taas vieras toimii ikään kuin se olisi erillinen fyysinen kone hyödyntäen näitä resursseja. Tällöin isännällä on täysi hallinta vieraskoneen resursseista.

Laboratorioympäristössä käytämme Linux-vierasjärjestelmää, johon on asennettu REMnux-järjestelmä. (Zeltser Security Corp, 2018). Tämä on Debian-based Linux -käyttöjärjestelmä, johon on asennettu laaja määrä takaisinmallintamiseen ja haittaohjelmien analysointiin tarvittavia ohjelmia. Järjestö tarjoaa sivustollansa esikonfiguroidun OVA:n (Open Virtual Appliance), jonka Oracle VirtualBox -virtuaalikoneohjelmisto avaa. Tämä vieraskone sisältää Ghidra-ohjelman, mutta ei IDA Free -ohjelmaa, joten käymme läpi sen asennuksen seuraavassa osassa.

3.1.1 IDA Freen asennus

IDA Freen asennustiedosto on hex-rays-tahon verkkosivuilla vapaasti ladattavissa, “https://hex-rays.com/ida-free/#download" -sivulla. Ladattu .run -tiedosto täytyy ensin tehdä muokattavaksi, komennolla "chmod +x idafree84_linux.run”, jonka jälkeen käynnistämme asennuksen komennolla “./ idafree84_linux.run”. Tämän jälkeen ohjelma asennetaan oletusarvoin.

3.2 *IllusionBot*-näyte

Tutkimuksessa käytetty haittaohjelma on IllusionBot, joka on ladattavissa “theZoo”-githubista, sivulta “https://github.com/ytisf/theZoo/tree/master“. Tämä repositorio sisältää monia aktiivisia haittaohjelmistoja, jotka ovat vapaasti ladattavissa. Tämän tarkoitus on tehdä aitojen haittaohjelmistojen tutkimisesta mahdollisimman helposti lähestyttävän, mutta on tärkeä tiedostaa näiden näytteiden olevan aktiivisia ja aitoja haittaohjelmia, joten näitä ei tule ajaa turvattomasti isäntäkoneella.

Näytteet ovat saatavilla zip-arkistoina, joiden purkaminen vaatii salasanan. Arkisto sisältää seuraavat tiedostot:

- BOTBINARY.EXE
- Build.exe
- WebAdmin -hakemiston.

BOTBINARY.EXE on tutkimukseen käytettävä haittaohjelmanäyte. Build.exe on ohjelma, jolla kyseisen binääritiedoston voi piilottaa eri ohjelmaan. Tätä metodia kutsutaan troijalaiseksi, jonka tarkoitus on piilottaa haittaohjelma toisen, “aidon” ohjelman sisään. Tällöin kohdekäyttäjä lataa tiedoston, jonka hän luulee olevan jokin muu, kuten tekstitiedosto, kuva, tai jonkin muun ohjelmiston asennustiedosto.

```
remnux@remnux:~/Malware Samples/IllusionBot/illusion_bot$ ls -al
total 480
drwxrwxr-x 3 remnux remnux   4096 May 31 07:14 .
drwxrwxr-x 3 remnux remnux   4096 May 31 05:37 ..
-rw-rw-r-- 1 remnux remnux  77824 Feb  5  2007 BOTBINARY.EXE
-rw-rw-r-- 1 remnux remnux 401408 May 27  2006 Build.exe
drwxrwxr-x 2 remnux remnux   4096 Feb 12  2007 WebAdmin
```

Kuva 1. *IllusionBot*-arkisto purettuna.

WebAdmin-hakemisto sisältää tarvittavat hallintapäätteet, joilla tartunnan saanutta laitetta voidaan hallita etätoimintojen avulla. Nämä näkyvät kuvassa 2, joka sisältää hallintaan ja päivittämiseen tarvittavat tiedostot.

```
remnux@remnux:~/Malware Samples/IllusionBot/illusion_bot/WebAdmin$ ls -al
total 92
drwxrwxr-x 2 remnux remnux  4096 Feb 12  2007 .
drwxrwxr-x 3 remnux remnux  4096 May 31 07:14 ..
-rw-rw-r-- 1 remnux remnux 59712 Apr 23  2006 index.php
-rw-rw-r-- 1 remnux remnux  8356 Apr 23  2006 Man.html
-rw-rw-r-- 1 remnux remnux  7846 Apr 20  2006 Readme.html
-rw-rw-r-- 1 remnux remnux  2736 Mar 26  2006 updater.php
```

Kuva 2. *WebAdmin*-hakemiston sisältö.

3.2.1 VirusTotal

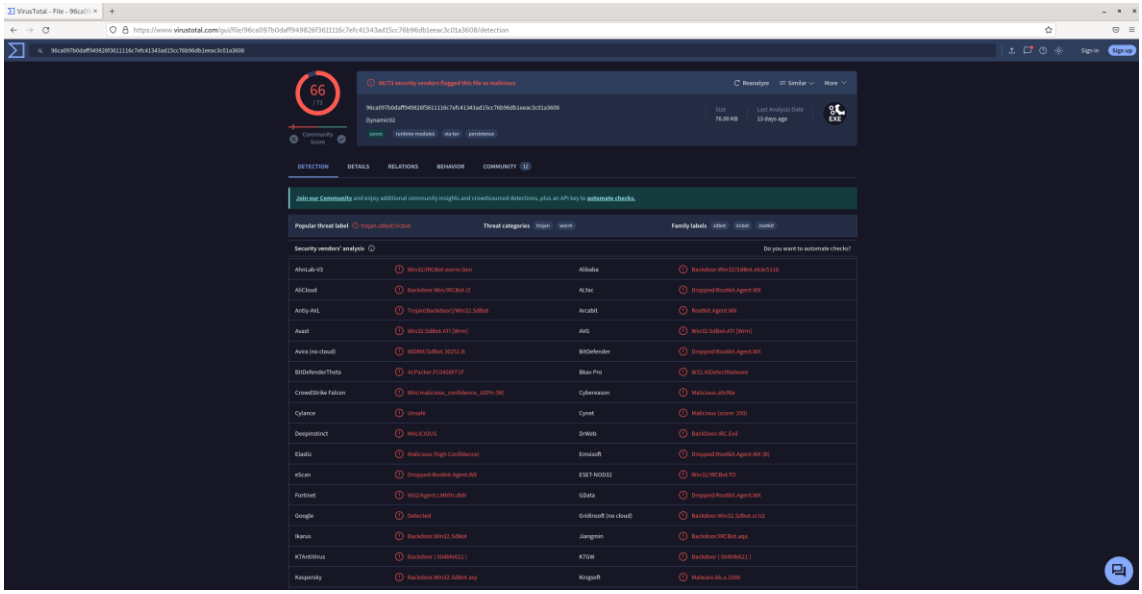
Haittaohjelma-analyysiä tehdessä on tärkeä ottaa huomioon myös, miten olemassaolevat viruksentorjuntajärjestelmät tulkitsevat kyseiset tiedostot. VirusTotal on Hispasec Sistemasin tuottama palvelu, joka ajaa ladatun tiedoston montaa eri viruksentorjuntajärjestelmää vasten. Näin voidaan nähdä, miten eri järjestelmät reagoivat ja tunnistavat haittaohjelmanäytteet. Vanhat, hyvin tunnetut näytteet ovat usein helposti tunnistettavissa, jolloin moni järjestelmä tunnistaa kyseisen näytteen haitalliseksi. Uudemmat haittaohjelmat saattavat olla täysin tuntemattomia, jolloin viruksentorjuntajärjestelmät eivät välttämättä osaa tunnistaa näitä haitallisiksi.

Hyödyllisiä resursseja on monia, kuten esimerkiksi https://any.run/-sivusto, johon haittaohjelman voi ladata. Sivusto tarjoaa palvelun, jossa näyte ajetaan palveluntarjoajan virtuaalikoneella. Tätä kutsutaan dynaamiseksi analyysiksi (eng. *Dynamic analysis*), sillä haittaohjelmaan prosessia voidaan valvoa sen ajon aikana.

Kun näyte analysoidaan VirusTotal-sivustolla, voidaan todeta, että 66/73:sta antivirusjärjestelmästä tunnistaa kyseisen näytteen haittaohjelmaksi. Voidaan myös todeta etenkin AhnLab-V3:n ja AliCloud:in tunnistavan näyte takaoveksi (eng. *Backdoor)* ja troijalaiseksi “IRCBotiksi” (eng. *IRCbot)*. Takaovilla tarkoitetaan metodeja, joilla voidaan ohittaa järjestelmän eri turvatarkastukset, jolloin hyökkääjälle annetaan kyky päästä järjestelmään sisään tai sen sisältämiin tietoihin. (Chris Wysopal & Chris Eng, 2010). Analyysissä voidaan täten ottaa huomioon, että näyte käyttää mahdollisesti *Internet-Relay Chat* (IRC) -järjestelmää kommunikoidakseen hyökkääjän kanssa.

Kuvassa 3 näkyvä analyysi sisältää tietoa itse haittaohjelmasta ja sen mahdollisista toiminnoista. Erityisesti kiinnostavat yksityiskohdat ovat tämän yksilölliset hajautusarvot, joiden avulla kyseinen näyte voidaan identifioida:

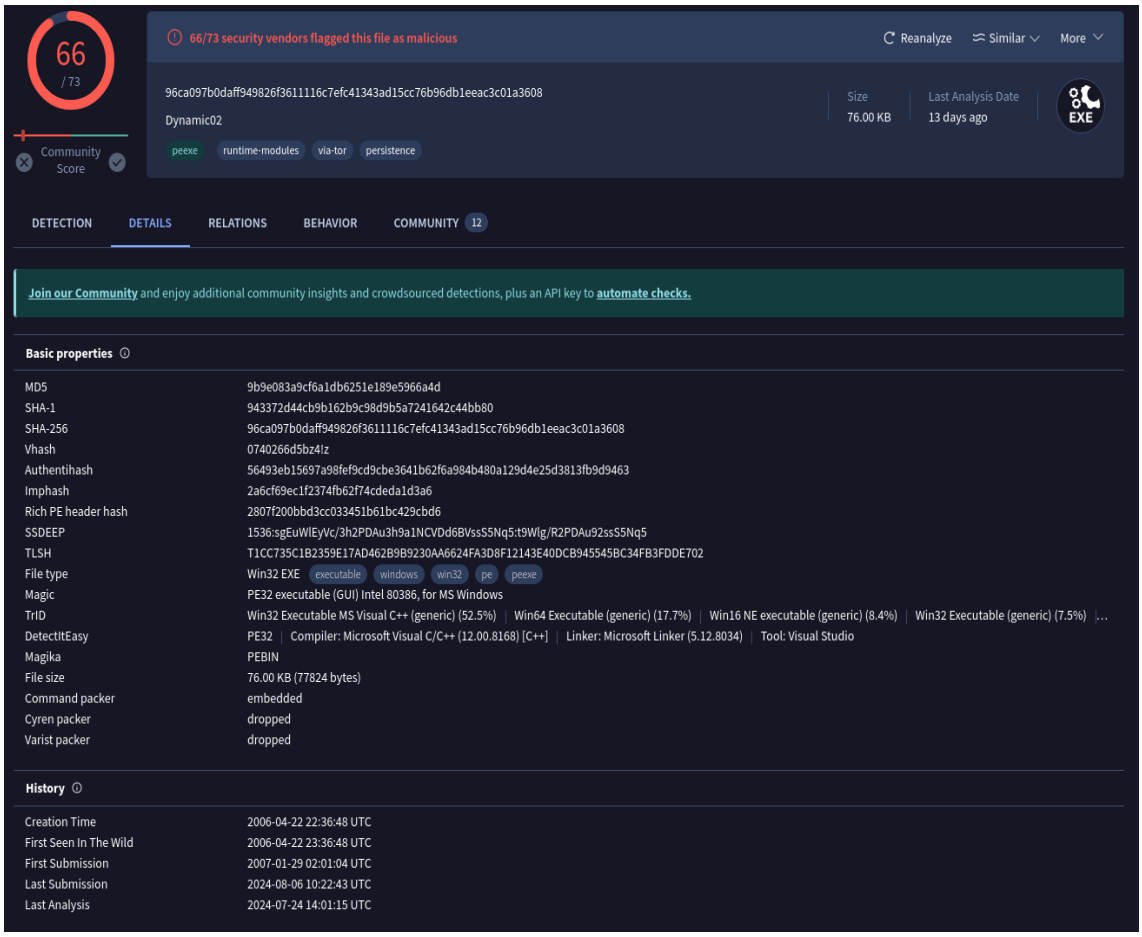
- MD5: 9b9e083a9cf6a1db6251e189e5966a4d
- SHA-1: 943372d44cb9b162b9c98d9b5a7241642c44bb80
- SHA-256: 96ca097b0daff949826f3611116c7efc41343ad15cc76b96db1eeac3c01a3608.



Kuva 3. VirusTotalin *Detection*-näkymä haittaohjelmanäytteestä.

Behavior-välilehti, joka näkyy kuvassa 4, kertoo myös näytteen oletetusta käytöksestä. Tätä tietoa voi käyttää analyysin kohdentamiseen, sillä nämä käytökset voidaan varmistaa näytteen takaisinmallintamisen avulla. Erityisen mielenkiintoisia tietopisteitä ovat seuraavat:

- Haittaohjelman luoma prosessi luo dynaamisesti haitallisia toimintoja.
- Näyte käyttää “CreateRemoteThread” -toimintaa, jota voidaan käyttää koodin syöttöön (eng. code injection).
- Näyte käyttää myös puolustuksenkiertämistekniikkaa hämärtämällä tiedostoja ja tietoa käyttämällä XOR- ja base64-koodausta.
- Näyte käyttäytyy vakoiluohjelman tavoin, tallentamalla käyttäjän näppäinpainallukset (eng. keylogger.).



Kuva 4. VirusTotalin *Details* -näkymä haittaohjelmanäytteestä.

Analyysin sisältämä haittaohjelman käyttäytymispuu (eng, *The Malware Behavior Catalog Tree*) tunnistaa laajalti samat käytökset kuin edellämainittu, mutta sisältää myös näytteen mahdollisen HTTP-kommunikaation, joka vahvistaa aiempaa epäilystä IRC-protokollien käytöstä hyökkääjän C2-verkon kanssa.

VirusTotal antaa tutkijoille paljon tärkeää tietoa, kuten tiedostojärjestelmätoiminnoista, joiden perusteella voimme tunnistaa näytteen avaavan, muokkaavan, sekä poistavan rekisteriavaimia. Nämä tapahtumat ovat oleellisia haittaohjelman pysyvyyden luomisessa, sillä näitä hyödyntämällä haittaohjelma voi luoda komentoja joilla sitä ei voida löytää tai poistaa. On myös mahdollista, että rekisterimuutokset sisältävät automatisoidun elvytystoiminnon, jonka avulla laitteen uudelleenkäynnistäminen tai käyttöjärjestelmän uudelleenasentaminenkaan ei kykene poistamaan haittaohjelmaa.

VirusTotalin yksittäiset huomiot eivät välttämättä varmista näytteen olevan haitallinen. Mikäli näyte on antivirus-ohjelmille tuntematon, tulosten arvo heikentyy. Näytteitä tutkiessa on tärkeä muistaa kokonaisuus, sillä ohjelman HTTP-kommunikaatio, tai rekisteriavainten muutos, ei ole tarpeeksi vahvistamaan tämän toimivan haitallisesti. Näitä toimintoja voidaan käyttää myös täysin harmittomista syistä. Näitä voivat olla esimerkiksi asennuksen aikana tarvittavien rekisteriavainten luonti tai verkko-ominaisuuksien toteuttaminen. Mutta kokonaisuutta tarkastellen voidaan olettaa näiden kaikkien luovan yhtenäisen kuvan epäilyttävästä ohjelmasta, jonka syvällisempi analyysi on tarpeen. Näytettä analysoitaessa tutkijoiden tulisi muistaa pitää silmällä näitä tietoliikenneprotokollia, salausmekanismeja sekä vakoilumenetelmiä.

4 Takaisinmallintamisprosessi

Tässä vaiheessa aloitetaan takaisinmallintaminen. Tutkimuksessa aloitettiin automaattisella analyysillä, jonka jälkeen pureudutaan itse haittaohjelman kyvykkyyksiin sekä vaarantumisindikaattoreihin. Jokaisessa osassa suoritetaan sama analyysi sekä Ghidralla että IDA Free -ohjelmistolla.

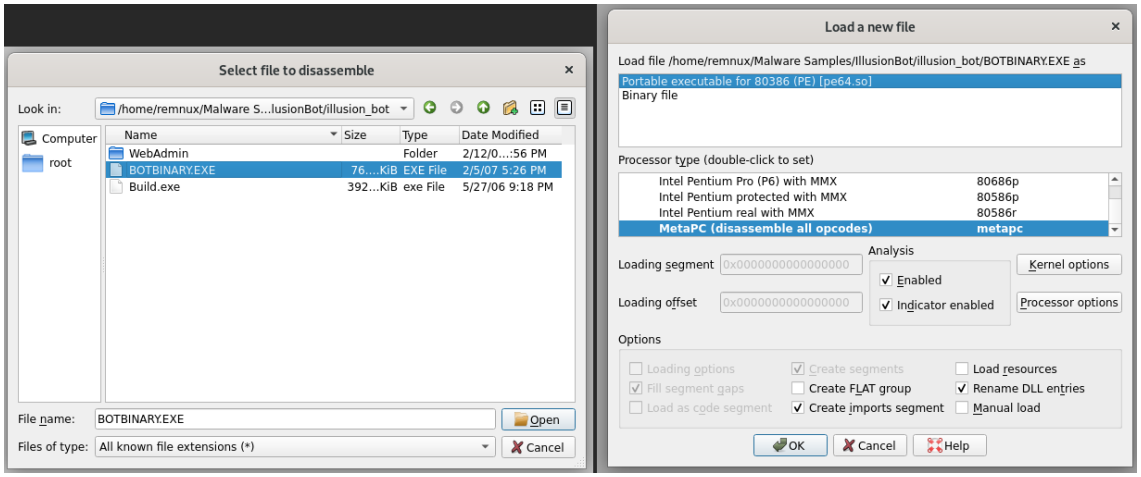
4.1 Näytteen lataus ja analyysi

Näyte ladataan ensin koodin takaisinmallintamisohjelmiin. Molemmat ohjelmat suorittavat automaattisen, alustavan analyysin, joka tuotteesta riippuen voi antaa arvokasta tietoa haittaohjelman suorituksesta, sen sisältämästä datasta, sekä mahdollisista vaarantumisindikaattoreista. Tämä analyysi on tärkeä osa takaisinmallinnusta, mutta automatisoidut prosessit eivät ole aina täysin luotettavia. Tämän takia syvällisempi analyysi on tarpeen.

4.1.1 IDA

Kuva 5 sisältää näkymät joissa haittaohjelmanäyte ladataan IDAan. Lataamalla binääritiedoston IDA-ohjelmaan tämä ehdottaa alustavasti tiedoston PE-ylätunnisteiden takia sen purkamista “Portable executable for 80386”-tyyppinä.

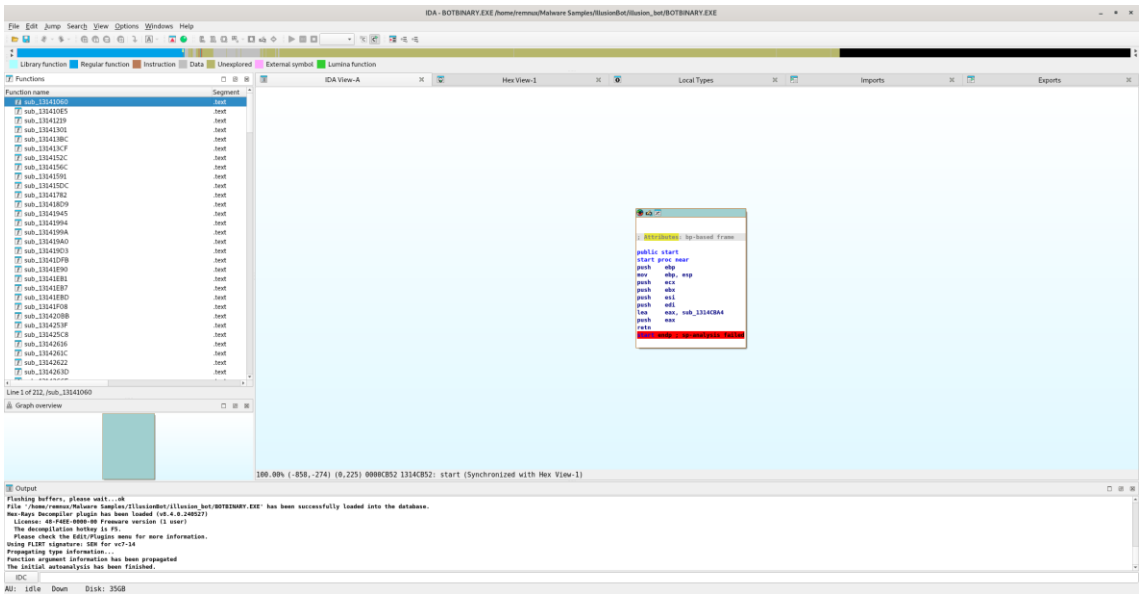
Suorittimen tyypiksi IDA on myös valinnut “MetaPC (disassemble all opcodes)”, jolla pyritään purkamaan kaikki suorittimen toimintakoodi (eng. o*peration code*).



Kuva 5. Näkymä, jossa haittaohjelmanäyte ladataan IDA Free -ohjelmaan.

Ohjelman ladattua näytteen, IDA suorittaa analyysin. Analyysin kesto riippuu kohdenäytteen suuruudesta ja monimutkaisuudesta. Tämän haittaohjelmanäytteen automaattinen analyysi kesti alle minuutin. Analyysin valmistuttua näkymä on kuvan 6 mukainen.

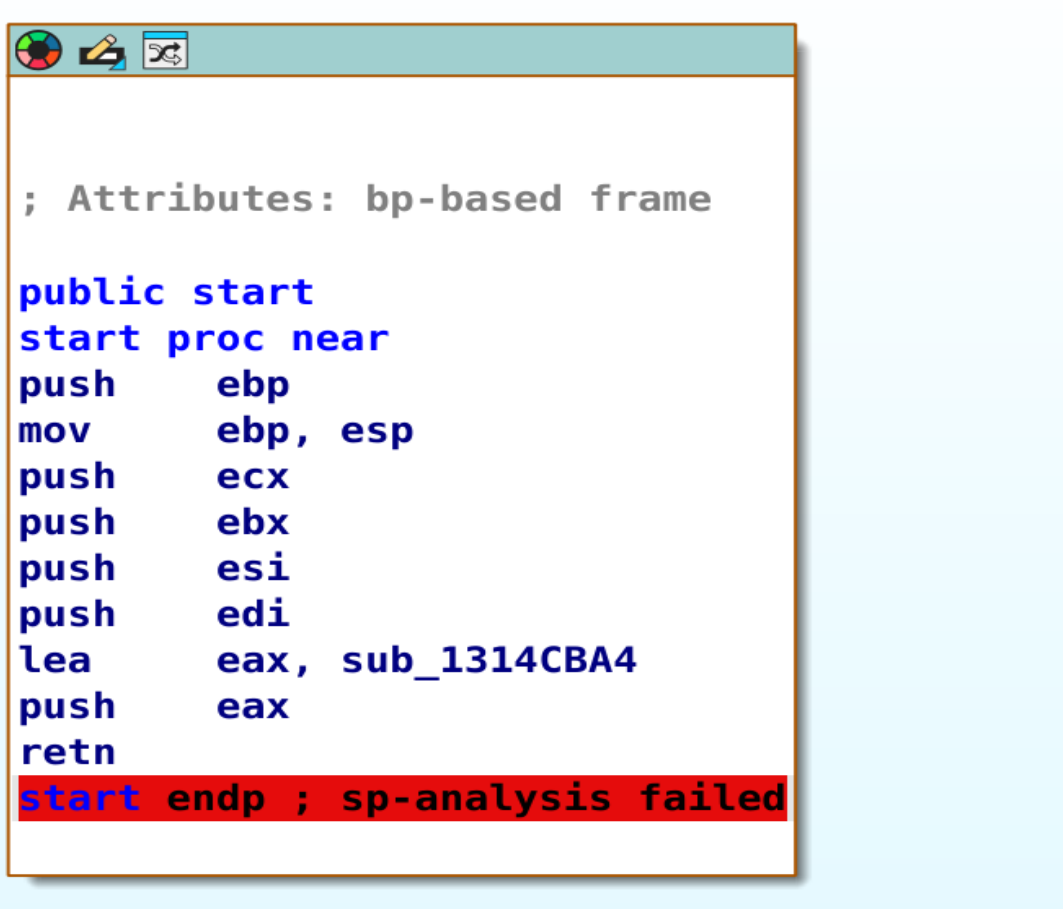
Tässä näkymässä disassemblointiohjelman keskeltä löytyy *Assembly*-kielellä ohjelman ajon aloituspiste, eli ohjelmiston lähtökohta. Näkymän vasemmalla näkyy lista aliohjelmia, jotka IDA on tunnistanut. IDA:n nimeämisjärjestelmän tuottamat nimet aliohjelmille ovat esimerkiksi *sub_1314CBA4*. Välilehdet sisältävät heksadesimaalisen näkymän kyseisen aliohjelman suoritusohjeista, tämän paikalliset tyypit (*Local types*) sekä ohjelman tuomat ja viemät kirjastot (*Imports, Exports*).



Kuva 6. IDA Freen näkymä automaattisen analyysin valmistuttua.

IDA myös luo kommentteja analyysinsä perusteella. Nämä näkyvät esimerkiksi kuvassa 6. Kommentit on merkitty puolipisteellä eivätkä sisälly alkuperäiseen binääriin. Aliohjelma asettaa pinoalueen (*stack frame*) ja tallentaa useita rekistereitä (ecx, ebx, esi ja edi) säilyttääkseen niiden arvot. Tämän jälkeen ladataan toisen aliohjelman, *sub_1314CBA4*, osoite eax-rekisteriin, joka sitten työnnetään pinoon. Palautusohje ”retn” viittaa siihen, että tämä ohjelma valmistautuu kutsumaan toista aliohjelmaa tai on osa kutsuketjua. Aliohjelman osoitteen työntäminen pinoon viittaa siihen, että nämä ovat valmistelevia vaiheita epäsuoraa aliohjelmakutsua varten.

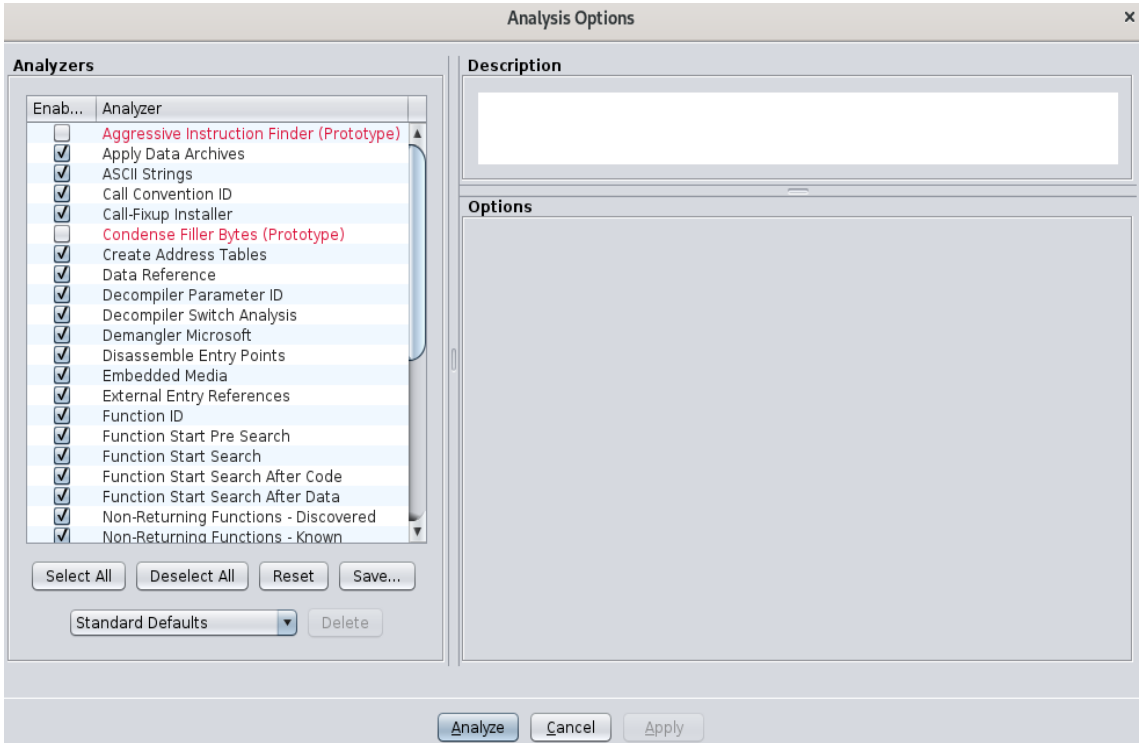
Kuten kuvasta 7 näkyy, IDA toteaa analyysin menneen pieleen. Viesti ”*sp-analysis failed”* tarkoittaa pino-osoittimen analyysin epäonnistuneen, mutta koodin mukaisesti näemme sen silti asettaneen edellämainitun aliohjelman kutsua varten. Voidaan siis olettaa tämän ohjelman kutsuvan aliohjelmaa *sub_1314CBA4*, joka toimii ohjelman varsinaisena aloituspisteenä.



Kuva 7. Tarkempi kuva Assembly-ohjeista.

4.1.2 Ghidra

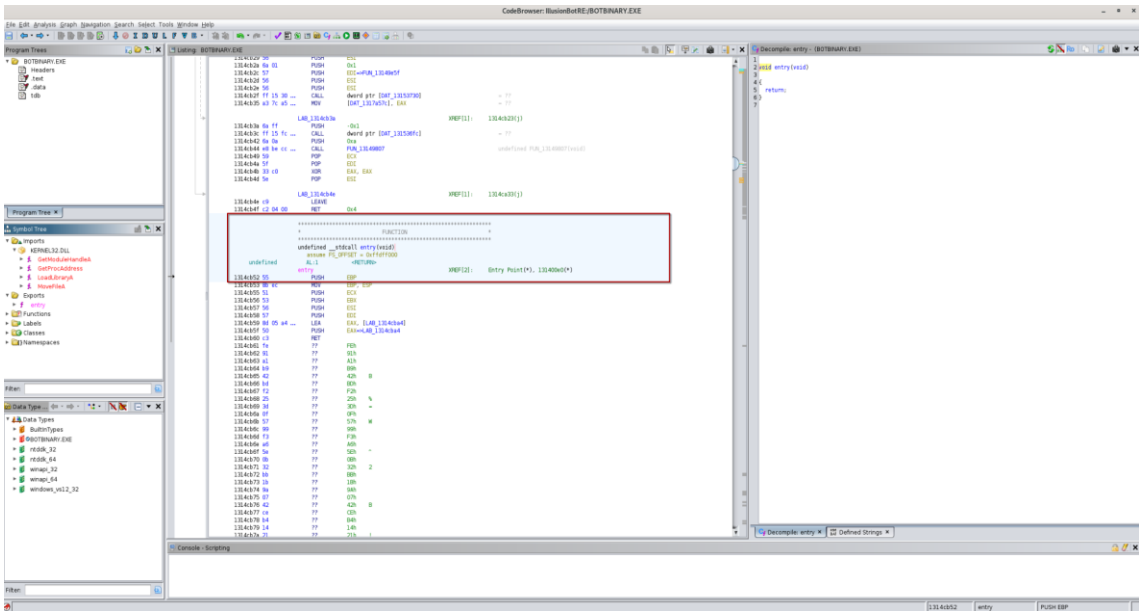
Haittaohjelmanäyte ladataan Ghidraan samalla menetelmällä kuin IDAan. Ghidra lukee näytteen ja ehdottaa sen analysoimista automaattisesti. Tämä kehoite näkyy kuvassa 8, jossa nähdään Ghidran oletuskehotteiden sisältävän esimerkiksi merkkijonojen, aliohjelmien ja osoitetaulun tunnistamiseen helpottavia analysaattoreita.



Kuva 8. Ghidran oletusanalysaattoriehdotukset.

Tutkimuksessa jätettiin näihin oletusarvot, jolloin analyysi kesti viisi minuuttia. Tämä oli huomattavasti pidempi aika IDAan verrattuna. Ghidran tuottama näkymä näkyy kuvassa 9. Näkymä sisältää haittaohjelmabinäärin disassembloituna näkymän keskellä. Vasemmalla näkyvät symboli- ja ohjelmapuut sekä *Imports*-osio, johon palaamme osassa 4.3.

Näkymän oikealla sijaitsee *decompiled*-versio Assembly-kielestä. Ghidra on kääntänyt Assembly-kielestä valitun osion C-ohjelmointikieleen. Kuten kuvasta ilmenee, Ghidra ei ole osannut analysoida kyseistä aliohjelmaa oikein. Kuvan 9 takaisinkäännetty koodi perustuu vain osaan keskellä näkyvää Assembly-ohjelmaa, jonka takia Ghidra tuottaa vajaan ohjelmanpätkän.



Kuva 9. Ghidran oletusnäkymä automaattisen analyysin jälkeen.

4.1.3 Vertailu

Molemmat ohjelmat tekevät näytteen latauksesta vaivatonta ja tarjotut näkymät tarjoavat paljon tietoa alustavan analyysin suorituttua. Konekoodin kääntäminen Assembly-ohjelmakoodiksi on yhdenmukaista Ghidran ja IDA Freen välillä.

Kuten taulukossa 1 nähdään, IDA Freen lisäämät kommentit ovat lisäetu etenkin tämän analyysin aikana, sillä tämä auttaa korostamaan ohjelman varsinaisen aloituskohdan lisäystä. Tämä korostus helpottaa Assembly-koodin tulkitsemista huomattavasti.

Ghidran purkama näkymä on yleisesti hyödyllinen, sisältäen *decompiled*-näkymän. Tämä auttaa tutkijoita joilla on vähemmän kokemusta Assembly-ohjelmointikielen tulkinnasta. Mutta tässä tapauksessa Ghidran näkymä on harhaanjohtava, koska se ei onnistunut havaitsemaan aliohjelman latausta pinoon, mikä toimii haittaohjelman todellisena sisääntulopisteenä. Lisäksi korostus Assembly-näkymässä jättää pois sitä seuraavat ohjeet, jotka tunnistaisivat tämän mahdollisesti haitallisen lisäyksen.

Ohjelmistojen välinen automaattisen analyysin kesto oli myös huomattava, sillä Ghidran analyysi vei noin neljä minuuttia pidempään. Vaikka tällä ei todennäköisesti ole suurta vaikutusta itse haittaohjelmaa takaisinmallinnuksessa, on silti huomioitava, että kyseinen haittaohjelmanäyte oli suhteellisen pieni. Isompien haittaohjelmanäytteiden kohdalla, näiden kahden ohjelman välisen aikaeron voidaan olettavan kasvaa. Navigointi oli myös huomattavasti nopeampaa ja reagoivampaa IDA:ssa Ghidraan verrattuna.

Taulukko 1. Eri indikaattorien vertailu automatisoidun alkuanalyysin aikana.

Ilmaisin	**Ghidra**	**IDA Free**	**Lisäkommentit**
Käyttöliittymätuki	Keskitasoinen	Korkea	Ghidra oli huomattavasti hitaampi alustavan analyysin suorittamisessa.
Kontekstuaalisen lisätiedon laatu	Vähäinen	Korkea	*Decompiled*-tuki Ghidrassa on eduksi, mutta tieto on vajanainen. IDA Freen lisäämät kommentit auttavat paljon.
Tutkijalta vaadittava manuaalinen työ	Keskitasoinen	Keskitasoinen	Molemmissa vaaditaan Assembly-osaamista haitallisen koodin tulkitsemiseen.

4.2 Defined Strings

Tämä osio sisältää defined strings -analyysin, joka kertoo tutkijalle, mitä merkkijonoja ohjelma sisältää. Näitä useasti käytetään toiminnallisuusanalyysiin ja vaarantumisindikaattorien löytämiseen. Ohjelman sisältämät merkkijonot voivat kertoa seuraavaa:

- Mihin IP-osoitteisiin ohjelma ottaa yhteyttä?

- Mitä rekistereitä luodaan tai muokataan?
- Mitä protokollia haittaohjelma käyttää hyväkseen?

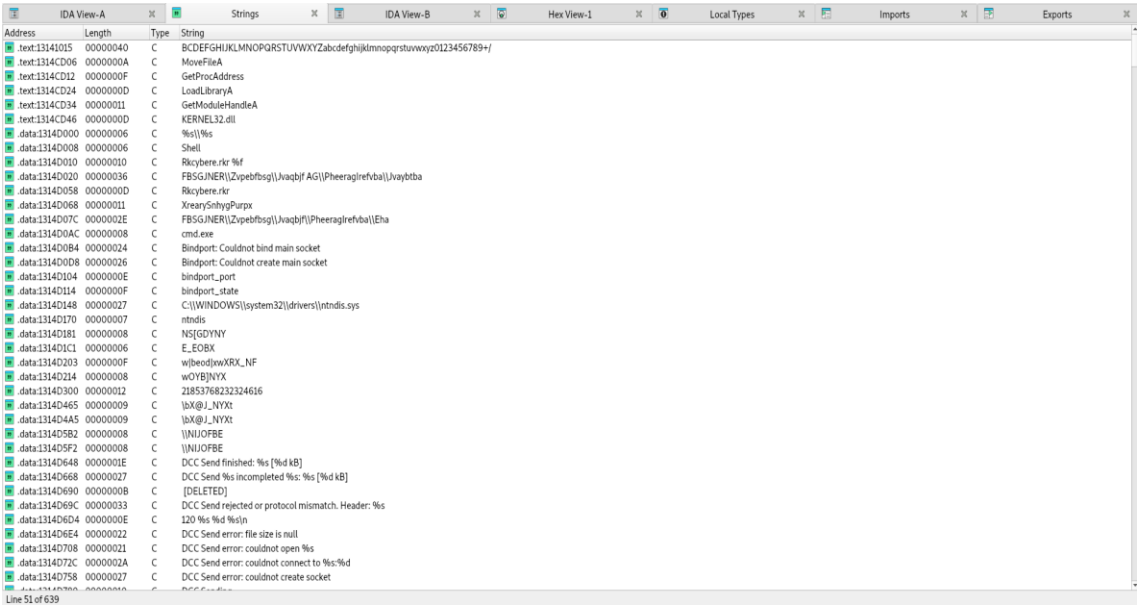
4.2.1 IDA Free

IDA ei oletuksena näytä *Strings*-välilehteä, vaan se pitää tuoda erikseen näkyväksi. Tämä voidaan tehdä ylärivivalikon kautta. *View* > *Open subviews* > *Strings*, tai pikavalintanäppäinyhdistelmän *Shift+F12* avulla.

Kun näkymä on luotu, IDA esittää listan kaikista merkkijonoista, jota ohjelma sisältää. Näitä on IDA:n mukaan 639. Näkymä sisältää monia mielenkiintoisia kohtia, kuten maininnan "cmd.exe”, jonka voidaan olettaa luovan komentorivikehotteen, jonka avulla haittaohjelma voi ajaa kohdekoneella haittaohjelmaan ohjelmoituja komentoja. On myös mahdollista, että tämä antaa hyökkääjälle mahdollisuuden ajaa myös kohdennettuja käskyjä. Maininnat bindporteista ovat myös varma osoitus tämän ohjelmiston sisältämistä verkko-ominaisuuksista, sillä nämä sisältävät tietoa, joka määrittää, missä ja miten viestejä lähetetään tai vastaannotetaan.

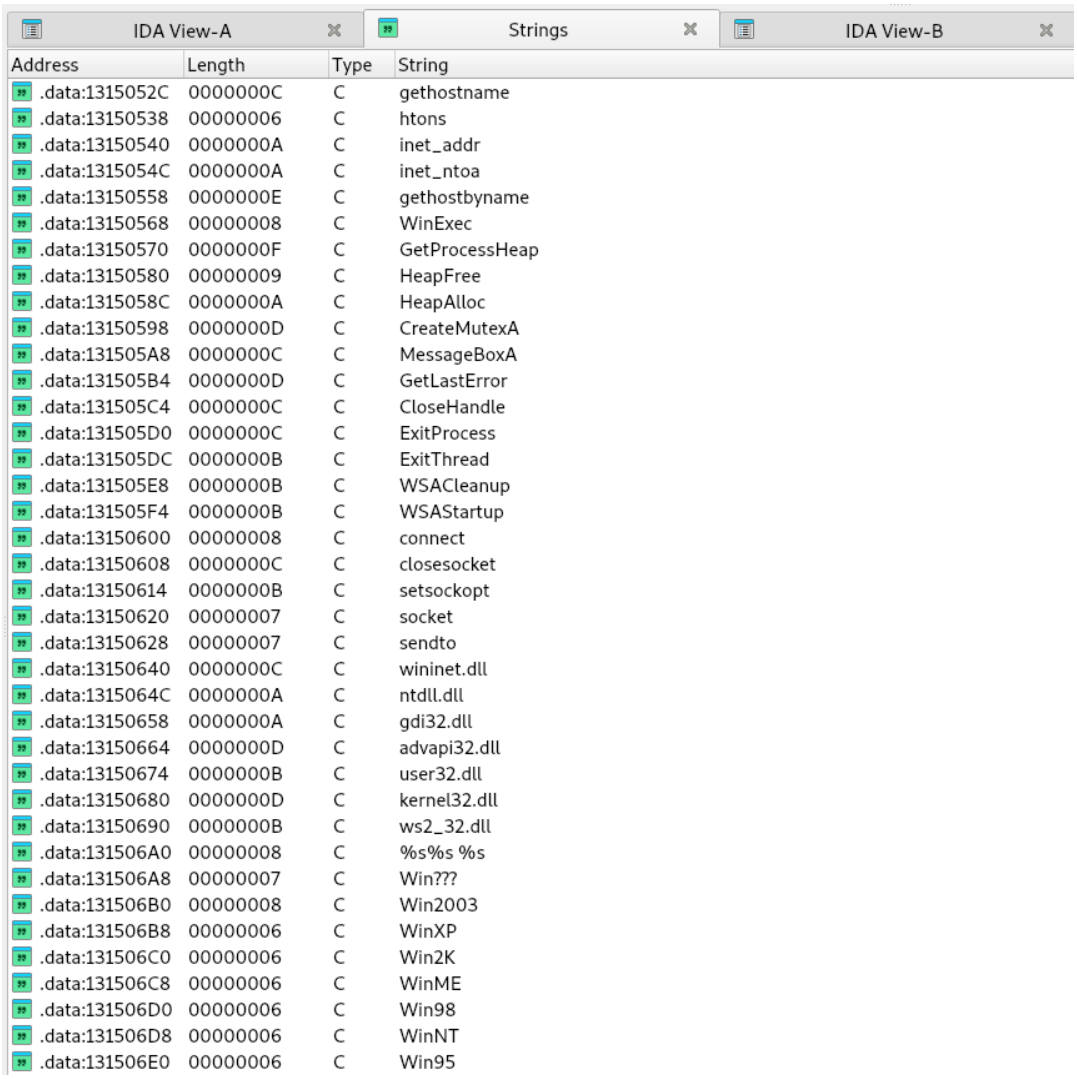
Näkymä myös sisältää monia merkkijonoja, jotka on selvästi salattu. Nämä ovat esimerkkejä koodin obfuskaatiosta, jolla pyritään vaikeuttamaan ohjelmakoodin analysointia. Seuraavat merkkijonot ovat esimerkkejä löydetyistä salatuista merkkijonoista, joita IDA tunnisti:

- FBSGJNER\\Zvpebfbsg\\Jvaqbjf AG\\Pheeraglrefvba\\Jvaybtba
- \\NIJOFBE
- wOYB]NYX



Kuva 10. IDA:n *Strings*-välilehti.

Strings-välilehti myös sisältää viittauksia muihin aliohjelmiin, jotka eivät sisälly kirjastoihin, jotka ohjelma on ilmoittanut tuoneensa. Näistä on esimerkkejä kuvassa 11. Kuvassa myös näkyy monia kirjastoja sekä niiden sisältämiä aliohjelmia. Useita näistä käytetään usein haittaohjelmissa, kuten *CreateMutexA*- sekä *HeapFree*- ja *HeapAlloc* -aliohjelmat, joita käytetään muistinhallintaan. (Microsoft, 2023; Microsoft, 2024.)

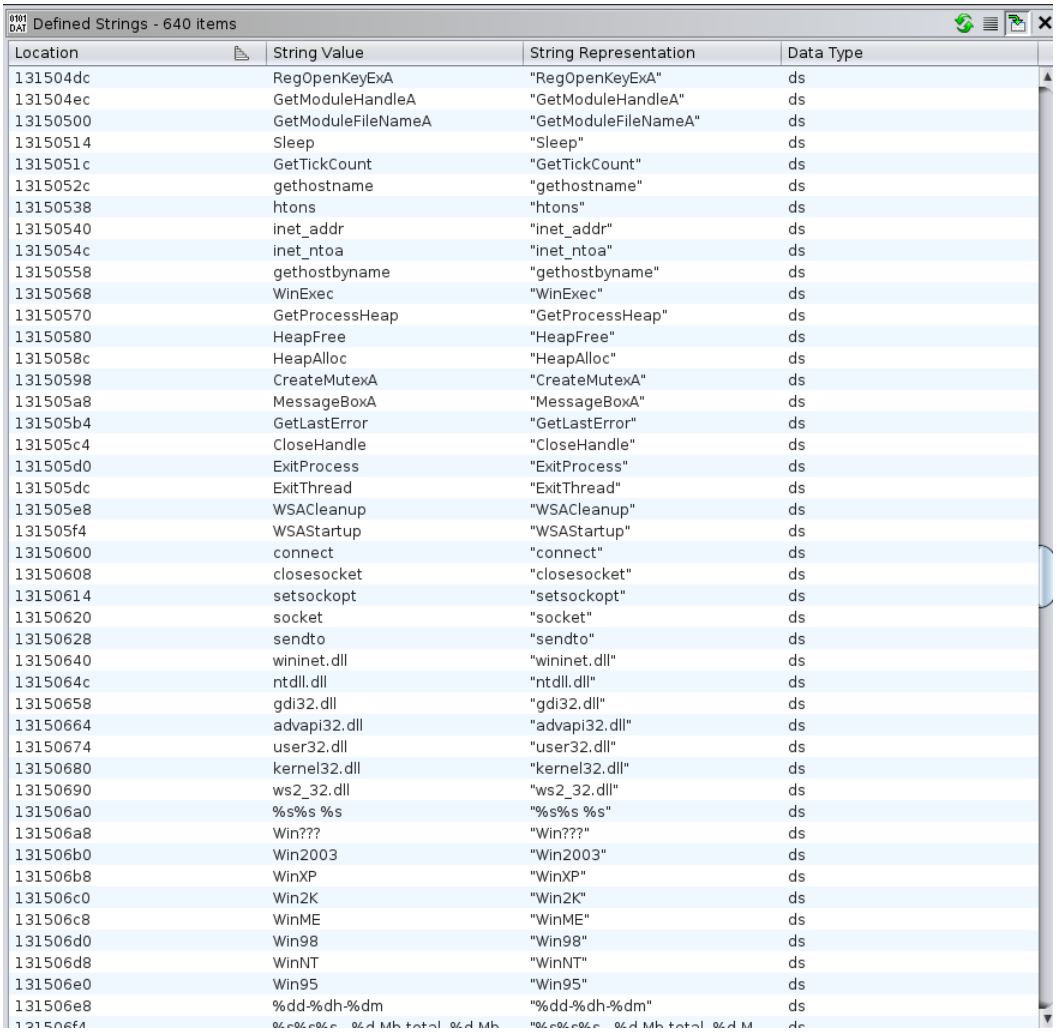


Kuva 11. IDA:n *Strings*-välilehti, joka sisältää monia aliohjelmia sekä viittauksia muihin kirjastoihin.

4.2.2 Ghidra

Ghidran oletusnäkymä sisältää *Defined Strings* -välilehden, joka tunnistaa samat merkkijonot kuin IDA. Suurin ero näkyy itse käyttöliittymän saavutettavuudessa, sillä Ghidran tuottama listaus korostaa joka toisen rivin taustan eri värillä, joka helpottaa lukemista. Se myös tunnistaa sekä merkkijonon sijainnin muistissa (*Location*), sen arvon (*String Value*) ja merkkijonoesityksen ihmisluettavana (*String Representation*) sekä tietotyypin (*Data Type*). Tässä tapauksessa suurin osa näistä tyypeistä on "ds”, joka tarkoittaa “defined strings”. Ghidra käyttää tätä, kun se ei tunnista tiettyä tietotyyppiä, kuten esimerkiksi

“unicode”, “uchar[]” tai “wchar16[].” Tämä ylimääräinen tieto on erityisen hyödyllistä syvällisemmän tutkimuksen kannalta.



Defined Strings - 640 items

Location	String Value	String Representation	Data Type
131504dc	RegOpenKeyExA	"RegOpenKeyExA"	ds
131504ec	GetModuleHandleA	"GetModuleHandleA"	ds
13150500	GetModuleFileNameA	"GetModuleFileNameA"	ds
13150514	Sleep	"Sleep"	ds
1315051c	GetTickCount	"GetTickCount"	ds
1315052c	gethostname	"gethostname"	ds
13150538	htons	"htons"	ds
13150540	inet_addr	"inet_addr"	ds
1315054c	inet_ntoa	"inet_ntoa"	ds
13150558	gethostbyname	"gethostbyname"	ds
13150568	WinExec	"WinExec"	ds
13150570	GetProcessHeap	"GetProcessHeap"	ds
13150580	HeapFree	"HeapFree"	ds
1315058c	HeapAlloc	"HeapAlloc"	ds
13150598	CreateMutexA	"CreateMutexA"	ds
131505a8	MessageBoxA	"MessageBoxA"	ds
131505b4	GetLastError	"GetLastError"	ds
131505c4	CloseHandle	"CloseHandle"	ds
131505d0	ExitProcess	"ExitProcess"	ds
131505dc	ExitThread	"ExitThread"	ds
131505e8	WSACleanup	"WSACleanup"	ds
131505f4	WSAStartup	"WSAStartup"	ds
13150600	connect	"connect"	ds
13150608	closesocket	"closesocket"	ds
13150614	setsockopt	"setsockopt"	ds
13150620	socket	"socket"	ds
13150628	sendto	"sendto"	ds
13150640	wininet.dll	"wininet.dll"	ds
1315064c	ntdll.dll	"ntdll.dll"	ds
13150658	gdi32.dll	"gdi32.dll"	ds
13150664	advapi32.dll	"advapi32.dll"	ds
13150674	user32.dll	"user32.dll"	ds
13150680	kernel32.dll	"kernel32.dll"	ds
13150690	ws2_32.dll	"ws2_32.dll"	ds
131506a0	%s%s %s	"%s%s %s"	ds
131506a8	Win???	"Win???"	ds
131506b0	Win2003	"Win2003"	ds
131506b8	WinXP	"WinXP"	ds
131506c0	Win2K	"Win2K"	ds
131506c8	WinME	"WinME"	ds
131506d0	Win98	"Win98"	ds
131506d8	WinNT	"WinNT"	ds
131506e0	Win95	"Win95"	ds
131506e8	%dd-%dh-%dm	"%dd-%dh-%dm"	ds

Kuva 12. Ghidran *Defined Strings* -välilehti, joka sisältää saman informaation kuin IDA.

4.2.3 Vertailu

Molemmat ohjelmat tukevat vahvasti merkkijonojen löytämistä ja analysointia. Kuten taulukko 2 osoittaa. Vaikka IDA Free ei sisällä oletusarvoisesti *strings*-näkymää, sen lisääminen käyttöliittymään on erittäin helppoa.

Ghidran sisältämä kontekstuaalinen lisätieto on suurempi kuin IDA Freen, mutta tässä tapauksessa tällä ei ole kovin suurta vaikutusta, sillä kaikki löydetyt

merkkijonot ovat *data strings* -tyyppiä, joka ei anna kovin paljon ylimääräistä tietoa ohjelman haitallisuudesta tai ajosta. Käytettyjen muuttujatyyppien ymmärtäminen voi olla hyödyllistä, mutta pääpaino käytetyn merkkijonon haitallisuuden määrittämisessä riippuu suuresti kontekstista jossa sitä käytetään, sekä tietenkin itse merkkijonosta.

Taulukossa myös nähdään molempien ohjelmien vaativan jonkin verran manuaalista työtä merkkijonojen tulkitsemiseen. Molemmat ohjelmat tukevat tätä työtä hyvin, sillä käyttöliittymät mahdollistavat helpon navigoinnin niihin aliohjelmiin joissa kyseisiä merkkijonoja käytetään.

Taulukko 2. Eri indikaattorien vertailu automatisoidun merkkijonolöydöksissä.

Ilmaisin	**Ghidra**	**IDA Free**	**Lisäkommentit**
Käyttöliittymätuki	Korkea	Korkea	Molemmat ohjelmistot löysivät haitalliset merkkijonot. Molemmat ohjelmat tukevat merkkijonojen tulkitsemista erityisen hyvin.
Kontekstuaalisen lisätiedon laatu	Korkea	Keskitasoinen	Ghidra sisältää hieman enemmän tietoa muuttujista. Ristiinviittaus on helppoa molemmissa ohjelmistoissa.
Tutkijalta vaadittava manuaalinen työ	Korkea	Korkea	Merkkijonojen tulkitseminen vaatii manuaalista työtä.

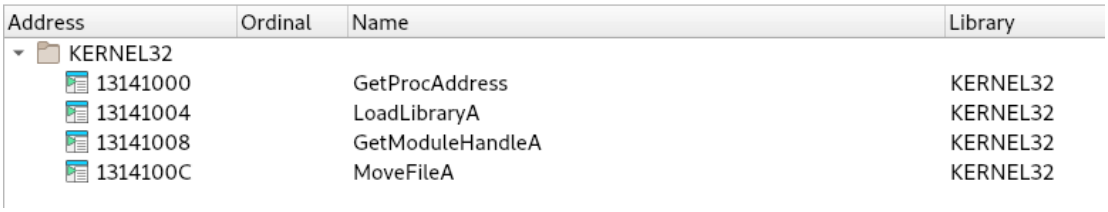
4.3 Imports

Imports-osio sisältää kirjastot, joita ohjelma tuo sen ajon alkaessa. Kirjastot ovat kokoelma aliohjelmia, luokkia tai kokonaisia ohjelmia, joita yleisesti käytetään ohjelmoinnin aikana silloin kun kyseessä on ominaisuus, jota ohjelmoija ei itse halua tai tarvitse kirjoittaa itse, kuten esimerkiksi tiedostojen käsittelyn tai käyttöliittymien luomisen. Windows-ympäristössä nämä ovat usein Dynamic-link library (DLL), joiden etuna on keskusmuistin ja levytilan säästö. Säästöt johtuvat siitä, että samaa ohjelmakoodia ei tarvitse linkittää jokaiseen ohjelmaan erikseen, vaan ne ladataan ajonaikana yhteisestä kirjastosta.

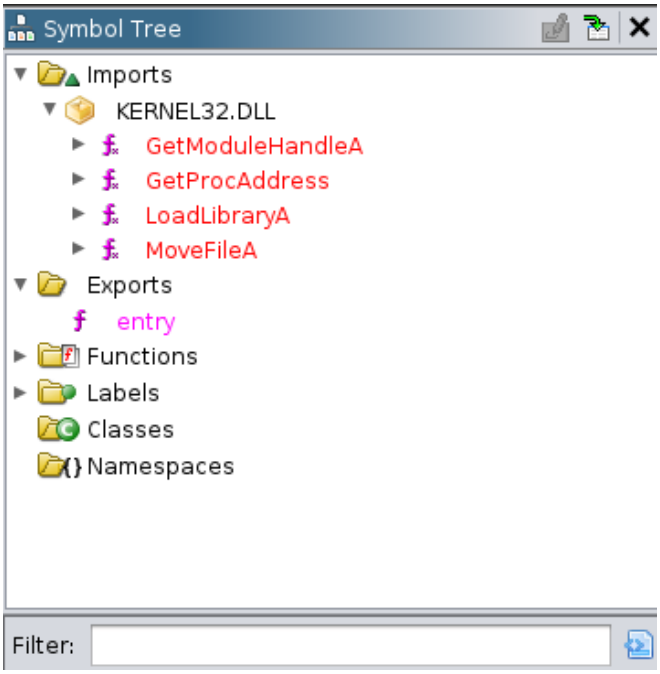
Haittaohjelmat myös käyttävät kirjastoja, joten niitä tarkastelemalla voidaan tunnistaa, mitä ominaisuuksia nämä käyttävät. Tähän myös liittyy erilaisia obfuskaatiomenetelmiä, jolla haittaohjelmien tekijät pyrkivät vaikeuttamaan kirjastojen tulkintaa disassembleri-työkalujen kautta. Yksi menetelmä, joka *IllusionBot*-haittaohjelma käyttää, on kirjastojen tuonnin piilottaminen.

4.3.1 *Imports*-välilehti

IDA ja Ghidra molemmat sisältävät *Imports*-välilehden, joka näkyy heti ohjelman analysoimisen jälkeisessä näkymässä, kuvissa 13 ja 14. IDA ja Ghidra molemmat osoittavat ohjelmanäytteen tuovan käyttöönsä vain yhden kirjaston, Kernel32, sekä neljä sen sisältämää aliohjelmaa. Nämä aliohjelmat ovat toiminnallisesti epäilyttäviä, sillä niitä vaaditaan prosessien hallintaan sekä tiedostojen luomiseen. *LoadLibraryA*-aliohjelmaa myös käytetään nimenomaan kirjastojen tai suoritettavien tiedostojen lataamiseen ohjelman ajon aikana. (Microsoft, 2023). Tämä on yleinen haittaohjelmointimenetelmä, jolla obfuskoidaan toiminnallisuuksia piilottamalla niiden tuonti ajon alkaessa. Näiden tuontitietojen perusteella, voidaan olettaa näytteen tuovan käyttöönsä ulkoisia kirjastoja, joita *Imports*-välilehdellä ei näy.



Kuva 13. IDA:n *Imports*-välilehti.



Kuva 14. Ghidran *Symbol Tree* -näkymä, joka sisältää tiedon tuoduista kirjastoista.

4.3.2 Vertailu

Sekä Ghidra että IDA Free pystyivät tunnistamaan tuodut kirjastot ja auttoivat tutkijaa seuraamaan niiden kutsuja haittaohjelmanäytteen sisällä. Kuten taulukosta 3 voidaan nähdä, molemmat ohjelmat tarjosivat yhtä paljon apua käyttöliittymässä, tarjosivat hyödyllistä kontekstia ja tekivät haittaohjelmanäytteen sisältämän koodin navigoinnista helppoa.

Kirjastojen sisällön tulkitseminen on usein työläämpi prosessi, koska tämä vaatii syvällisempää taustatutkimuksesta niiden sisällöstä. Vaikka tässä tapauksessa ohjelmanäytteen tuomia kirjastoja käytetään usein haittaohjelmistoissa, on tärkeä ottaa huomioon konteksti joissa kirjastoja ja sen tuomia aliohjelmia käytetään sillä ne tietenkään aina ole haitallisia. On myös epätodennäköistä, että

haittaohjelma hyödyntää jokaista tietyn kirjaston toimintoa, joten seuraavassa osassa käsitellään tarkemmin varsinaista toiminnallisuusanalyysiä.

Taulukko 3. Eri indikaattorien vertailu tuotujen kirjastojen analysoinnissa.

Ilmaisin	**Ghidra**	**IDA Free**	**Lisäkommentit**
Käyttöliittymätuki	Korkea	Korkea	Molemmat ohjelmistot löysivät haitalliset DLL tiedostot.
Kontekstuaalisen lisätiedon laatu	Keskitasoinen	Keskitasoinen	Ristiinviittaus onnistuu helposti.
Tutkijalta vaadittava manuaalinen työ	Korkea	Korkea	Kirjastojen tulkitseminen voi vaatia paljon aikaa, riippuen niiden sisällöstä.

4.4 Toiminnallisuusanalyysi

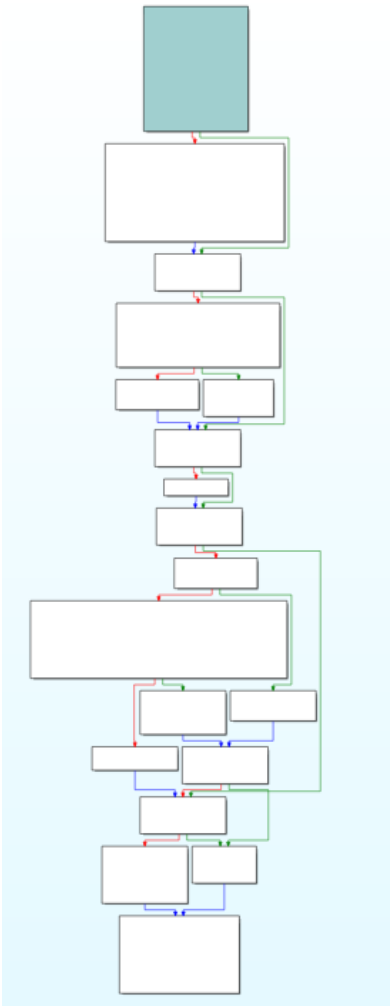
Tämä osio sisältää haittaohjelman toiminnallisuusanalyysiä, jossa pyritään ymmärtämään, mitä toiminnallisuuksia haittaohjelma sisältää.

Edellisten osioiden perusteella on selvää, että haittaohjelmalla on todennäköisesti useita eri toiminnallisuuksia Näitä ovat eri verkko- ja verkkoindeksointitoiminnot, sekä kyky lähettää komentoja komentorivin kautta. Lisäksi näytteessä on viitteitä Windows-rekisteriavaimista, mikä voi viitata pysyvyysmekanismien olemassaoloon. On kuitenkin tärkeää todeta, että nämä johtopäätökset on saatu merkkijonojen analysoinnista, joten näihin tulee suhtautua varovaisesti. Haittaohjelman toiminnan ymmärtämiseksi on tärkeää arvioida sen todellinen toiminnallisuus sen sijaan, että luotetaan pelkästään automatisoitujen työkalujen antamiin havaintoihin.

4.4.1 Lisäominaisuuksien tuonti

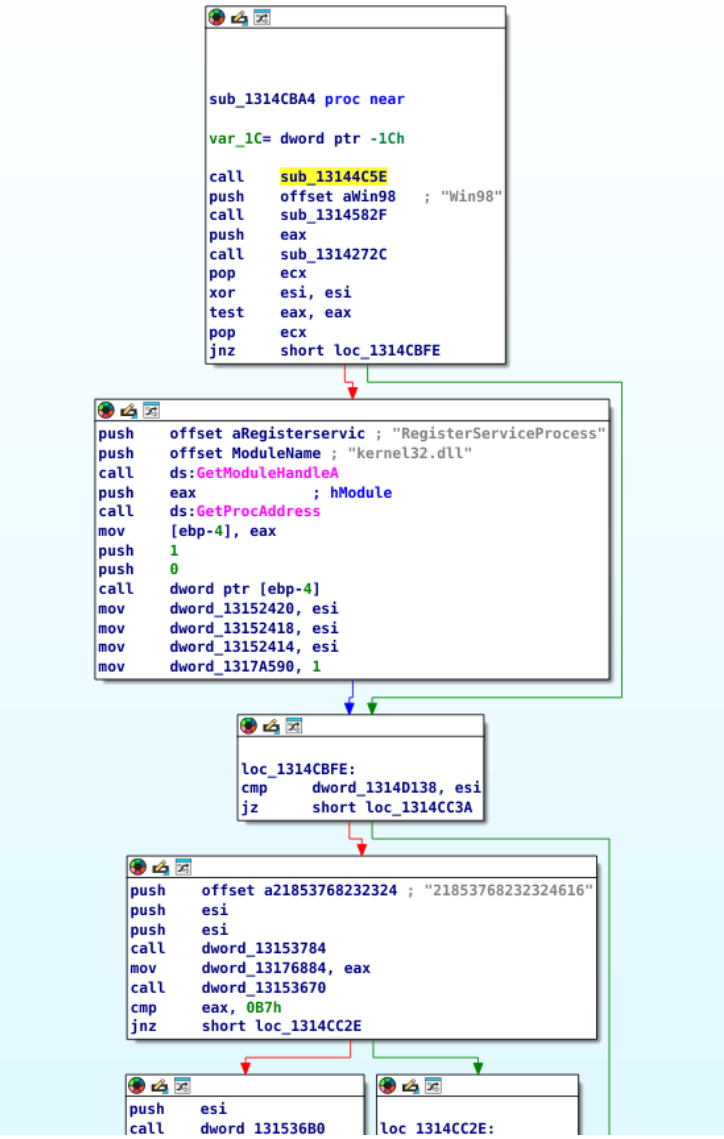
IDA antaa tutkijoiden siirtyä eri aliohjelmiin kaksoisnapsauttamalla halutun aliohjelman nimeä käyttöliittymässä. Kuvassa 7 näkyvä aliohjelma *sub_1314CBA4* johtaa kuvan 15 mukaiseen kuvioon.

Tämä on aliohjelman sisäinen logiikka, joka kuvaa tähän ohjelmoituja päätöksiä. Jokainen haarautumiskohta esittää kohtaa, jossa aliohjelma tekee päätöksen siihen ohjelmoidun logiikan perusteella. Punaiset nuolet esittävät negatiivista päätöstä ja vihreät positiivista. Nämä voivat usein olla aliohjelman sisäinen *if-else* lause tai jokin muu päätösmekanismi. Siniset nuolet edustavat ehdotonta siirtymistä seuraavaan askeleeseen.



Kuva 15. IDA:ssa loitonnettu kuva *sub_1314CBA4*-aliohjelman logiikkakaaviosta.

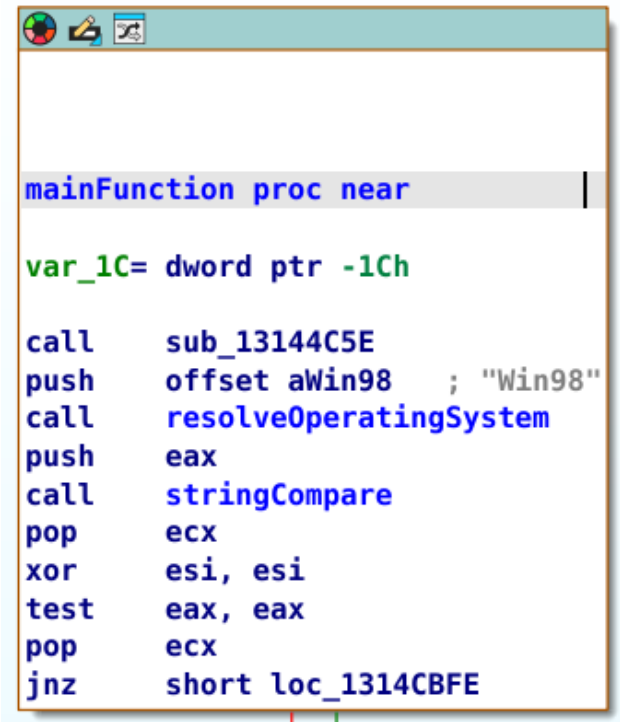
Näkymää suurentamalla aliohjelmaa pystytään tarkastelemaan selvemmin. Kuva 16 näyttää aliohjelman Assembly-ohjelmakoodia. Tässä näkyy ohjelman kutsuvan ensin toista aliohjelmaa, *sub_13144C5E*, jonka jälkeen se kutsuu aliohjelmaa *sub_1314582F*. Seuraamalla näitä aliohjelmia voimme päätellä näiden aliohjelmien selvittävän kohdekoneen käyttöjärjestelmän, johon aiemmin mainitut päätökset perustuvat.



Kuva 16. Suurennettu kuva *sub_1314CBA4*-aliohjelman logiikkakaaviosta, joka sisältää ohjelmakoodin.

IDA tarjoaa myös uudelleennimeämistoiminnan, jota voidaan käyttää analyysin helpottamiseen. Tämän avulla voidaan selventää analyysin aikana selvitettyjen aliohjelmien nimiä, joten aliohjelma *sub_1314CBA4, sub_1314582F* ja

sub_1314272C muuttuvat esimerkiksi *main*-, *resolveOperatingSystem*- ja *stringCompare*-nimisiksi. Tämä voidaan toistaa tarkastelemalla aliohjelman käyttäytymistä, jonka jälkeen ohjelman toiminnalisuuden ymmärtäminen nope-alla vilkaisulla on helpompaa.



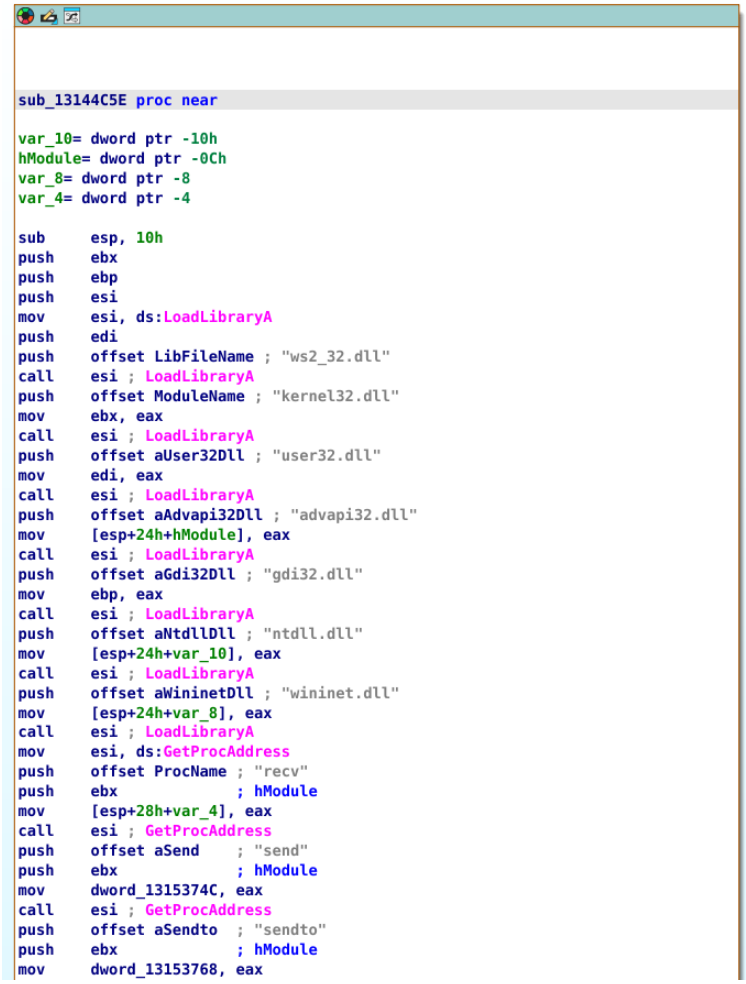
Kuva 17. Näkymä uudelleennimetyistä aliohjelmista.

Tarkastelemalla sub_13144C5E-aliohjelmaa tarkemmin, voidaan todeta tämän tuovan lisätoiminnallisuuksia haittaohjelmaan. Kuvassa 18 on esitetty menetelmä, jolla ohjelma tuo lisätoimintoja lataamalla lisää kirjastoja sekä niiden sisältämiä aliohjelmia. Aiemmin todettiin ohjelmanäytteen tuovan vain neljä kirjastoa, mutta kuten kuvassa näkyy, tämä aliohjelma tuo lisää kirjastoja. Nämä kirjastot ovat:

- ws2_32.dll
- kernel32.dll
- user32.dll
- advapi32.dll
- gdi32.dll
- ntdll.dll

- wininet.d.

Nämä sisältävät monia haittaohjelmien yleisesti käyttäviä aliohjelmia, kuten *CreateMutexA-, RegOpenKeyExA-*, *RegCloseKey-* ja *OpenServiceA*-aliohjelmia. Näitä käytetään rekisterien muokkaamiseen ja palveluohjelmien aloittamiseen. Haittaohjelmat käyttävät usein mutexeja estämään järjestelmän uudelleentartunnan samalla tai eri haittaohjelmaversiolla, johon käytetään *CreateMutexA*-aliohjelmaa.

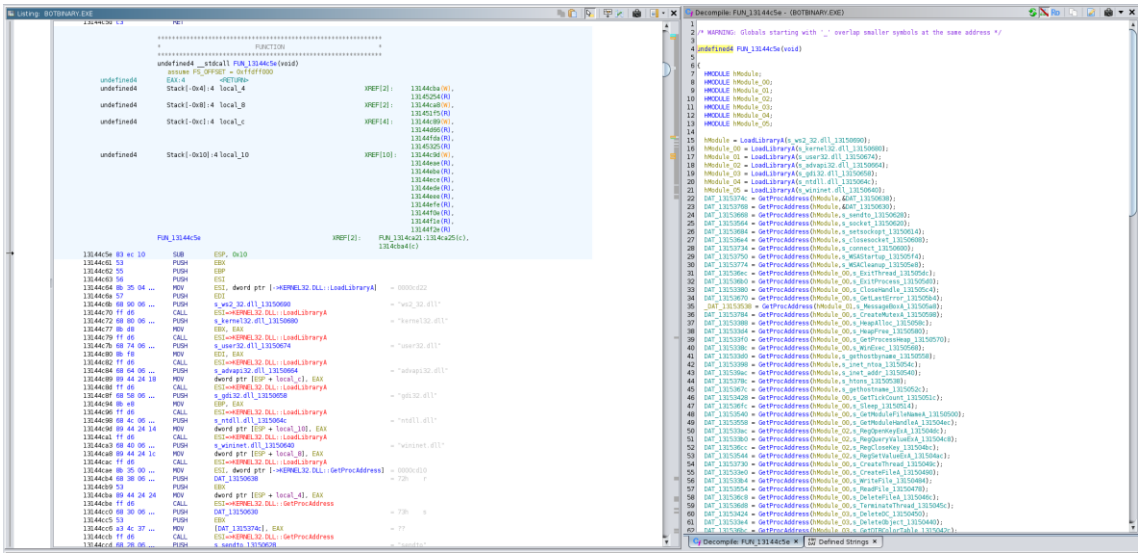


```
sub_13144C5E proc near

var_10= dword ptr -10h
hModule= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

sub     esp, 10h
push    ebx
push    ebp
push    esi
mov     esi, ds:LoadLibraryA
push    edi
push    offset LibFileName ; "ws2_32.dll"
call    esi ; LoadLibraryA
push    offset ModuleName ; "kernel32.dll"
mov     ebx, eax
call    esi ; LoadLibraryA
push    offset aUser32Dll ; "user32.dll"
mov     edi, eax
call    esi ; LoadLibraryA
push    offset aAdvapi32Dll ; "advapi32.dll"
mov     [esp+24h+hModule], eax
call    esi ; LoadLibraryA
push    offset aGdi32Dll ; "gdi32.dll"
mov     ebp, eax
call    esi ; LoadLibraryA
push    offset aNtdllDll ; "ntdll.dll"
mov     [esp+24h+var_10], eax
call    esi ; LoadLibraryA
push    offset aWininetDll ; "wininet.dll"
mov     [esp+24h+var_8], eax
call    esi ; LoadLibraryA
mov     esi, ds:GetProcAddress
push    offset ProcName ; "recv"
push    ebx             ; hModule
mov     [esp+28h+var_4], eax
call    esi ; GetProcAddress
push    offset aSend    ; "send"
push    ebx             ; hModule
mov     dword_1315374C, eax
call    esi ; GetProcAddress
push    offset aSendto  ; "sendto"
push    ebx             ; hModule
mov     dword_13153768, eax
```

Kuva 18. *sub_13144C5E*-aliohjelmanäkymä IDA:ssa.

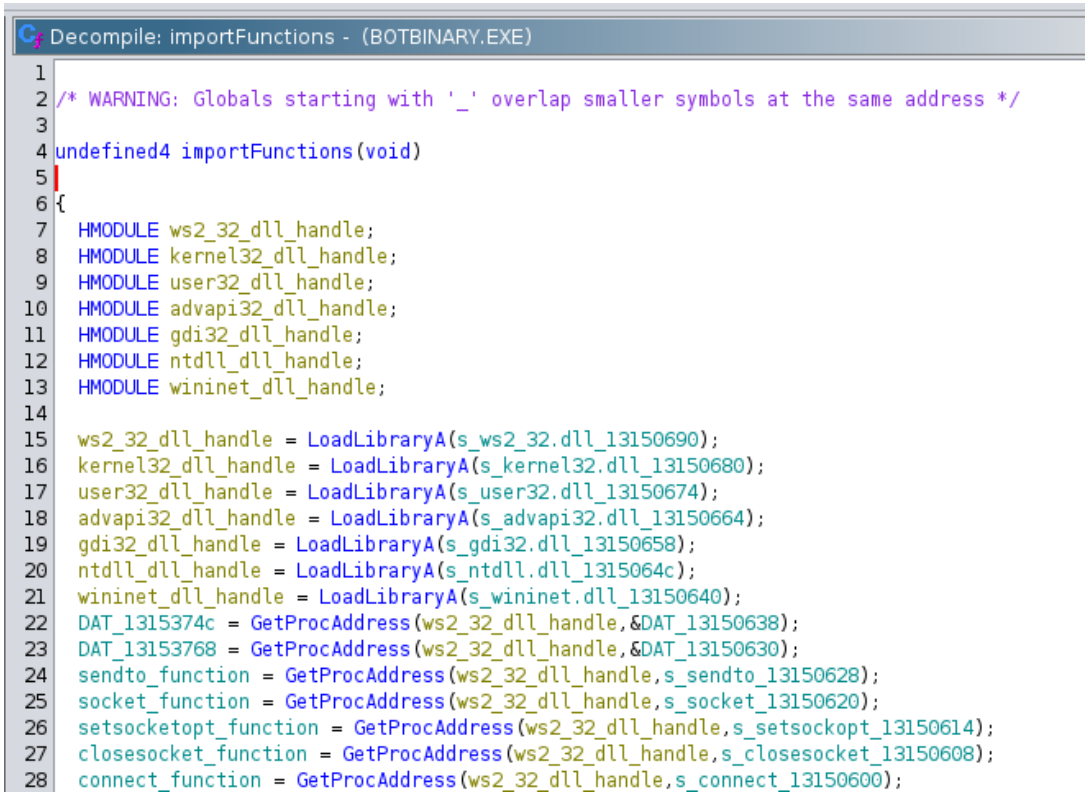
Ghidrassa tämä näkymä on vaikeampi lukea, mutta sama johtopäätös on tulkittavissa tarkastelemalla *Decompile*-ikkunaa. Kuvassa 19 näkyvät samat ohjelmakutsut sekä niiden tallennus data-osioon. Näkymän oikea puoli, *Decompile*-

ikkuna sisältää C-ohjelmakielellä rekonstruoidun näkymän, joka on helppolukuisempi kuin Assembly-ohjelmointikieli. Tässä näkyy selkeämmin, miten eri aliohjelmat tuodaan haittaohjelman käytettäviksi. Esimerkkinä *DAT_13153668*-muuttujaan tallennetaan *hModule*-muuttujasta, *ws2_32.dll*-kirjastosta, *sendto*-aliohjelma.



Kuva 19. *FUN_13144c5e*-aliohjelmanäkymä Ghidrassa.

Toiminnallisuusanalyysiä helpottaa kyky uudelleennimetä muuttujia sitä mukaa, kun ne on tunnistettu. Koska aliohjelman tarkoitus on nyt tunnistettu, se voidaan nimetä esimerkiksi *importFunctions*-nimiseksi. Tämä kuvastaa sitä, mitä se tekee haittaohjelman sisällä. Samoin muuttujat ja viitteet voidaan nimetä kuvastamaan näiden toiminnallisuuksia. Kuva 20 toimii esimerkkinä tästä, jossa nimettiin tunnetut toiminnallisuudet sekä muuttujanimet, tulkinnan helpottamiseksi.



```
Decompile: importFunctions - (BOTBINARY.EXE)
1
2  /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4  undefined4 importFunctions(void)
5
6  {
7    HMODULE ws2_32_dll_handle;
8    HMODULE kernel32_dll_handle;
9    HMODULE user32_dll_handle;
10   HMODULE advapi32_dll_handle;
11   HMODULE gdi32_dll_handle;
12   HMODULE ntdll_dll_handle;
13   HMODULE wininet_dll_handle;
14
15   ws2_32_dll_handle = LoadLibraryA(s_ws2_32.dll_13150690);
16   kernel32_dll_handle = LoadLibraryA(s_kernel32.dll_13150680);
17   user32_dll_handle = LoadLibraryA(s_user32.dll_13150674);
18   advapi32_dll_handle = LoadLibraryA(s_advapi32.dll_13150664);
19   gdi32_dll_handle = LoadLibraryA(s_gdi32.dll_13150658);
20   ntdll_dll_handle = LoadLibraryA(s_ntdll.dll_1315064c);
21   wininet_dll_handle = LoadLibraryA(s_wininet.dll_13150640);
22   DAT_1315374c = GetProcAddress(ws2_32_dll_handle,&DAT_13150638);
23   DAT_13153768 = GetProcAddress(ws2_32_dll_handle,&DAT_13150630);
24   sendto_function = GetProcAddress(ws2_32_dll_handle,s_sendto_13150628);
25   socket_function = GetProcAddress(ws2_32_dll_handle,s_socket_13150620);
26   setsocketopt_function = GetProcAddress(ws2_32_dll_handle,s_setsockopt_13150614);
27   closesocket_function = GetProcAddress(ws2_32_dll_handle,s_closesocket_13150608);
28   connect_function = GetProcAddress(ws2_32_dll_handle,s_connect_13150600);
```

Kuva 20. *importFunctions*-aliohjelmanäkymä, jossa näkyvät päivitetyt muuttujanimet.

Sekä Ghidraa että IDAa tarkastelemalla voidaan todeta ohjelmanäytteen sisältävän monia toiminnallisuuksia, joita nähdään usein haittaohjelmistoissa. Näitä ovat esimerkiksi erilaiset verkko-yhteyksien luomiseen tarvittavat aliohjelmat, Windows-rekisteriavainten muokkaukseen tarvittavat aliohjelmat, sekä tiedostojen luku-, muokkaus- ja keruuominaisuudet.

4.4.2 Vertailu

Haittaohjelman sisäisten aliohjelmien toiminnallisuuden tulkitseminen on usein työläin osa haittaohjelman takaisinmallinnusta. Tämä usein vaatii huomattavan määrän manuaalista työtä, kuten esimerkiksi Windows API -dokumentaation lukua. Lisäksi itse haittaohjelman omien toteutusten tutkiminen voi usein viedä suuria aikoja.

Mikäli IDA Freetä käytetään ilman yhteyttä pilvipalveluihin, haittaohjelmakoodin analysoiminen muuttuu usein aiempaa työläämmäksi. Tällöin tutkijalta

vaaditaan enemmän Assembly-ohjelmointikoodin tuntemista, joka todennäköisesti hidastaa monia vasta-aloittelijoita.

Ghidran sisältämä disassemblointi nopeuttaa aliohjelman tulkitsemista, mutta kuten kohdassa 4.1.2 todettiin, tämä vaatii tutkijalta erityistä huomiota, sillä automatisoitu koodin takaisinmallintaminen voi tehdä virheitä. Siksi on usein suotavaa, että tutkijat osaavat jonkin verran Assembly-ohjelmointikoodin tulkitsemista, jolloin tutkija kykenee varmistamaan puretun koodin todenmukaisuuden.

Molemmat ohjelmat myös tukevat erityisen vahvasti eri muuttujien ja aliohjelmien nimien muuttamista. Tämä huomattavasti parantaa analyysin luettavuutta tulevaisuudessa, jolloin eri aliohjelmakutsujen ristiviittaus on tarpeen, kuin myös tiedonjaossa muiden tutkijoiden kanssa.

Taulukko 4 osoittaa näiden työkalujen olevan yhdenmukaisia toimintojen takaisinmallintamistarkoitukseen. Suurin ero ohjelmistojen välillä on Ghidran tuoma käyttöliittymätuki, joka sisältää puretun haittaohjelmakoodin, kun taas IDA Free kykenee samaan vain pilviresursseihin kytkettynä. Mutta kun otetaan huomioon IDA:n sisältämä aliohjelman logiikkakaavion, sekä sen automaattisesti luodut kommentit, IDA helpottaa takaisinmallintamisprosessia huomattavasti. Ghidra tukee myös näitä samoja toimintoja, joten aliohjelman analyysi suoriutuu huomattavasti nopeammin Ghidran avulla.

Taulukko 4. Eri indikaattorien vertailu toiminnallisuusanalyysin aikana.

Ilmaisin	**Ghidra**	**IDA Free**	**Lisäkommentit**
Käyttöliittymätuki	Korkea	Keskitasoinen	Ghidra sisältää disassembloidun koodin. Molemmat ohjelmistot sisältävät graafisen tuen aliohjelmien tulkitsemiseen sekä kaavioiden että

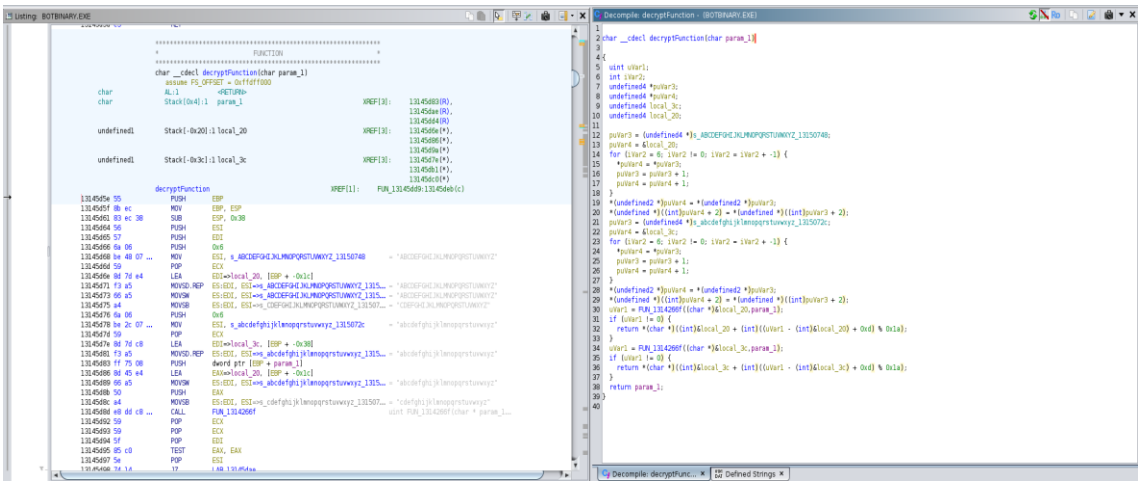
			uudelleennimeämistoimintojen kautta.
Kontekstuaalisen lisätiedon laatu	Keskitasoinen	Keskitasoinen	Ristiviittaus aliohjelmakutsujen välillä on erinomainen. Aliohjelmat ja muuttujat on selkeästi nimetty molemmissa ohjelmistoissa. Molemmat ohjelmistot sisältävät auttavia kommentteja.
Tutkijalta vaadittava manuaalinen työ	Korkea	Korkea	Itse aliohjelmien tulkitseminen vaatii tutkijalta merkittävää työtä, mutta Ghidran automaattinen koodinpurku nopeuttaa työtä.

4.5 Salauksen takaisinmallintaminen

Tutkimuksen aikana myös ilmeni monia salattuja merkkijonoja, joiden sisältö ei ollut luettavissa. Haittaohjelman analyysiprosessiin voi usein kuulua käytettyjen salausalgoritmien tai muun toiminnallisuuden takaisinmallintaminen Assembly-kielestä ylemmän tason kieleen. Tämä voidaan saavuttaa tarkastelemalla ohjelman eri aliohjelmia, joissa näitä salattuja merkkijonoja käsitellään ja muokataan, sillä salatut merkkijonot täytyy jossain vaiheessa purkaa, jotta haittaohjelma voi ottaa ne käyttöönsä.

Ghidra antaa mahdollisuuden tarkastella takaisinkäännettyä versiota Assemblysta. Tämä tekee analyysistä huomattavasti helpompaa, sillä Ghidra kääntää tämän korkeamman tason C-ohjelmointikieleksi.

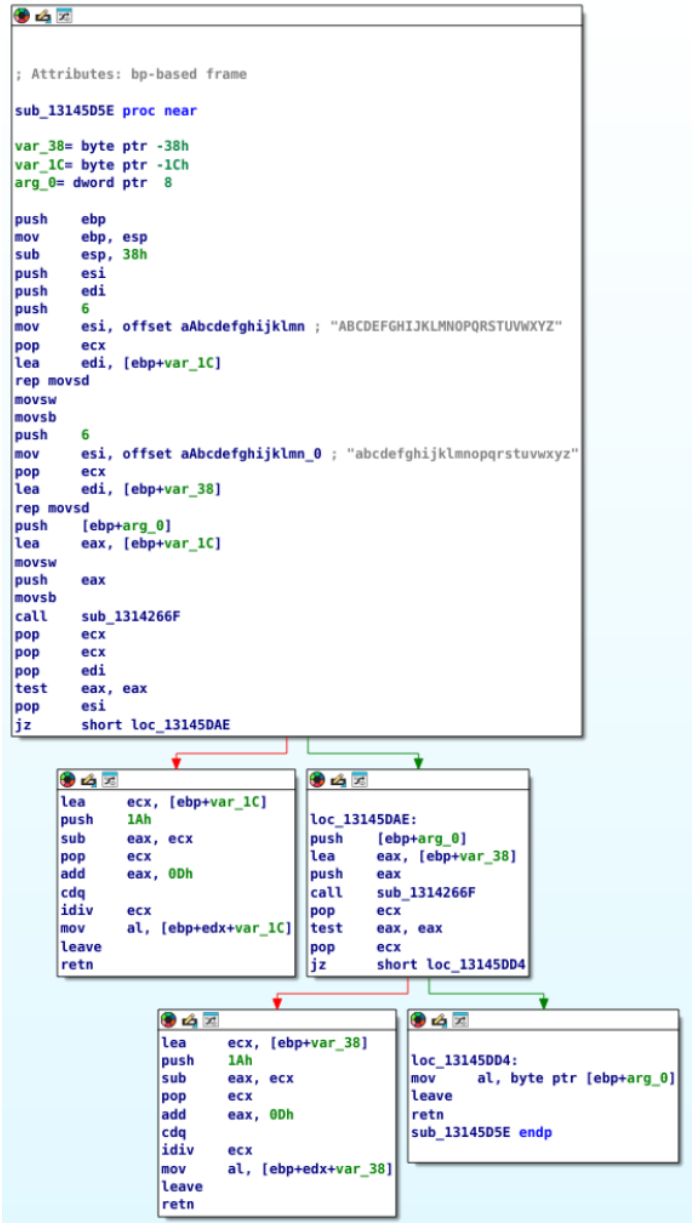
Tarkastelemalla aliohjelmaa kuvassa 21 voidaan tulkita tämän käyttävän “Rotation 13” (ROT13) -salausmenetelmää. Ohjelma suorittaa tämän ottamalla merkkijonon yksi merkki kerrallaan, jonka jälkeen tarkistetaan sen olevan iso tai pieni kirjain. Mikäli kirjain on erikoismerkki, kuten “/”, “\”, tai “?”, aliohjelma palauttaa merkin sellaisenaan. Muussa tapauksessa, aliohjelma siirtyy aakkosissa 13 merkkiä eteenpäin ja palauttaa tämän. Mikäli merkki on aakkosten lopussa, aliohjelma kiertää takaisin alusta. Tämä salaus on vanha ja helposti murrettavissa, minkä takia se ei ole enää yleisesti käytössä. (Christopher Swenson, 2008: 5.)



Kuva 21. *decryptFunction*-aliohjelmanäkymä.

Kun salausmekanismi on tiedossa, voidaan ottaa alkuperäinen salattu merkkijono ja purkaa salaus. Esimerkkinä löydetty merkkijono “FBSGJNER\Zvpebfbsg\Jvaqbjf AG\PheeragIrefvba\Jvaybtba” muuttuu merkkijonoksi “SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon". Tämä on rekisteriavain, jota voidaan käyttää automatisoimaan käyttäjän sisäänkirjautuminen. (Microsoft, 2024.) Toinen mielenkiintoinen merkkijono on “SOFTWARE\Microsoft\Windows\CurrentVersion\Run”, jota voidaan käyttää määrittämään, mitä ohjelmia ajetaan käyttäjän kirjautuessa sisään. (Microsoft, 2024.) Näitä käyttämällä haittaohjelma luo itselleen erilaisia pysyvyysmekanismeja, joiden avulla haittaohjelma voi automatisoida oman käynnistymisen joka kerta, kun laite käynnistyy.

IDA Freen *decompile*-ominaisuus eriää vahvasti Ghidrasta. Vaikka ohjelma mahdollistaa Assembly-koodin purkamisen ja muuntamisen korkeamman tason kieleksi, tämä vaatii yhteyden IDA:n pilviresursseihin. Koska tutkimus tehtiin täysin internetistä eristetyllä järjestelmällä, koodin purkaminen ei ole mahdollista IDA:n ilmaisversiolla. Kuten kuvan 22 näkymä osoittaa, ohjelma on edelleen tulkittavissa, mutta salausalgoritmin takaisinmallinnus riippuu tutkijan kyvystä tulkita Assembly-kieltä.



Kuva 22. *sub_13145D5E*-aliohjelmanäkymä.

4.5.1 Vertailu

Ghidran tuki koodin purkamiselle on erityisen suuri apu eri aliohjelmien tulkitsemiseen. Tämä antaa nopean yleiskuvan siitä mitä komentoja haittaohjelma suorittaa ajon aikana. Taulukko 5 osoittaa Ghidran käyttöliittymätuen sekä kontekstuaalisen lisätiedon laatu olivat huomattavan suuri apu kyseisen aliohjelman tutkimiseen.

Kuten taulukosta 5 nähdään, molempien ohjelmien antama informaatio on muuten lähes sama. Muuttujien nimet ovat samanlaisia ja graafinen funktiokaavio auttaa aliohjelman sisäisen logiikan seuraamisessa. Lisäksi sekä Ghidra että IDA Free lisäävät kommentteja, jotka edesauttavat tutkijaa ymmärtämään laajemman kontekstin, kuten kuvista 21 ja 22 näkyy. Molemmat ohjelmat myös korostavat aakkosmerkkien käyttöä, joita tässä kontekstissa usein pidetään sekä salaus- että salauksenpurku-toimintojen ilmaisimena. Myös kohdassa 4.4.2 mainittu ominaisuus, jolla tuetaan muuttujien nimien muuttamista, on avuksi.

Taulukko 5. Eri indikaattorien vertailu salauksen purkamisen analysoinnissa.

Ilmaisin	**Ghidra**	**IDA Free**	**Lisäkommentit**
Käyttöliittymätuki	Korkea	Keskitasoinen	Ghidra sisältää disassembloidun koodin, mikä helpottaa analyysiprosessia.
Kontekstuaalisen lisätiedon laatu	Korkea	Keskitasoinen	Muuttujien tulkitseminen on molemmissa ohjelmistoissa suoraviivaista.
Tutkijalta vaadittava manuaalinen työ	Keskitasoinen	Korkea	Itse aliohjelman tulkitseminen vaatii manuaalista työtä, mutta Ghidran automaattinen disassemblointi

			nopeuttaa prosessia huomatt.

5 Tulokset

Kun verrataan Ghidran ja IDA Freen ominaisuuksia aloittelevien haittaohjelmia tutkivien kannalta toiminnallisuudessa, käytettävyydessä ja yhteisön tuessa on useita tärkeitä eroja, jotka tulee ottaa huomioon.

Välittömin ero on hinta. IDA Free on rajoitettu versio IDA Prosta, joka on kallis työkalu omalla lisenssillä, kun taas Ghidra on ilmainen ja avoimen lähdekoodin työkalu, joten sitä voi käyttää laajasti ja vähäisin rajoituksin. Ghidra on siten kaikkien saatavilla ilman lisenssimaksuja ja tarjoaa suurimman osan sen ominaisuuksista heti. Tähän verrattuna IDA Free sisältää merkittäviä rajoituksia kalliimpaan IDA Prohon verrattuna, kuten sen tuki vain x86- ja x64-arkkitehtuureille ja kyvyttömyys käyttää useita IDA Pron saatavilla olevia edistyneempiä laajennuksia. Tämä sisältää takaisinmallintamiselle keskeisen ominaisuuden käytön - *decompiler* ominaisuuden offline-tilassa. Takaisinmallintaessa muiden kuin x86-arkkitehtuurien binääreissä, joita nähdään usein IoT-laitteissa, mobiililaitteissa tai sulautetuissa järjestelmissä löytyvissä haittaohjelmissa, Ghidralla on ilmeinen etu.

Käyttöliittymää verrattaessa Ghidra sisältää nykyaikaisemman, Java-pohjaisen graafisen käyttöliittymän, joka saattaa olla käyttäjien mielestä intuitiivisempi ja käyttäjäystävällisempi. Ghidra tarjoaa myös decompiler-ominaisuuden, joka tukee useita arkkitehtuureja. Se on ratkaiseva ominaisuus monimutkaisten haittaohjelmien ymmärtämisessä kääntämällä kokoonpanokoodin takaisin ihmisen luettavammaksi pseudo-C-koodiksi. IDA Prossa on tehokas decompiler-ominaisuus, mutta se on maksullinen lisäosa ja IDA Freen tarjoama versio tästä vaatii yhteyden IDA:n pilvipalvelujen tukea. Tämän vuoksi käyttäjät, joilla on rajoitettu verkko, eivät pysty hyödyntämään takaisinmallintamiseen käytettyä kriittistä työkalua. Jos binäärin purku on välttämätöntä, Ghidran työkalut ovat välittömästi ja yleisemmin saatavilla verkosta riippumatta.

Kokemus Ghidran käytöstä on kuitenkin toisinaan hitaampaa, varsinkin kun kyse on suurempista binääreista. IDA Freen käyttöliittymä on näkymältään vanhempi, mutta se oli huomattavasti nopeampi useaan otteeseen, erityisesti hakemistoissa ja navigaatio eri muistiosoitteiden välillä. Myös automatisoitu analyysi oli huomattavasti nopeampi kuin Ghidralla. Yksinkertaisempi käyttöliittymä helpottaa myös asettelun ymmärtämistä, kun tutkija on tottunut siihen, tai asetellut käyttöliittymän mieleisekseen.

Koska Ghidra on myös avoimen lähdekoodin ohjelma, se hyötyy kasvavasta takaisinmallintajien yhteisöstä, joka toimittaa laajennuksia, komentosarjoja ja eri moduuleja, jotka laajentavat sen toimintoja. Nämä ovat edistyneempiä ominaisuuksia, mutta keskitason ja kokeneet käyttäjät hyötyvät näistä. Yhteisö on myös erittäin tärkeä oppimisprosessissa ja tutkimuksen aikana sekä IDA Freen että Ghidran yhteisöt auttoivat kovasti molempien ohjelmistojen käyttöönottoa. IDA:lla on vakiintunut käyttäjäkunta, koska se on yli 30 vuotta vanha. Tästä johtuen, IDA:n käytön helpottamiseksi on olemassa useita artikkeleita ja kirjoja, jotka tarjoavat runsaasti tietoa aloittelijoille, keskitason ja jopa edistyneille takaisinmallintajille.

6 Yhteenveto

Ghidra saavutettavuus on IDA Freetä parempi, sekä käyttöliittymästään että tärkeimpien ominaisuuksiensa, kuten Assembly-koodin purkamisen, toteuttamisessa ilman pilviresurssien käyttöä. Koska se on myös avoimen lähdekoodin ohjelmisto, Ghidra on täysin ilmainen. Verrattuna IDA Pro -ohjelmistotuotteeseen, jonka lisenssi maksaa 1975 euroa per Windows-lisenssi. Tämä saattaa olla vaikeampaa perustella ellei tutkijalla ole tarvetta nimenomaan sellaiselle toiminnallisuudelle, jota Ghidra ei tarjoa.

Yhteenvetona voidaan todeta, että haittaohjelmia takaisinmallintaessa, Ghidra tarjoaa useita keskeisiä etuja IDA Free -ohjelmistoon verrattuna. Siinä on laajempi arkkitehtuurituki, sisäänrakennettu kääntäjä ja paremmat komentosarja- ja automaatiovaihtoehdot kokeneille käyttäjille. Se saavuttaa kaiken tämän ilman

lisenssimaksuja. IDA Free on edelleen hyödyllinen työkalu erityisesti niille, jotka tarvitsevat reagoivamman käyttöliittymän tai työskentelevät yksinomaan x86/x64-binäärien kanssa, mutta se on rajoitettu verrattuna sekä Ghidraan että IDA Pron täysversioon. Ghidran avoimen lähdekoodin luonne ja yhteisölähtöinen kehitys tekevät siitä joustavamman vaihtoehdon erityisesti käyttäjille, jotka käsittelevät kehittyviä haittaohjelmauhkia eri alustoilla.

Etenkin vasta-aloittavalle takaisinmallintamista opiskelevalle Ghidra on huomattavasti parempi vaihtoehto. IDA on kuitenkin erittäin vankka kolmen vuosikymmenen kehitystyönsä ja yritystuen ansiosta. IDA Free hyötyy tietysti myös IDA:n lisensoidusta versiosta, jonka myynnin avulla Hex Rays myös tukee IDA Freen kehitystä. Sen käyttöliittymä on myös lähtökohtaisesti helpompi tulkita, mutta molempia ympäröi suuri yhteisö, jonka ansiosta uudet tulokkaat voivat oppia ja opiskella suhteellisen helposti.

Jatkotutkimuksen aiheeksi voisi kuulua osallistuminen avoimen lähdekoodin kirjoittamiseen Ghidran repositoriossa, jotta voitaisiin parantaa ohjelmaa näiden havaintojen perusteella. Tällä pyrittäisiin parantamaan responsiivisuutta tai lisäämän uusia toimintoja haitallisten binäärien havaitsemisen helpottamiseksi.

Kehittyneempiä haittaohjelmien analysointitekniikoita sekä Ghidrassa että IDA:ssa sopisi vertailla syvällisemmin, kuten lisäosien ja komentosarjojen luominen tiettyjen haittaohjelmanäytteiden analysoimiseksi. Kattavampi määrä monimutkaisempia haittaohjelmia olisi myös omiaan parantamaan vertailutuloksia jatkotutkimuksen aikana.

Lähteet

Binary Ninja. 2024. Verkkoaineisto. <https://binary.ninja/free>. Päivitetty 16.8.2024. Luettu 31.10.2024.

Cimpanu, Catalin. 2019. Verkkoaineisto. Zdnet. <https://www.zdnet.com/article/nsa-release-ghidra-a-free-software-reverse-engineering-toolkit/>. Päivitetty 5.3.2019. Luettu 16.9.2024.

Eagle, C. (2008) *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler.*

Hex-Rays. 2024. Verkkoaineisto. IDA. <https://hex-rays.com/ida-pro>. Luettu 31.10.2024.

Indicators of Compromise. Verkkoaineisto. Fortinet. <https://www.fortinet.com/resources/cyberglossary/indicators-of-compromise>. Luettu 16.9.2024.

Jones, S. Verkkoaineisto. Bitdefender. <https://www.bitdefender.com/en-us/blog/businessinsights/what-is-dynamic-malware-analysis/>. Päivitetty 21.3.2023. Luettu 16.9.2024.

Sikorski, M. and Honig, A. (2012) *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. http://cds.cern.ch/record/1526301.

Swenson, C. (2008) *Modern Cryptanalysis: Techniques for Advanced Code Breaking.*

Manpages. Verkkoaineisto. Debian. <https://manpages.debian.org/stretch/manpages/elf.5.en.html>. Päivitetty 5.4.2017. Luettu 16.9.2024.

Microsoft Learn. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/windows/win32/debug/pe-format>. Päivitetty 29.2.2024. Luettu 16.9.2024.>. Päivitetty 29.2.2024. Luettu 16.9.2024.

Microsoft Learn. 2023. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexa>. Päivitetty 2.9.2023. Luettu 16.9.2024.

Microsoft Learn. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/windows/win32/api/heapapi/nf-heapapi-heapfree>. Päivitetty 6.2.2024. Luettu 16.9.2024.

Microsoft Learn. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/windows/win32/api/heapapi/nf-heapapi-heapalloc>. Päivitetty 2.2.2024. Luettu 16.9.2024.

Microsoft Learn. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/windows/win32/setupapi/run-and-runonce-registry-keys>. Päivitetty 19.7.2024. Luettu 16.9.2024.

Microsoft Learn. 2024. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/troubleshoot/windows-server/user-profiles-and-logon/turn-on-automatic-logon>. Päivitetty 11.3.2024. Luettu 16.9.2024.

Pancake. 2016. Verkkoaineisto. Chaos Computer Club. <https://media.ccc.de/v/33c3-8095-radare_demystified>. Päivitetty 29.12.2016. Katsottu 31.10.2024.

Radare Organization. 2024. Verkkoaineisto. <https://rada.re/n/>. Luettu 31.10.2024.

Web Archive. 2024. Verkkoaineisto. Apple. <https://web.archive.org/web/20140904004108/https://developer.apple.com/library/mac/documentation/developertools/conceptual/MachORuntime/Reference/reference.html>. Päivitetty 4.9.2014. Luettu 16.9.2024.

Wysopal, C., Eng, C. & Shields, T. Static detection of application backdoors. DuD 34, 149–155 (2010). https://doi.org/10.1007/s11623-010-0024-4.

Zeltser Security Corp. 2018. Verkkoaineisto. <https://remnux.org>. Päivitetty 2020. Luettu 30.10.2024.