jamk

# Muistiforensiikan automatisointi-ratkaisun suunnittelu ja toteutus

Tuomo Viljakainen

Opinnäytetyö, AMK
Toukokuu 2021
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

**Kuvailulehti**

**Viljakainen, Tuomo**

**Muistiforensiikan automatisointiratkaisun suunnittelu ja toteutus**

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2021, 100 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: Suomi

Verkkojulkaisulupa myönnetty: kyllä

**Tiivistelmä**

Jyväskylän ammattikorkeakoulun yhteydessä toimivalla JYVSECTECillä oli tarvetta muistiforensiikan prosesseja automatisoivalle järjestelmälle. Kyseistä järjestelmää lähdettiin rakentamaan innovatiiviseen konstruktioon tähtäävän konstruktiivisen tutkimusotteen avulla. Konstruktiivisessa tutkimuksessa läheinen yhteistyö kohdeorganisaation kanssa on tärkeää, joten toimeksiantajan kanssa pidettiin tietyin väliajoin palavereita, joissa keskusteltiin tuotteen kehityksestä.

Tuotteen toteutustavaksi valittiin Docker-konttien päälle rakennettava järjestelmä nimeltään Autovola, joka sisältää kontin muistivedoksista saatua tietoa säilövälle MongoDB-tietokannalle sekä Flask-rajapintaa ja React-käyttöliittymää ylläpitävälle Apache-palvelimelle. Näiden konttien lisäksi järjestelmään kuuluvat vielä Volatility 3:n lisäosia muistivedoksiin suorittavat Analyysi-kontit, joiden määrään ja resursseihin järjestelmän ylläpitäjä pystyy vaikuttamaan.

Käyttäjät voivat ladata Autovolan käyttöliittymästä järjestelmään muistivedoksia ja ISF-tiedostoja sekä päättää, mitä Volatility 3:n lisäosia muistivedoksiin suoritetaan. Lisäosien tuloksia voidaan tutkia ja suodattaa jälkeenpäin käyttöliittymästä. Autovola tukee Linux- ja Windows-käyttöjärjestelmistä otettujen muistivedosten analyysia.

Valmista tuotetta arvioitiin toimeksiantajan palautteen ja kolmiosaisen markkinatestin perusteella. Tuote läpäisi markkinatestin ensimmäisen tason, sillä se otettiin käyttöön toimeksiantajan organisaatiossa. Järjestelmää käytetään muun muassa JYVSECTECin kyberharjoituksissa ja työvälinekoulutuksissa. Toisen tason läpäiseminen olisi vaatinut tuotteen käyttöönottamista myös joissain muissa organisaatioissa. Järjestelmän katsottiin sijoittuneen markkinatestin tasojen yksi ja kaksi väliin, koska muut organisaatiot voivat käyttää tuotetta JYVSECTECin tarjoamien palveluiden kautta. Toimeksiantaja oli tyytyväinen tuotteeseen ja koki sen vastaavan sitä, mitä järjestelmältä oli haluttu.

Opinnäytetyön tavoitteena oli luoda JYVSECTECille käyttökelpoinen tuote, joka nopeuttaisi muistiforensiikkaan liittyviä prosesseja sekä helpottaisi analysoijien välistä yhteistyötä. Näiden tavoitteiden katsottiin toteutuneen kokonaan tai osittain. Toimeksiantaja tulee vielä jatkokehittämään järjestelmää siihen liittyvien tarpeitten ja käyttökokemusten perusteella.

**Avainsanat (asiasanat)**

Muistiforensiikka, automatisointi, JYVSECTEC, Volatility, haittaohjelmat, Docker, Apache, MongoDB, Python

**Muut tiedot (salassa pidettävät liitteet)**

jamk

**Description**

**Viljakainen, Tuomo**

**Design and implementation of memory forensics automation solution**

Jyväskylä: JAMK University of Applied Sciences, May 2021, 100 pages.

Information and Communications. Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for web publication: Yes

Language of publication: Finnish

**Abstract**

JYVSECTEC, which operates as a part of Jyväskylä University of Applied Sciences, needed a system that automates memory forensics processes. Constructive research approach aiming at innovative construction was used during the development process. In constructive research, close cooperation with the target organization is important, hence meetings with the client were held at regular intervals to discuss product development.

The construction was implemented by building a system called Autovola, which is based on Docker containers. One of the containers contains MongoDB database that stores information gathered from memory dumps and there is also a second container with Apache web server that maintains React built UI and Flask API. In addition, there are Analysis containers that run Volatility 3 plugins to memory dumps. Number of Analysis containers can be changed and resources that they utilize may also be modified.

Users can upload memory dumps and ISF files to Autovola by using its user interface. They may also decide which Volatility plugins are run to the memory dumps. Plugin results can be examined and filtered afterwards using the user interface. Autovola supports analysis of memory dumps taken from Linux and Windows operating systems.

Finished product was evaluated using client's feedback and a three-part market test. The product passed first level of the market test, since JYVSECTEC started to use the finished product. Autovola is used in JYVSECTEC's cyber exercises and tool trainings, among other things. The product did not pass second level since it is not used in any other organization. Autovola was considered to place between levels one and two of the market test, as other organizations may use the product through services provided by JYVSECTEC. The client was satisfied with Autovola and felt that it corresponded to what was desired from the product.

Purpose of the thesis was to create a practical product for JYVSECTEC, which would speed up processes related to memory forensics and ease collaboration between analysts. These objectives were considered to have been fully or partially achieved. Client will further develop the product based on demands and user experiences.

**Keywords/tags (subjects)**

Memory forensics, automation, JYVSECTEC, Volatility, Malware, Docker, Apache, MongoDB, Python

**Miscellaneous (Confidential information)**

**Sisältö**

**Kuviot**

**Taulukot**

# Lyhenteet

| | |
|---|---|
| ADS | Alternate Data Stream |
| AJAX | Asynchronous JavaScript and XML |
| API | Application programming interface |
| BSON | Binary JSON |
| CnC | Command-and-Control |
| DFRWS | The Digital Forensic Research Workshop model |
| DLL | Dynamic-Link Library |
| DNS | Domain Name System |
| EEPROM | Electronically Erasable Programmable Read-Only Memory |
| EPROM | Erasable Programmable Read-Only Memory |
| EXE | Executable file |
| FTP | File Transfer Protocol |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| IoC | Indicator of Compromise |
| IRC | Internet Relay Chat |
| ISF | Intermediate Symbol File |
| JSON | JavaScript Object Notation |
| MFT | Master File Table |
| MIC | Mandatory Integrity Control |
| NTFS | New Technology File System |
| PDB | Program database |
| PEB | Process Environment Block |
| PID | Process identifier |
| PPID | Parent process identifier |
| RAM | Random Access Memory |
| Regex | Regular expression |
| ROM | Read-Only Memory |
| SANS | Escal Institute of Advanced Technologies |
| SCM | Service Control Manager |
| SSDT | System Service Descriptor Table |
| Tor | The Onion Router |
| WSGI | Python Web Server Gateway |
| YAML | YAML Ain’t Markup Language |

# 1 Johdanto

Opinnäytetyön tarkoituksena oli automatisoida muistiforensiikan prosesseja mahdollisimman paljon. Automatisoitavana työkaluna käytettiin Volatility Frameworkia. Työ toteutettiin JYVSECTECille, joka on Jyväskylän ammattikorkeakoulun yhteydessä toimiva kyberturvallisuuden tutkimus-, kehitys- ja koulutuskeskus. JYVSECTECiltä löytyy erityisasiantuntijuutta muun muassa tietoturvapoikkeamien hallinnan ja IT-tekniikoiden saralta. (Specializing in cyber security expertise n.d.)

Automaation tarkoituksena oli luoda järjestelmä, joka analysoi sille syötettyjä muistivedoksia (engl. memory dump) tietyiltä osa-alueilta ja luo mahdollisista löydöksistä raportin tutkijalle, joka voi jatkaa löydösten syvällisempää tutkintaa. Työssä haluttiin kiinnittää erityistä huomiota järjestelmän analysoimiin osa-alueisiin ja tutkijalle esitettävään raporttiin. Mitä enemmän ja tarkemmin osa-alueita pystytään analysoimaan, sitä parempaa dataa tutkija muistivedoksesta saa. Raporttiin pyrittiin selkeästi listaamaan muistivedoksesta löydetyt mielenkiinnon kohteet, jotta tutkija saa kattavan kokonaiskuvan vedoksesta ja siitä, mitä kannattaisi vielä tutkia syvemmin. Järjestelmästä yritettiin myös tehdä usealle käyttäjälle skaalautuva, mikä helpottaisi tutkijoiden välistä keskinäistä yhteistyötä.

Valmiin järjestelmän lähdekoodia ei julkaista avoimena toimeksiantajan toiveesta, mutta opinnäytetyön dokumentaatio sai sisältää korkeamman tason luonnehdintaa valmiista tuotteesta. Toimeksiantajan mukaan valmista tuotetta voidaan käyttää, jos se osoittautuu hyödylliseksi. Tavoitteena olikin rakentaa JYVSECTECille käyttövalmis tuote, joka palvelee kehityskeskusta sen toivomalla tavalla. Tuotteen tulisi tehostaa olemassa olevia muistiforensiikkaan liittyviä prosesseja, kuten auttaa tuotteen käyttäjiä löytämään analyysille olennaisia yksityiskohtia muistivedoksista nopeammin ja vähentää analyysiin liittyvää manuaalista työtä. Myös käyttäjien välistä yhteistyötä helpottavat toiminnot tuotteessa, kuten muistivedoksiin liittyvän tiedon jakaminen käyttäjien välillä, olisivat hyödyllisiä ominaisuuksia.

# 2 Tutkimusasetelma

Tutkimusongelmana oli JYVSECTECiltä puuttuva muistiforensiikkaa automatisoiva järjestelmä. Muistiforensiikkaan käytettäviä työkaluja oli tarkoitus vertailla, mutta yhteistuumin käytettäväksi työkaluksi toimeksiantajan kanssa rajattiin Volatility Framework, jolloin automaatiojärjestelmän

kehittämiselle jää enemmän aikaa. Alun perin tarkoituksena oli automatisoida sekä muistiforensiikan, että levyforensiikan prosessia, mutta aihetta rajattiin pelkkään muistiforensiikkaan, jotta yhteen osa-alueeseen pystytään syventymään enemmän.

**Tutkimuskysymykset ja -menetelmä**

Tutkimusongelman pohjalta hahmoteltiin seuraavat kysymykset, joihin työssä etsitään ratkaisuja:

- Miten pitkälle muistiforensiikan prosessia voi automatisoida?
- Kuinka löydökset muistivedoksesta esitetään/raportoidaan?
- Mitä teknologioita automatisointiin kannattaisi käyttää?

Tutkimuskysymysten ja työn käytännönläheisen innovatiivisuuteen perustuvan toteutuksen takia tutkimusmenetelmäksi valittiin konstruktiivinen tutkimus. Konstruktiivisessa tutkimuksessa pyritään ratkaisemaan tosielämän ongelma rakentamalla konstruktio. Konstruktio on abstrakti käsite, jolla tarkoitetaan ihmisen kehittämiä luomuksia, kuten erilaisia tuotteita tai malleja. Tutkimuksen aikana suoritettava läheinen yhteistyö toimeksiantajan kanssa on tärkeää, jotta lopullinen tuotos vastaa toimeksiantajan näkemystä. (Lukka 2001.)

Vaikka tutkimustyyppi on käytännönläheinen, toteutettavaa tuotetta suunniteltaessa ja kehitettäessä tulisi käyttää aiempaa tutkimusongelmaan liittyvää teoriaa, jotta lopulliseen tuotokseen johtaneet valinnat voidaan teoriaa käyttäen selkeästi perustella kasvattaen työn arvoa. Tutkimuksen aikana harkittujen ratkaisuvaihtoehtojen tulisi käydä tutkimuksesta ilmi, ja valittu vaihtoehto pitäisi kyetä perustelemaan kattavasti ja puolueettomasti. (Ojasalo, Moilanen & Ritalahti 2015, 65-66.)

Ojasalo ym. (2015, 67) mukaan konstruktiivisen tutkimuksen prosessi voidaan jakaa kuuteen eri vaiheeseen:

1. Sopivan tutkijaa kiinnostavan tutkimusongelman löytäminen.
2. Perehtyminen tutkimusongelmaan ja siihen liittyvän tiedon kerääminen.
3. Tutkimusongelmaan perustuvan tuotteen rakentaminen.
4. Vaiheessa kolme toteutetun konstruktion testaaminen peilaten tutkimusongelmaan.
5. Konstruktion kehittämisessä käytettyjen teoreettisten lähteiden esitteleminen sekä tutkimuksen mahdollisen uutuusarvon havainnollistaminen.
6. Pohdintaa konstruktion käyttömahdollisuuksista laajemmalla mittakaavalla.

Toteutettua konstruktiota voidaan arvioida kolmiosaisen markkinatestin avulla. Ensimmäisen tason läpäisemiseksi konstruktion on oltava käyttökelpoinen kohdeorganisaatiolle. Toisen testin läpäisy vaatii konstruktion käyttöönottoa useassa eri organisaatiossa. Kolmannessa testissä onnistuminen taas edellyttäisi konstruktion käyttöönottaneiden yritysten parempaa menestystä verrattuna saman toimialan organisaatioihin, jotka eivät ole ottaneet konstruktiota käyttöön. (Ojasalo ym. 2015, 68.)

Tutkimusmenetelmän etuja ovat esimerkiksi uuden kehittäminen ja rakentaminen itse tutkimisen ohessa, mikä luo tutkimukselle myös materiaalista arvoa. Tutkijat saavat käytännön kokemusta luodessaan uutta tuotetta sekä harjaantuneisuutta projektityöskentelyyn, sillä konstruktiiviseen tutkimukseen kuuluu läheinen yhteistyö toimeksiantajan kanssa. Konstruktiivisen tutkimuksen riskeissä taas korostuu kohdeorganisaation halu pitää tutkimuksen tuloksia salassa, kuten tutkimuksen luoma konstruktio tai osia siitä. Myöskään organisaation intressit tuotteeseen eivät välttämättä ole tarpeeksi vahvat, jotta projektiin haluttaisiin panostaa riittävästi, mikä voi johtaa tutkimuksen keskeytymiseen. Tutkijan tulisikin varmistaa kohdeorganisaation tarve ja kiinnostus saattaa projekti alusta loppuun saakka. (Lukka 2001.)

Opinnäytetyön luonteen puolesta konstruktiiviseen tutkimukseen liittyvät riskit eivät ole niin suuria kuin ne jossain toisessa tutkimuksessa voisivat olla. Toimeksiantajan ei tarvitse määritellä tuotekehitykselle budjettia, koska tuotteen toteuttamisesta ei tarvitse maksaa rahallista korvausta. Tämä myös pienentää toimeksiantajan riskiä olla haluton jatkamaan tuotteen kehittämiseen liittyvää prosessia taloudellisista syistä. Tämä onkin yksi syy, miksi konstruktiivinen tutkimus sopii tämän opinnäytetyön kontekstiin hyvin. Toimeksiantajan riskit ovat pienet ja työn toteuttajan on mahdollista saada arvokasta käytännön kokemusta tuotekehityksestä.

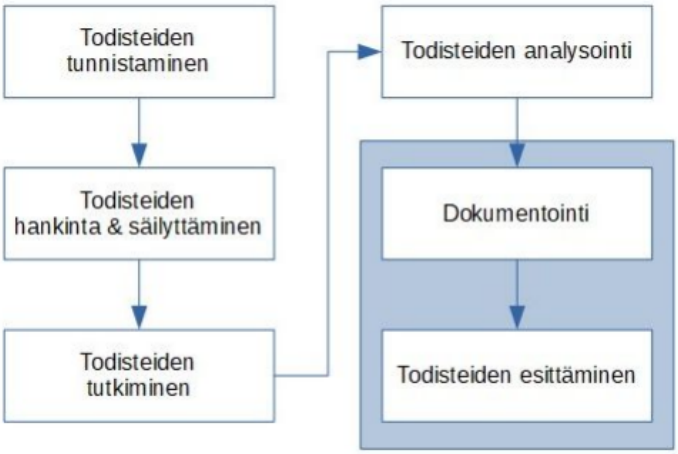
# 3 Digitaalinen forensiikka

Digitaalinen forensiikka on rikosteknisen tutkimuksen ala, joka keskittyy rikostutkinnassa hyödyllisten sähköisten tietojen hakemiseen, tallentamiseen ja analysoimiseen. Tietoa voidaan hakea tietokoneista, kännyköistä, kovalevyistä ja muista sähköisistä tallennuslaitteista. (Digital evidence n.d.) Digitaaliseen forensiikkaan käytetään erityisiä tekniikoita ja työkaluja, joilla tutkitaan tietokonerikollisuuden eri muotoja, kuten petoksia ja laitteisiin murtautumisia (Vacca 2013, DIGITAL FORENSICS).

## 3.1 Digitaalisen forensiikan prosessi

Fyysinen ja digitaalinen tutkinta sisältävät paljon samankaltaisuuksia, mutta perinteiset forensiikan menetelmät eivät sovellu digitaalisen aineiston tutkimiseen, koska niillä ei pystytä keräämään elektronisessa muodossa olevaa aineistoa. Todisteita saattaa löytyä monista eri paikoista, kuten chat-istunnoista, viestintälokeista tai muista digitaalisista jäänteistä. Jos digitaalisen kohteen todetaan olevan jollain tavalla merkityksellinen tutkimuksessa, sitä pidetään digitaalisena todisteena. Tutkinnassa tulisikin pyrkiä löytämään vahvin näyttö käytettävissä olevilla tiedoilla sekä tuottaa asianmukainen dokumentaatio tutkimuksen prosesseista, keskeisistä oletuksista ja epävarmuustekijöistä. (Årnes 2018, Introduction.)

Johansenin (2017, 33) mukaan digitaalisen forensiikan prosessin voi jakaa kuuteen osaan: tunnistaminen, hankinta, säilyttäminen, tutkiminen, analysointi ja raportointi. Tätä kutsutaan DFRWS-malliksi (The Digital Forensic Research Workshop model), joka on standardisoitu ja jota yleensä käytetään perustana parannelluille malleille (Tahiri 2016). Kuviossa 1 on kuvattu prosessin järjestys. Hankinta ja säilyttäminen on sisällytetty toiseen kehykseen, ja raportointi sisältää sekä dokumentoinnin että todisteiden esittämisen.




Kuvio 1. Muistiforensiikan prosessi (Koskela 2020, 5)

**Tunnistaminen**

Årnes (2018) kertoo tutkinnan kannalta tärkeiden kohteiden tunnistamisen olevan prosessin ensimmäinen askel. Vaihe on tärkeä tulevien vaiheiden osalta, sillä tunnistetut kohteet määrittelevät, mitä todisteita tutkinnan aikana kannattaa etsiä. Tutkijat joutuvat tekemään olettamuksia siitä, mitkä järjestelmät tai laitteet saattaisivat sisältää tutkinnan kannalta tärkeää elektronista todistusainestoa. Tunnistamisvaiheessa voidaan esimerkiksi levymuistin kohdalla määritellä, mitkä tiedostot muistissa ovat käytettävissä tai poistettu. (Årnes 2018, The Identification Phase.)

**Hankinta**

Kun ensimmäinen vaihe on valmis ja tutkinnan kannalta merkittävät kohteet tunnistettu, niitä täytyy suojata modifioinnilta. Esimerkiksi tietokoneen kohdalla, sen käyttäjiltä voidaan estää pääsy tietokoneelle ja tietokone voidaan sulkea pois muusta verkosta. Toimilla lievennetään todisteiden mahdollisuutta joutua sabotoiduiksi jälkeenpäin. Todisteita sisältävistä järjestelmistä voidaan myös tallentaa laitteen sen hetkinen tilannekuva (engl. snapshot), eli levynkuva haihtumattomalle muistille. (Johansen 2017, 34.)

**Säilyttäminen**

Johansenin (2017, 35) mukaan prosessin tässä vaiheessa tutkijat alkavat keräämään digitaalista todistusaineistoa säilytykseen. Brezinski ja Killalea (2002) ovat koonneet yhteen listan, jota seuraamalla saadaan kerättyä suurimmassa riskissä oleva aineisto järjestyksessä pienimmässä riskissä olevaan aineistoon. Suurin riski on todistusaineistolla, joka katoaa laitteen sammuessa. Tällaista aineistoa ovat aktiiviset verkkoyhteydet, laitteiden keskusmuistin sisältö, välimuistit, prosessi- ja reititystaulukot sekä väliaikaiset tiedostot. (Brezinski & Killalea 2002, 3.)

Aineistoa keräävien tutkijoiden tulee olla varovaisia, sillä todistusaineisto saattaa muuttua tai tuhoutua, jos sitä käsittelee väärin. Järjestelmä saattaa sisältää haitallisia palveluita tai skriptejä, jotka tarkoituksella tuhoavat tietokoneesta löytyvää dataa. Skriptit saattavat aktivoitua esimerkiksi järjestelmän menettäessä yhteytensä verkkoon. Tutkijoiden tulisikin dokumentoida kaikki toimintansa todistusaineiston parissa ja käyttää aktiivisia järjestelmiä vain tarpeen tullen. (Johansen 2017, 35-36; Brezinski & Killalea 2002, 3-4.)

**Tutkiminen**

Aiemmin kerätystä todistusaineistosta aletaan etsimään ja poimimaan myöhemmin analysoitavaa materiaalia. Kaikki todistusaineisto ei välttämättä ole tutkinnan kannalta merkityksellistä, vaan ne voivat sisältää pieniä tiedonrippeitä, jotka vaativat tarkempaa analysoimista. Esimerkiksi verkkoliikennettä sisältävästä aineistosta voitaisiin vain hakea kaikki FTP-protokollan (File Transfer Protocol) liikenne analysointia varten. Poimittua raakadataa täytyy usein vielä prosessoida jälkeenpäin, jotta rikostutkijan olisi helpompi analysoida sitä prosessin seuraavassa vaiheessa. Tätä vaihetta pystytään automatisoimaan valmiilla ohjelmilla ja skripteillä, mikä vähentää manuaalista työtä huomattavasti, varsinkin isoja datamääriä tutkittaessa. (Johansen 2017, 39; Årnes 2018, The Examination Phase.)

**Analysointi**

Aiemmin poimittua materiaalia analysoidaan, ja jokaisen objektin kohdalla tehdään erillinen päätös siitä, voidaanko sitä käyttää todistusaineistona siihen liittyvässä rikoksessa tai tapahtumassa. Rikos voi olla fyysinen, ja digitaalinen aineisto saattaa sisältää rikoksentekijän motiivia selkiyttävää materiaalia, kuten sähköpostiviestejä ja -osoitteita, joita voidaan käyttää todisteina. (Årnes 2018, The Analysis Phase.)

Tutkittava aineisto saattaa sisältää analyysia haittaavia anti-forensisia metodeja. Yritysmaailmassa tekniikat liittyvät usein levymuistin tyhjentämiseen, sillä muistista saattaa löytyä yritykselle kriittistä dataa, jonka ei haluta päätyvän kolmansille osapuolille. Varsinkin kehittyneemmät haittaohjelmat sisältävät anti-forensiikkaa. Tämä voi olla datan sekoittamista tai salaamista tietyllä algoritmilla, jolla pyritään vaikeuttamaan haittaohjelman toimintojen selvittämistä. (Årnes 2018, The Analysis Phase.)

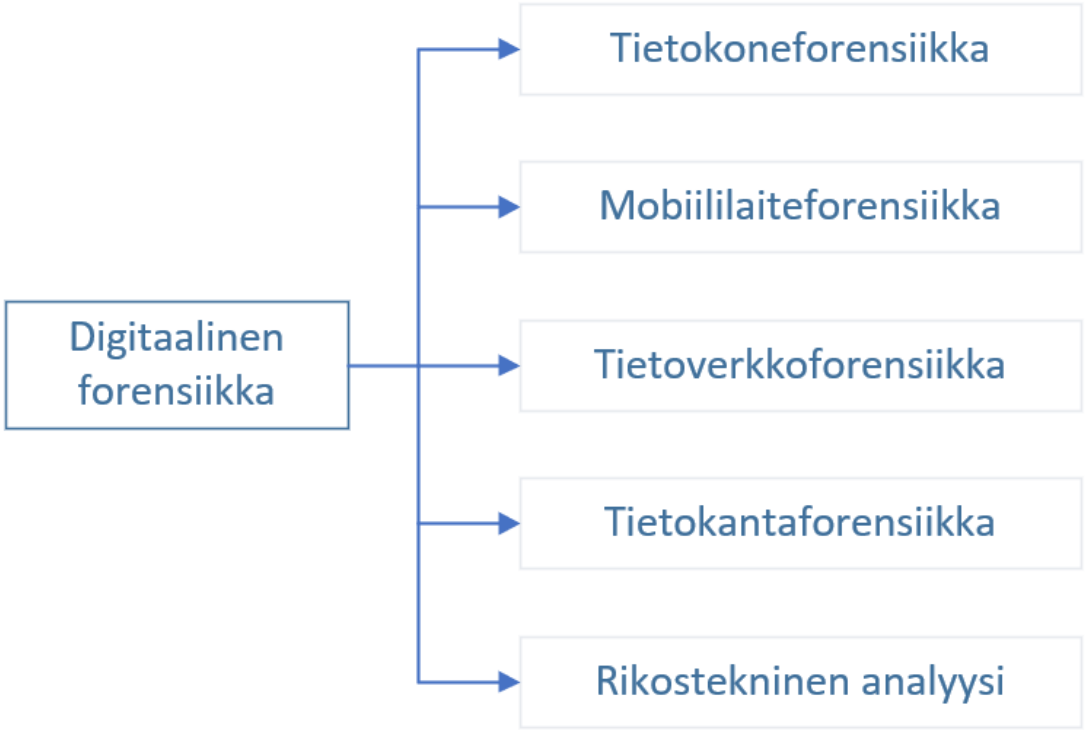
**Raportointi**

Prosessin viimeisessä vaiheessa kaikki tutkinnan tulokset dokumentoidaan ja esitetään yleisölle, kuten tuomioistuimelle, jos tutkinta on liittynyt rikokseen. Raportissa kuvataan tarkasti koko prosessiin ja sen vaiheisiin liittyviä oleellisia seikkoja. Raportti voi sisältää muun muassa tutkinnassa käytettyjä työkaluja, kuvia, aineistoa ja tuloksia. Jos kaikkia aktiviteetteja ei ole dokumentoitu sopivalla tavalla, tuomioistuin tai vastaava taho voi evätä todisteet. (Årnes 2018, The Presentation Phase.)

## 3.2 Digitaalisen forensiikan haarat

Digitaalinen forensiikka tunnettiin aiemmin synonyymina tietokoneforensiikalle, mutta nykyisin termi digitaalinen forensiikka sisältää kaiken digitaaliseen teknologiaan liittyvän forensiikan (Reith, Carr & Gunsch 2002, 3). Digitaalinen forensiikka sisältää yhä enemmän uutta teknologiaa, ja se voidaankin yleisesti jakaa viiteen eri haaraan: tietokoneiden, mobiililaitteiden, verkkojen ja tietokantojen forensiikkaan sekä rikosteknisen datan analyysiin (engl. forensic data analysis) (Tunggal 2020).

Kuviossa 2 on esitetty visuaalisesti digitaalisen forensiikan haarautumista eri alalajeihin. Alalajit jatkavat haarautumistaan vielä yksityiskohtaisempiin kategorioihin, luoden rakenteen, joka kattaa kaikki digitaalisen forensiikan osa-alueet. Tämän opinnäytetyön kannalta oleellisin digitaalisen forensiikan alalaji on muistiforensiikka, joka haarautuu tietokoneforensiikasta (Zhang & Che 2018, 1).




Kuvio 2. Digitaalisen forensiikan haarautuminen

## 3.3 Muistiforensiikka

Muistiforensiikassa analysoidaan laitteiden keskusmuistista löytyvää tietoa. Data keskusmuistissa on binäärimuodossa, ja siitä voidaan oikeilla työkaluilla esimerkiksi etsiä käynnissä olevia prosesseja, luotuja verkkoyhteyksiä, käyttäjien salasanoja tai haittaohjelmiin liittyviä IoC:eja (Indicator of Compromise), muistivedoksia tutkimalla. (Reynolds & Horvath 2017, 14, 28; Zhang & Che 2018, 1.)

### 3.3.1 Metodologia

Johansen (2017, 156) kertoo SANS-instituution (Escal Institute of Advanced Technologies) käyttämästä kuusiosaisesta metodologiasta, jota hyödynnetään muistivedosten analysoinnissa:

1. Ensimmäiseksi muistivedoksesta kannattaa lähteä etsimään järjestelmässä suoritettavia prosesseja. Jos jokin prosessi vaikuttaa jollain tapaa epäilyttävältä, sitä voi lähteä tutkimaan tarkemmin esimerkiksi selvittämällä, mistä tiedostosta kyseistä prosessia suoritetaan.
2. Toisessa vaiheessa kuuluisi tutkia haitallisiksi todettuja prosesseja tarkemmin, kuten niiden yhteyksiä DLL-tiedostoihin (Dynamic-Link Library) (Windows-järjestelmissä).
3. Kolmannessa vaiheessa tutkijan tulisi analysoida internettiin luotuja aktiivisia verkkoyhteyksiä. Niissä huomionarvoista ovat esimerkiksi yhteydet epätavallisiin IP-osoitteisiin.
4. Seuraavaksi pitäisi kiinnittää huomio mahdollisiin koodi-injektioihin. Koodin injektointia käsitellään tarkemmin luvussa 7.2.
5. Viidennessä vaiheessa tarkistetaan, löytyykö muistivedoksesta merkkejä piilohallintaohjelmista, joiden määritelmää ja toimintaa kuvaillaan luvussa 5.2.
6. Viimeisessä vaiheessa kaikki epäilyttävät prosessit ja kernelin (engl. kernel) moduulit tulisi ottaa talteen tarkempaa analyysiä varten.

### 3.3.2 Muistin kaappaaminen

Koska keskusmuisti on haihtuvaa muistia, sen sisältö täytyy kopioida levymuistille ennen kuin sitä voi tutkia. Luvussa 4.2 käsitellään keskusmuistin haihtuvuutta. Muistin kaappausta varten on kehitelty erilaisia työkaluja, mutta käytettävä työkalu kannattaa aina valita kohteen mukaan, sillä työkalu voi olla suunniteltu käytettäväksi vain tietyillä käyttöjärjestelmän versioilla. Virtuaalikoneita tutkiessa kannattaa ottaa huomioon virtualisointiympäristöä pyörittävän hypervisor-ohjelmiston tarjoamat ominaisuudet, kuten ympäristön laittaminen tauolle tai tilannekuvan ottaminen. Muistia kaappaavan tutkijan täytyykin osata valita oikeat työkalut sekä käyttää työkaluja oikein, tai hän saattaa haluamattaan tuhota todistusaineistoa. (Ligh, Case, Levy & Walters 2014, 69-70, 82.)

Keskusmuistin haihtuvaa dataa voi myös olla tallennettuna levymuistille horros- tai sivutustiedostoina sekä kaatumisvedoksena. Sivutustiedostoa käsitellään luvussa 4.2. Ennen kuin tietokone menee horrostilaan, sen keskusmuistin sisältö kopioidaan horrostiedostoon. Kun järjestelmä palautuu

horrostilasta, horrostiedoston sisältö kopioidaan takaisin keskusmuistiin, jolloin ympäristö palautuu samanlaiseksi kuin hetkeä ennen järjestelmän horrostilaan asettumista. Windows-järjestelmissä horrostiedoston nimi on *Hiberfil.sys*, eikä käyttöjärjestelmä voi mennä horrostilaan, jos kyseistä tiedostoa ei löydy. Kaatumisvedos syntyy järjestelmän kaatuessa, jolloin käyttöjärjestelmä kirjoittaa keskusmuistista tiedostoon kaatumiseen liittyvää dataa, jota tutkimalla voi selvitä järjestelmän kaatumisen syy. Jos järjestelmä kaatui haittaohjelmaan liittyvän prosessin takia, vedos voi tarjota hyödyllistä tietoa haittaohjelmasta. (How to disable and re-enable hibernation on a computer that is running Windows 2020; Ligh ym. 2014, 71; Marcho 2019.)

# 4 Tietokoneen muisti

Muistia käytetään sovellusten ja datan tallentamiseen, jota mikroprosessori käyttää erilaisten tehtävien suorittamiseen. Muisti voi olla joko luku- tai keskusmuistia, ja tiedon tyyppi määrittelee, kumman tyyppiseen muistiin se varastoidaan. Prosessoreiden kehittyessä myös muistiteknologiat ovat kehittyneet, jotta näiden komponenttien välille ei pääse syntymään pullonkaulaa, joka rajoittaisi prosessorin toimintaa. (Gupta, Arora & Westcott 2017, 343-344; Null & Lobur 2015, Types of Memory.)

## 4.1 Lukumuisti

Lukumuistia (engl. Read Only-Memory) (ROM) käytetään yleensä säilytyspaikkana järjestelmän vakituiselle datalle, jonka sisältöä tarvitsee harvoin tai ei koskaan muuttaa. Esimerkiksi monista emolevyistä löytyvä BIOS-laiteohjelmisto on tallennettu ROM-muistiin. Lukumuisti on haihtumatonta, eli siihen taltioitu data ei katoa virran katketessa. ROM-muisteja on useita erityyppisiä, kuten EPROM (engl. Erasable Programmable ROM), EEPROM (engl. Electrically Erasable Programmable ROM) ja flash-muisti. EPROM-muistin voi uudelleenohjelmoida tyhjentämällä muistin ensin ultraviolettivaloa käyttävällä työkalulla. Kehittyneempi EEPROM-muisti voidaan tyhjentää sähkökenttää hyödyntäen tavu kerrallaan. Flash-muisti on ominaisuuksiltaan samanlaista kuin EEPROM, mutta dataa voidaan poistaa tai kirjoittaa suurempi lohko kerrallaan. (Gupta ym. 2017, 344; Null & Lobur 2015, A Look Inside a Computer, Types of Memory.)

## 4.2 Keskusmuisti

Keskusmuisti (engl. Random Access Memory) (RAM) on nopeaa muistia, jossa sijaitsevat suorituksessa olevat ohjelmat, kuten käyttöjärjestelmä ja sen data. Keskusmuisti kuuluu haihtuviin muisteihin, jolloin sähkövirran katketessa kaikki keskusmuistin sisältämä data katoaa, joten keskusmuistia ei käytetä vakituisena sijoituspaikkana datalle. (Gupta ym. 2017, 344; Null & Lobur 2015, Types of Memory.)

**Virtuaalimuisti**

Virtuaalimuisti laajentaa keskusmuistin kapasiteettia lisäämällä massamuistin keskusmuistin jatkeeksi. Keskusmuistin kapasiteetti ei välttämättä riitä kaikille prosesseille, jolloin osia prosessista voidaan väliaikaisesti säilöä hitaammassa massamuistissa. Jos virtuaaliseen muistiin siirretystä lohkosta haetaan dataa, muistihallintayksikkö (engl. Memory Management Unit) (MMU) ja muistinhallitsija siirtävät sen takaisin keskusmuistiin, eli fyysiseen muistiin. Kaikille prosesseille jaetaan omat virtuaaliset muistialueensa, joiden koko riippuu käyttöjärjestelmästä ja laitteen komponenteista. (Ligh ym. 2014, 20; Null & Lobur 2015, Virtual Memory.)

Virtuaalimuistin toteuttamiseen on useita eri tekniikoita. Null ja Lobur (2015) kertovat käytetyimmän tekniikan olevan sivutus (engl. paging). Sivutuksessa keskusmuisti ja prosessit jaetaan pieniin samankokoisiin lohkoihin. Prosessien lohkoja kutsutaan sivuiksi (engl. page) ja keskusmuistin lohkoja sivukehyksiksi (engl. page frame). Sivuja voidaan kopioida virtuaalimuistista sivukehyksiin, kun sivussa olevaa dataa halutaan prosessoida. Sivukehykseen siirretty data voidaan tämän jälkeen poistaa, kunnes sitä tarvitaan uudelleen. Näin prosessien osia, joita ei sillä hetkellä käytetä, voidaan säilöä massamuistissa, jolloin keskusmuistiin mahtuu yhteensä enemmän prosesseja. Sivujen sijainti fyysisessä muistissa on tallennettu prosessikohtaisiin sivutauluihin (engl. page table), jotta järjestelmä löytää prosessien sivut niitä tarvittaessa. (Null & Lobur 2015, Virtual Memory.)

## 4.3 Prosessit

Prosessit ovat suorituksessa olevia ohjelmia keskusmuistissa. Ohjelmat sisältävät komentoja, joita prosessori alkaa suorittamaan, kun ohjelmasta luodaan prosessi. Prosesseja voi olla suorituksessa samanaikaisesti useita ja samasta ohjelmasta voidaan luoda monia eri prosesseja. Prosessit eivät

välttämättä ole kuitenkaan aina suorituksessa, vaan niillä on erilaisia tiloja, jotka on listattu taulukkoon 1. (Garg & Verma 2017, 47-50.)

Taulukko 1. Prosessin tilat (Garg & Verma 2017, 48-50)

| Tila | Tilan merkitys |
|---|---|
| Uusi | Prosessin tila, kun sitä ollaan vielä luomassa. |
| Valmiustila | Vastaluodun prosessin tila, jossa prosessi odottaa vielä suorittamista. |
| Suorituksessa | Suorituksessa olevan prosessin tila. |
| Odottaa | Prosessi on tässä tilassa, kun se odottaa suoritukseensa vaativien resurssien vapautumista. |
| Päättynyt | Tila, jossa prosessin suoritus on päättynyt ja se poistetaan keskusmuistista. |

### 4.3.1 Säikeet

Säikeet mahdollistavat useiden prosessin toimintojen suorittamisen samaan aikaan. Esimerkiksi yksi prosessin säie hoitaa raskaampaa operaatiota, kuten kuvan käsittelyä, ja toinen säie voi samaan aikaan ottaa vastaan käyttäjän komentoja, kuten prosessiin kuuluvan graafisen käyttöliittymän painikkeiden klikkailua. Näin käyttäjä pystyy käyttämään sovellusta samaan aikaan, kun kuvaa prosessoidaan. Yksisäikeinen prosessi ei pystyisi suorittamaan useita toimintoja samaan aikaan. Prosessin säikeet jakavat keskenään prosessin muistialueen ja resurssit. (Garg & Verma 2017, 57-59.)

### 4.3.2 Jaettu muisti

Moderneissa käyttöjärjestelmissä prosessit voivat jakaa muistialueen keskenään. Yksi prosessi luo jaettavan muistisegmentin ja muut prosessit pääsevät alueeseen käsiksi liittämällä sen omaan osoiteavaruuteensa. Prosessit voivat käyttää jaettua muistialuetta jakaakseen tietoa keskenään tai

säästääkseen keskusmuistia luomalla datalle jaetun alueen. Tällöin identtistä dataa tarvitsevien prosessien ei tarvitse luoda omalle muistialueellensa erikseen segmenttiä kyseiselle datalle, vaan ne voivat hyödyntää jaetun muistialueen dataa. (Garg & Verma 2017, 85-87; Ligh ym. 2014, 22-23.)

# 5 Haittaohjelmat

Haittaohjelmia rakennetaan eri tarkoituksiin. Joidenkin haittaohjelmien tehtävä on tuhota tai salata järjestelmän dataa, mikä tekee ohjelmasta helposti huomattavan. Toiset haittaohjelmat taas saattavat vakoilla järjestelmää ja lähettää dataa hyökkääjän etäpalvelimelle. Jotta kyseinen haittaohjelma voi jatkaa toimintaansa mahdollisimman pitkään, se pyrkii olemaan huomaamaton. Nykyisin haittaohjelmat ovatkin suunniteltu olemaan huomaamattomampia kuin aiemmin. Yleisesti haittaohjelmiksi voidaan laskea ohjelmat, jotka jollain tapaa vaarantavat järjestelmän luottamuksellisuuden, eheyden tai saatavuuden. (Souppaya & Scarfone 2013, 2-5.)

Haittaohjelmia on useita erityyppisiä, ja kyseisen tyypin määrittelevät ominaisuudet, joita haittaohjelmalta löytyy (Malware Trends 2016, 2-11). Tämän luvun aliluvuissa on kuvailtu joitain yleisistä haittaohjelmien tyypeistä. Haittaohjelmien käyttämiä teknisiä menetelmiä kuvaillaan tarkemmin luvun 7 aliluvuissa.

## 5.1 Virukset

Virusta käytetään usein käsitteenä kaikille haittaohjelmille, mutta viruksella tarkoitetaan tietynlaista haittaohjelmaa. Virukset ovat melkein aina suoritettavia tiedostoja, kuten exe- tai bat-päätteen omaavat tiedostot Windows-järjestelmässä. Virus-tyyppinen haittaohjelma levittää omaa haittakoodiaan muihin järjestelmän tiedostoihin, jotka suoritettaessa levittävät haittakoodia vielä uusiin tiedostoihin. Tällä tavoin virus pääsee leviämään järjestelmässä tiedostosta toiseen, ja voi alkaa haittaamaan järjestelmän normaaleja operaatioita. Virukset voivat myös suorittaa muitakin toimintoja, kuten poistaa tai varastaa tiedostojen sisältämää tietoa. (Virus n.d.)

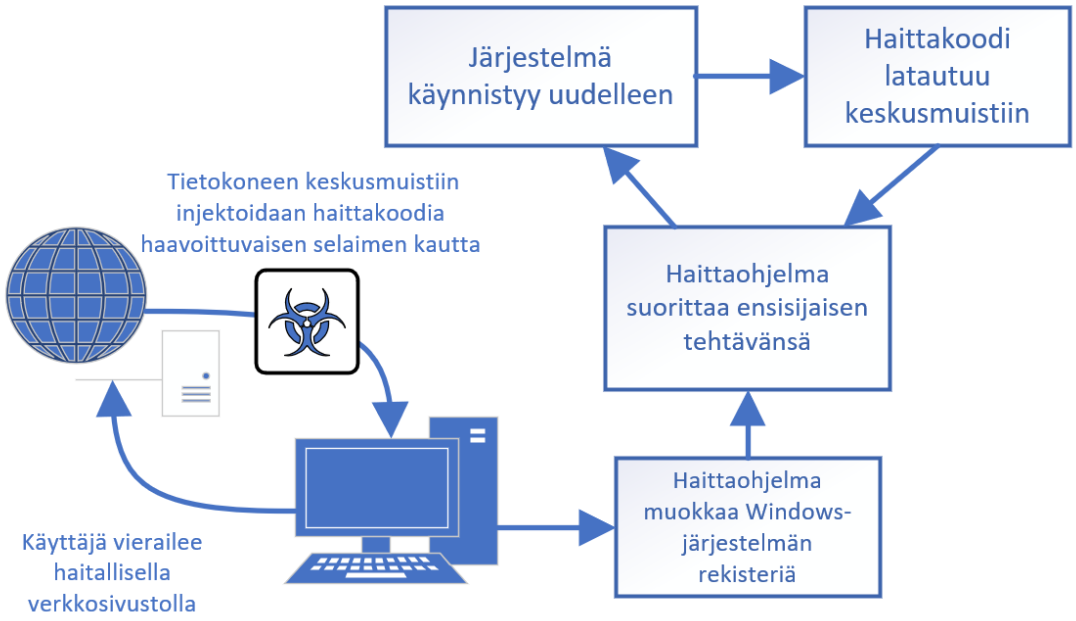
## 5.2 Piilohallintaohjelmat

Piilohallintaohjelmat (engl. rootkits) ovat vaikeasti havaittavia haittaohjelmia, joista suurin osa kykenee muokkaamaan kernelin toimintoja piilotellakseen omaa toimintaansa, kuten piilohallintaohjelman järjestelmään luomia haitallisia prosesseja tai verkkoyhteyksiä haittaohjelman etäpalvelimelle. Piilohallintaohjelmia on Nerenbergin (2007) mukaan kolmea erityyppistä: Kernel-, binääri- ja kirjasto-ohjelmia. Binääri-ohjelmat korvaavat järjestelmän sovelluksia omilla sovelluksillaan. Kernel-piilohallintaohjelmat asennetaan yleensä moduuleina kerneliin, joka mahdollistaa haittaohjelmalle pääsyn koko järjestelmään. Kirjasto-piilohallintaohjelmat muokkaavat kernelin käyttämiä kirjastoja saaden kernelin suorittamaan haittaohjelmalle suotuisia toimintoja, kuten piilottamaan prosesseja käyttäjän ja virustorjuntaohjelman näkyviltä. (Nerenberg 2007, 17-24; Sikorski & Honig 2012, 221-222.)

## 5.3 Tiedostottomat haittaohjelmat

Yksi vaikeasti havaittava haittaohjelman tyyppi on tiedostoton haittaohjelma (engl. fileless malware). Kyseiset haittaohjelmat sijaitsevat laitteen keskusmuistissa ja toimivat sieltä käsin. Ohjelma on kirjoitettu suoraan keskusmuistiin, eikä kiintolevylle, jolloin tietoturvaohjelmistojen on vaikeampi huomata sitä, toisin kuin perinteisiä kiintolevyllä sijaitsevia haittaohjelmia. Tiedostottomat haittaohjelmat eivät välttämättä kirjoita kiintolevylle mitään, vaan voivat käyttää esimerkiksi käyttöjärjestelmän omia työkaluja, kuten PowerShelliä Windows-järjestelmässä haittakoodin ajamiseen. Tietoturvaohjelmistot eivät välttämättä huomaa haittakoodia, koska PowerShellin suorittamista ei pidetä epäilyttävänä sen luotettavan digitaalisen allekirjoituksen takia ja PowerShell-prosessit ovat muutenkin yleisiä Windows-järjestelmissä. (Johansen n.d.)

Tiedostoton haittaohjelma voi levitä järjestelmään esimerkiksi käyttäjän vieraillessa selaimellaan verkkosivulla, joka käyttää selaimen haavoittuvaista lisäosaa injektoidakseen haitallista koodia selaimen muistiin. Koska haittaohjelma sijaitsee haihtuvassa muistissa, se poistuu järjestelmän sammuessa, kun keskusmuisti tyhjenee. Haittaohjelma voi kuitenkin pysyä järjestelmässä sammumisen jälkeen esimerkiksi muokkaamalla Windows-järjestelmän rekisteriä niin, että haittakoodi suoritetaan muistiin aina järjestelmän käynnistyessä. Kuviossa 3 on kuvattu tiedostottoman haittaohjelman tartunta ja toiminta Windows-käyttöjärjestelmässä aiemmin mainitulla tavalla. (Hyvärinen 2017; Johansen n.d.)




Kuvio 3. Tiedostottoman haittaohjelman eteneminen

Tiedostottoman haittaohjelman toimintaa voidaan tutkia esimerkiksi muistianalyysillä ottamalla muistivedos saastuneesta järjestelmästä luvussa 3.3.2 esitetyllä tavalla, ja analysoimalla kaapatun keskusmuistin sisältöä.

# 6 Volatility Framework

Volatility Framework sisältää keskusmuistin tutkimiseen tarkoitettuja työkaluja. Työkalut on kirjoitettu Python-ohjelmointikielellä, ja ne ovat avointa lähdekoodia, eli kaikille saatavilla. Lähdekoodia ylläpidetään Volatility Foundationin johdosta. Volatilityllä voi analysoida 32- ja 64-bittisten Windows-, Linux- ja Mac-käyttöjärjestelmien muistivedoksia. Volatilityllä ei ole graafista käyttöliittymää, vaan se on komentorivityökalu ja Python-kirjasto. (Ligh ym. 2014, 45-50.)

## 6.1 VTypes

Suurin osa käyttöjärjestelmistä on rakennettu C-kielellä. Volatility 2 käyttää omaa VTypes-kieltänsä C-kielen tietueiden (engl. struct) muuttamiseen Pythonille sopivaksi, jotta tietueita pystytään esittämään Python-kielen lähdekoodissa. VTypes-kielelle muutetut tietueet sisältävät Pythonin sanakirja- (engl. dictionary) ja lista- (engl. list) tietorakenteita sisäkkäin. VTypes-kielen automaattisesti

luomaa määrittelyä voi muokata Overlays-toiminnolla. Toiminnolla pystytään ohittamaan oletuksena asetettujen tietueen jäsenten datatyypit käyttäjän valitsemilla datatyypeillä. (Ligh ym. 2014, 51-54.)

## 6.2 Profiilit

Volatility 2 sisältää eri käyttöjärjestelmäversioiden profiileita. Aina muistivedosta tutkittaessa käyttäjän täytyy määritellä siihen käyttöjärjestelmään sopiva profiili, mistä muistivedos on otettu. Näin ohjelma tietää, mitä tietorakenteita, algoritmeja ja symboleita sen tulee käyttää muistivedosta käsitellessä. Profiilit on nimetty käyttöjärjestelmän versiota kuvaavasti, kuten esimerkiksi profiili *Win7SP1x64* tarkoittaa 64-bittistä Windows 7 Service pack 1 -käyttöjärjestelmäversiota. Volatility 2:sta löytyy myös työkalu nimeltään imageinfo, joka ehdottaa muistivedokselle sopivia profiileita. (Ligh ym. 2014, 55, 61-65.)

## 6.3 Käyttäminen

Volatilityä käytetään suorittamalla ohjelma komentorivillä ja antamalla sille parametrejä. Yksinkertaisimmillaan parametreiksi annetaan analysoitava muistivedos, muistivedokseen sopiva profiili (Paitsi imageinfo-lisäosaa käytettäessä) ja ajettava lisäosa (Ligh ym. 2014, 59-62). Volatility 2:en komento voisi näyttää esimerkiksi tältä Windows-järjestelmällä suoritettaessa:

```
volatility.exe -f muisti.dmp --profile=Win10x64_18362 psscan
```

Komennon eri osat selitettyinä (Ligh ym. 2014, 156-159; Command Reference 2020):

- **volatility.exe**: Suoritettavan ohjelman nimi.
- **-f muisti.dmp**: F-parametrillä osoitetaan analysoitava muistivedos muisti.dmp.
- **--profile=Win10x64_18362**: ”--profile”-parametrillä osoitetaan käytettävä profiili *Win10x64_18362*, eli 64-bittisen Windows 10 käyttöjärjestelmän käyttöliittymän versio 18362.
- **psscan**: Käytettävä lisäosa, joka osoitetaan ilman parametriä. psscan-lisäosa etsii muistista EPROCESS-objekteja, jotka ovat prosessien objekteja. Tulosteena lisäosa näyttää prosesseihin liittyviä tietoja, kuten prosessin sijainnin muistissa, prosessin nimen, prosessin ID:n (PID) ja äitiprosessin ID:n (engl. parent process identifier) (PPID).

Kuviossa 4 näkyy psscan-työkalun tulostamaa tietoa prosesseista. Winpmem_v3.3.rc3.exe-nimistä muistinkaappaamiseen käytettävää sovellusta ajettiin cmd.exe-komentotulkista, joten winpmem_v3.3.rc3.exe-prosessin äitiprosessi-ID on 184, mikä on cmd.exe-prosessin prosessi-ID (Command Reference 2020).

```
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Name                PID   PPID PDB                Time created
------------------ ---------------- ------ ------ ------------------ ------------------------------
0x00008f03c0eae080 svchost.exe        1964    660 0x0000000036c81002 2020-11-02 11:42:27 UTC+0000
0x00008f03c0eb3080 svchost.exe        1956    660 0x0000000036cd4002 2020-11-02 11:42:27 UTC+0000
0x00008f03c0edc080 svchost.exe        1836    660 0x000000003757b002 2020-11-02 11:42:26 UTC+0000
0x00008f03c0fd3040 Registry            104      4 0x0000000005a72002 2020-11-02 11:41:55 UTC+0000
0x00008f03c5a97080 NisSrv.exe          540    660 0x000000001d5ed002 2020-11-02 11:42:40 UTC+0000
0x00008f03c5b7b140 StartMenuExper     3128    796 0x000000000f491002 2020-11-02 11:42:54 UTC+0000
0x00008f03c5cbb080 svchost.exe         428    660 0x000000001626c002 2020-11-02 11:42:49 UTC+0000
0x00008f03c5f71340 wininit.exe         516    428 0x000000002b886002 2020-11-02 11:42:24 UTC+0000
0x00008f03c5f9e340 winlogon.exe        620    508 0x000000002c8c0002 2020-11-02 11:42:24 UTC+0000
0x00008f03c5fcc080 lsass.exe           688    516 0x000000002f811002 2020-11-02 11:42:25 UTC+0000
0x00008f03c6c26080 svchost.exe         796    660 0x0000000034881002 2020-11-02 11:42:25 UTC+0000
0x00008f03c6dde0c0 svchost.exe        1028    660 0x00000000316b6002 2020-11-02 11:42:26 UTC+0000
0x00008f03c6fdd080 svchost.exe        1596    660 0x00000000353c8002 2020-11-02 11:42:26 UTC+0000
0x00008f03c708e080 svchost.exe        1992    660 0x0000000039230002 2020-11-02 11:42:27 UTC+0000
0x00008f03c7092080 winpmem_v3.3.r     5656    184 0x0000000038ee7002 2020-11-02 11:45:12 UTC+0000
0x00008f03c71750c0 spoolsv.exe        2076    660 0x000000003c487002 2020-11-02 11:42:27 UTC+0000
0x00008f03c724c080 svchost.exe        2388    660 0x000000003bdb7002 2020-11-02 11:42:27 UTC+0000
0x00008f03c7317300 svchost.exe        2540    660 0x00000000398a2002 2020-11-02 11:42:27 UTC+0000
0x00008f03c77e6080 WindowsInterna     5288    796 0x000000003202a002 2020-11-02 11:43:14 UTC+0000
0x00008f03c77ef240 SearchUI.exe       4264    796 0x00000000380ba002 2020-11-02 11:42:56 UTC+0000
0x00008f03c78c4240 Windows.WARP.J     5004   4904 0x000000001bbc8002 2020-11-02 11:43:01 UTC+0000
0x00008f03c78ea080 ApplicationFra     4612    796 0x0000000025ac5002 2020-11-02 11:42:58 UTC+0000
0x00008f03c790f080 browser_broker     4912    796 0x0000000001e9a002 2020-11-02 11:43:01 UTC+0000
0x00008f03c7ae8080 SecurityHealth     5912   3856 0x000000001d7af002 2020-11-02 11:43:13 UTC+0000
0x00008f03c7aea080 RuntimeBroker.     5092    796 0x000000000db6d002 2020-11-02 11:43:01 UTC+0000
0x00008f03c7bd6080 cmd.exe             184   3856 0x000000001d015002 2020-11-02 11:43:26 UTC+0000
```

Kuvio 4. Volatilityn psscan-työkalun tulostetta

Jos jonkin prosessin suorittavaa tiedostoa haluaa tutkia tarkemmin, tiedoston voi ottaa talteen käyttämällä Volatility 2:en procdump-työkalua. Esimerkkikomento Windows-järjestelmällä:

```
volatility.exe -f muisti.dmp --profile=Win10x64_18362 procdump -p 184 -D
procdump/
```

Komennossa lisäosana käytetään nyt procdumpia. ”-p”-parametrin arvolla 184 kerrotaan työkalulle, että muistivedoksesta halutaan ottaa talteen prosessi-ID:n 184 tiedosto. Vaihtoehtoisesti voidaan käyttää ”--offset”-parametriä, jolle annetaan arvoksi prosessin cmd.exe osoite muistissa, joka on *0x00008f03c7bd6080* (ks. kuvio 4). ”-D”-parametrillä osoitetaan hakemisto, mihin tiedosto halutaan tallentaa, eli tässä tapauksessa procdump-hakemistoon. Kuviossa 5 näkyy työkalun antama

tuloste, joka kertoo prosessin suorittaneen tiedoston tallennuksen onnistuneen. (Ligh ym. 2014, 242-243; Command Reference 2020.)

```
Volatility Foundation Volatility Framework 2.6.1
Process(V)         ImageBase          Name                 Result
------------------ ------------------ -------------------- ------
0xffff8f03c7bd6080 0x00007ff6f4a80000 cmd.exe              OK: executable.184.exe
```

Kuvio 5. Volatilityn procdump-työkalun tuloste

Tiedostolle voisi tämän jälkeen tehdä vielä lisäanalyysia tarvittaessa. Osa luvun 7 aliluvuista sisältävät esimerkkejä eri Volatilityn työkalujen käytöstä.

### 6.4 Volatility 3

Volatilityn tämän hetken uusimmassa kolmannessa versiossa ei enää käytetä VTypes-kieltä ja profiileita kutsutaan nimellä SymbolSpace. Muutoksien taustalla on tarkoitus parantaa Volatility 3:sen käytettävyyttä käyttämällä VTypes-kielen sijaan Pythonin omia tietorakenteita ja vähentää profiileihin liittyviä ongelmia, kuten Windowsin 32-bittisten, 64-bittisessä järjestelmässä ajettujen ohjelmien analysoimista. Uusin versio on myös nopeampi vanhempaan verrattuna. (Volatility 3 Documentation 2021, 4-5, 13-14.)

## 7 Haittaohjelmat keskusmuistissa

Tämän luvun aliluvuissa tutustutaan eri tapoihin, joita haittaohjelmat käyttävät pysyvyytensä varmistamiseksi, ja keinoihin millä merkkejä haittaohjelmista voidaan hakea muistiforensiikan avulla.

### 7.1 Ympäristömuuttujat

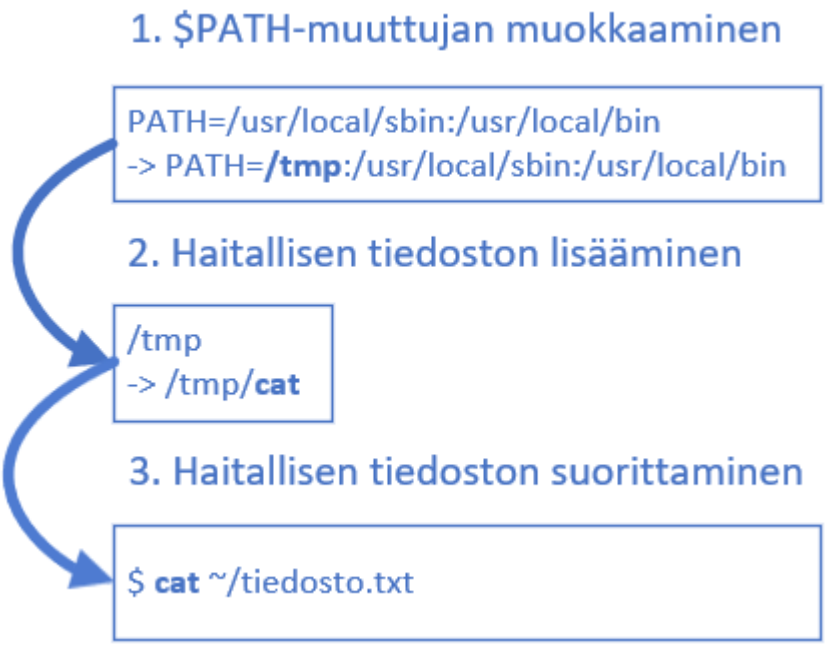
Käyttöjärjestelmistä löytyy ympäristömuuttujia, joihin on tallennettu erilaista järjestelmiin liittyvää tietoa. Ohjelmat ja skriptit voivat käyttää ympäristömuuttujia esimerkiksi kotihakemiston sijainnin hakemiseen Linux- ja Windows-käyttöjärjestelmillä. Ympäristömuuttujia pystyy myös itse lisäämään järjestelmiin. Linux-ympäristössä on paikallisia ja globaaleja ympäristömuuttujia. Globaalit muuttujat ovat saatavilla kaikille komentorivi-istunnoille ja prosesseille, mutta paikalliset ympäristömuuttujat vain niille istunnoille, joissa muuttujat on luotu. Käyttäjien on mahdollista luoda sekä

globaaleja että paikallisia ympäristömuuttujia. (About Environment Variables 2020; Blum & Bresnahan 2015, 135-139.)

Windows-käyttöjärjestelmissä ympäristömuuttujat voidaan jakaa kolmeen eri ryhmään: järjestelmä-, käyttäjä- ja prosessimuuttujat. Prosessimuuttujat ovat saatavilla vain nykyisessä prosessissa, ja lapsiprosessi perii ne aina äitiprosessilta. Käyttäjämuuttujat ovat käyttäjäkohtaisia, eli jokaisella käyttäjällä on omat muuttujansa, ja järjestelmämuuttujat ovat saatavilla kaikille käyttäjille sekä prosesseille. (About Environment Variables 2020.)

Windows- ja Linux-järjestelmistä löytyy $PATH-muuttuja, jonka arvona on polku suoritettaviin ohjelmiin. Muuttuja voi myös sisältää useamman polun eroteltuna toisistaan. Kun käyttäjä suorittaa ohjelmaa komentorivillä, $PATH-muuttujan sisältämistä poluista etsitään järjestyksessä käyttäjän suorittamaa tiedostoa. Esimerkiksi Linux-järjestelmässä $PATH-muuttuja voi näyttää tältä: *PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin*. Polut erotellaan toisistaan kaksoispisteillä ja ensimmäinen tarkistettava polku on */usr/local/sbin*, seuraava */usr/local/bin* ja niin edelleen. (Ligh ym. 2014, 226-228, 633-635.)

Haittaohjelmat saattavat luoda järjestelmään omia ympäristömuuttujiaan tai muokata jo olemassa olevia ympäristömuuttujia. Esimerkiksi $PATH-muuttujan arvon ensimmäiseksi poluksi voidaan lisätä uusi polku, jolloin komentorivillä ajettavaa ohjelmaa etsitään ensin sieltä. Kuvio 6 sisältää esimerkin Linux-järjestelmän $PATH-muuttujan manipuloinnista. Haittaohjelma liittää */tmp*-polun $PATH-muuttujan alkuun sekä lisää cat-nimisen haittakoodia sisältävän tiedoston */tmp*-hakemistoon. Täten aina kun käyttäjä suorittaa cat-työkalun komentorivillään, hän ajaakin hyökkääjän cat-tiedoston */tmp*-hakemistossa, koska */tmp*-hakemisto on $PATH-muuttujan ensimmäinen tarkistettava polku. (Ligh ym. 2014, 226-228, 633-635.)




Kuvio 6. $PATH-ympäristömuuttujan hyödyntäminen haittaohjelmassa

Volatilityn envars-lisäosa hakee muistivedoksesta prosessien ympäristömuuttujat ja tulostaa niihin liittyviä tietoja käyttäjälle, kuten prosessi-ID:n, prosessin nimen ja ympäristömuuttujan arvon (ks. kuvio 7). Ympäristömuuttujista voi myös päätellä tietoa, kuten tietokoneen nimen, käyttäjätunnuksen, prosessorin ytimien määrän tai muuta analyysin kannalta merkityksellistä tietoa. (Command Reference 2020.)

```
tuomo@Kala:~/volatility$ sudo python vol.py -f ../vedos.raw --profile=Win10x64_18362 envars
Volatility Foundation Volatility Framework 2.6.1
Pid      Process              Block              Variable                       Value
-------- -------------------- ------------------ ------------------------------ -----
     356 smss.exe             0x0000018c01c027f0 Path                           C:\Windows\System32
     356 smss.exe             0x0000018c01c027f0 SystemDrive                    C:
     356 smss.exe             0x0000018c01c027f0 SystemRoot                     C:\Windows
     436 csrss.exe            0x000001a3a0a027f0 ChocolateyInstall              C:\ProgramData\chocolatey
     436 csrss.exe            0x000001a3a0a027f0 ComSpec                        C:\Windows\system32\cmd.exe
     436 csrss.exe            0x000001a3a0a027f0 DriverData                     C:\Windows\System32\Drivers\DriverData
     436 csrss.exe            0x000001a3a0a027f0 FLARE_START                    C:\ProgramData\Microsoft\Windows\Start Menu\Programs\FLARE
     436 csrss.exe            0x000001a3a0a027f0 JAVA_HOME                      C:\Program Files\OpenJDK\openjdk-11u-11.0.4_11
     436 csrss.exe            0x000001a3a0a027f0 NUMBER_OF_PROCESSORS           4
     436 csrss.exe            0x000001a3a0a027f0 OS                             Windows_NT
```

Kuvio 7. Volatilityn envars-lisäosan käyttöä

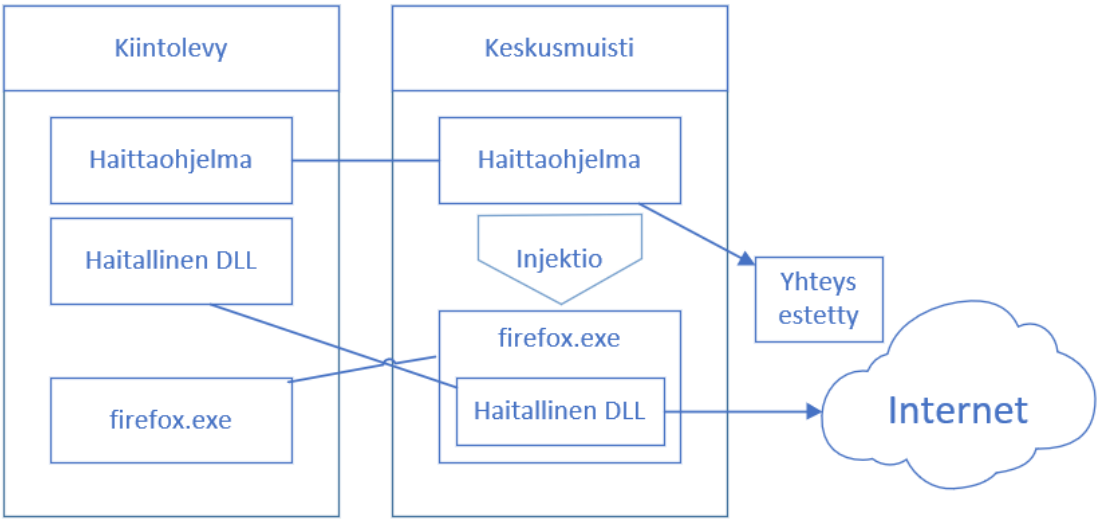
## 7.2 Koodin injektointi

Haittaohjelmat voivat injektoida omaa koodiaan muihin prosesseihin, saadakseen ajettua omaa koodiaan niiden muistialueilla. Näin prosessit saadaan suorittamaan haittaohjelmalle suotuisia toimintoja, ja haittaohjelma on vaikeampi poistaa kokonaan järjestelmästä sen levittyä useisiin prosesseihin. Injektoitu data voi sisältää esimerkiksi DLL- tai EXE-tiedoston (Executable file), jonka koodia prosessi pakotetaan suorittamaan. (Sikorski & Honig 2012, 254; Monnappa 2018, 293-294.)

### 7.2.1 DLL-injektiot

Windows-käyttöjärjestelmiin kohdistuvissa DLL-injektioissa prosessi pakotetaan lataamaan haitallinen DLL-tiedosto sen muistiin ja suorittamaan DLL:n DllMain-funktion, jota käyttöjärjestelmä kutsuu automaattisesti prosessin ladattua DLL:n. Tämän jälkeen prosessi suorittaa kyseisen haitallisen DLL-tiedoston koodia. (Sikorski & Honig 2012, 254; Monnappa 2018, 295.)

Kuviossa 8 on esitetty skenaario DLL-injektiosta. Haittaohjelma yrittää ottaa yhteyttä CnC-palvelimeen, mutta järjestelmän palomuuri estää yhteydenoton. Haittaohjelma injektoi firefox.exe-selaimen prosessiin DLL:n, jolloin haittaohjelma pystyy selainprosessin välityksellä ottamaan yhteyden CnC-palvelimeen, sillä tietokoneen palomuuri ei estä kyseisen selainprosessin yhteyksiä verkkoon. Tämä on yksi esimerkki haittaohjelmien mahdollisuudesta hyödyntää muiden prosessien oikeuksia. (Sikorski & Honig 2012, 254-255.)




Kuvio 8. DLL-injektio internet-yhteyden saavuttamiseksi (Sikorski & Honig 2012, 255, muokattu)

Tämäntyyppinen injektio toistaa yleensä tiettyä järjestystä (Hosseini 2017; Monnappa 2018, 295-297; Sikorski & Honig 2012, 254-257):

1. Haittaohjelma lataa tietokoneen kiintolevylle injektoitavan DLL-tiedoston.
2. Löytääkseen sopivan prosessin injektoitavaksi, haittaohjelma käyttää kolmea Windows-API:n (Application programming interface) funktiota, joiden avulla se listaa kaikki käynnissä olevat prosessit. Funktiot ovat CreateToolhelp32Snapshot, Process32First ja Process32Next.
3. Haittaohjelma valitsee sopivan prosessin ja ottaa sen prosessi-ID:n talteen. Haittaohjelma käyttää prosessin prosessi-ID:tä avatakseen siihen kahvan (engl. handle) Windows-API:n OpenProcess-funktiolla.
4. Kahvan avulla haittaohjelma varaa muistia DLL-tiedoston nimen merkkijonolle käyttäen VirtualAllocEx-funktiota. Muistialueen suojaukseksi valitaan yleensä PAGE_READWRITE, jolloin aluetta voidaan lukea ja sille voidaan kirjoittaa. VirtualAllocEx-funktio palauttaa varatun muistialueen muistiosoitteen.
5. Haittaohjelma kirjoittaa varatulle muistialueelle haitallisen DLL:n polun kiintolevyllä käyttäen WriteProcessMemory-funktiota. Tiedoston polku voi olla esimerkiksi *C:\Users\Tuomo\AppData\Local\Temp\tiedosto.dll*.
6. Haittaohjelma kutsuu GetModuleHandleA-funktiota parametrillä Kernel32.dll, mikä palauttaa haittaohjelmalle kyseisen kirjaston muistiosoitteen. GetProcessAddress-funktiota kutsutaan parametreinä Kernel32.dll:n muistiosoite ja LoadLibrary-funktion nimi. Funktio palauttaa LoadLibrary-funktion sijainnin muistissa.
7. Haittaohjelma luo säikeen prosessiin kutsumalla CreateRemoteThread-funktiota. Funktiolla annetaan parametreiksi muun muassa kahva injektoitavaan prosessiin, osoitin (engl. pointer) haitallisen DLL:n nimeen prosessissa ja LoadLibrary-funktion osoite.
8. Prosessiin luotu säie kutsuu LoadLibrary-funktiota, mikä lataa haitallisen DLL:n prosessin muistiin. Käyttöjärjestelmä kutsuu DLL:n DLLMain-funktiota automaattisesti, jolloin haittakoodia ajetaan prosessin sisällä.

Jotta haittaohjelman prosessi pystyy injektoimaan MIC:n (Mandatory Integrity Control) systemtason prosesseja, sillä täytyy olla käytössä SE_DEBUG_PRIVILEGE-oikeudet, mikä antaa prosessille luvan lukea ja kirjoittaa muiden prosessien muistialueita. SE_DEBUG_PRIVILEGE-oikeudet vaativat järjestelmänvalvojan (engl. Administrator) oikeuksia. (Ligh ym. 2014, 252; Monnappa 2018, 297.)

### 7.2.2 Suora injektio

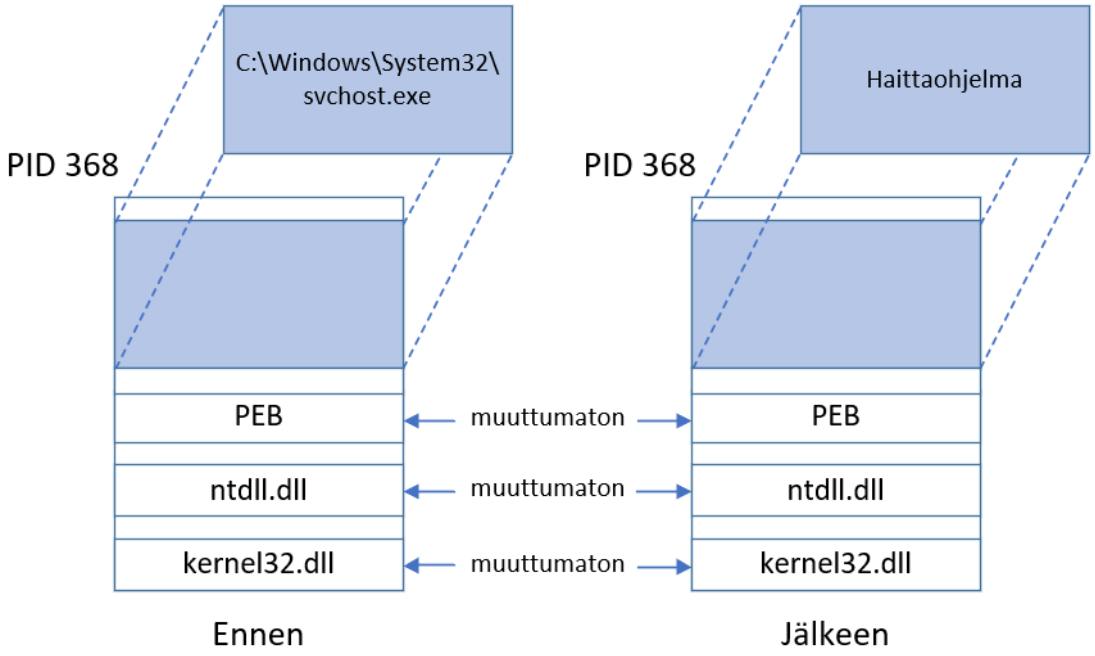
Suorassa injektiossa haittaohjelma kirjoittaa haittakoodin suoraan toisen prosessin sisälle. Tämän tyypin injektiossa haittaohjelman ei tarvitse kirjoittaa tietokoneen kiintolevylle mitään, toisin kuin luvun 7.2.1 DLL-injektiossa, jossa injektoitavan DLL:n piti löytyä tietokoneen kiintolevyltä ennen

sen injektointia prosessiin. Suorassa injektiossa prosessin muistialueelle voidaan kirjoittaa haitallista koodia tai suoritettava tiedosto. (Hosseini 2017; Monnappa 2018, 311; Sikorski & Honig 2012, 257.)

Suorassa injektiossa käytetään osittain samoja Windows-API:n funktioita kuin luvussa 7.2.1 kuvaillussa DLL-injektiossa. Injektoitavaan prosessiin avataan kahva OpenProcess-funktiolla, ja sen muistialueelle kirjoitetaan VirtualAllocEx-funktiolla. Muistialueen suojaustyypiksi on määritelty PAGE_EXECUTE_READWRITE, jotta muistialueen koodia voidaan myös suorittaa toisin kuin PAGE_READWRITEssä. Haittakoodi kirjoitetaan prosessin muistialueelle WriteProcessMemory-funktiolla, minkä jälkeen haittaohjelman prosessi kutsuu CreateRemoteThread-funktiota luodakseen säikeen injektoitavan prosessin haittakoodin muistialueelle. Säie alkaa suorittamaan prosessista löytyvää haittakoodia ja injektio on onnistunut. Säikeitä kuvailtiin tarkemmin luvussa 4.3.1. (Ligh ym. 2014, 253; Sikorski & Honig 2012, 257; Monnappa 2018, 311-313.)

### 7.2.3 Onton prosessin injektio

Onton prosessin injektiossa (engl. Hollow Process Injection) haittaohjelma aloittaa uuden prosessin jostain sovelluksesta, kuten Windows-käyttöjärjestelmään kuuluvasta svchost.exe-tiedostosta. Seuraavaksi haittaohjelma tyhjentää svchost.exe-prosessin koodiosion, jolloin koko prosessin muistialue jää osittain ontoksi. Lopuksi haittaohjelma korvaa tyhjennetyn muistialueen omalla haittakoodillaan. Vaikka prosessi sisältääkin nyt haittaohjelman koodia, useat prosessin tietorakenteet, kuten PEB (Process Environment Block) osoittavat yhä sijaintiin *C:\Windows\System32\svchost.exe*. PEB-rakenne sisältää tietoa, kuten polun prosessin suorittavaan tiedostoon, prosessin lataamien DLL-tiedostojen polut sekä ajetaanko prosessia virheenjäljittimessä (engl. debugger). Kuviossa 9 on kuvattu svchost.exe-prosessin muistialuetta ennen injektiota ja sen jälkeen. (Ligh ym. 2014, 220-222, 258; Monnappa 2018, 313-314, 401.)




Kuvio 9. Svchost.exe-prosessin koodin korvaaminen haittaohjelman koodilla (Ligh ym. 2014, 258, muokattu)

## 7.3 Kerneli

Kerneli on käyttöjärjestelmän ytimessä toimiva ohjelma, joka on vastuussa monista järjestelmän perustoiminnoista, kuten siirrännästä (engl. input / output) ja kovalevyjen hallinnasta. Koska kerneli ja sen toiminnot ovat niin olennaisessa osassa järjestelmää, sen muistialueen suojaamiseksi on kehitetty erilaisia tekniikoita. Yksi näistä on järjestelmän käyttäjien oikeudet. (Perla & Oldani 2011, Introducing the Kernel and the World of Kernel Exploitation.)

Windows-järjestelmistä löytyy pääkäyttäjä nimeltään Järjestelmänvalvoja ja UNIX-pohjaisista käyttöjärjestelmistä root. Pääkäyttäjät omaavat täydet oikeudet järjestelmään, joten nämä käyttäjät pystyvät myös muokkaamaan kerneliä. Näistä järjestelmistä löytyy myös tavallisia käyttäjiä, joiden oikeuksia muihin käyttäjiin ja koko systeemiin on rajattu. Käyttöjärjestelmien muisti on myös jaettu kernel-tason ja käyttäjätason muistiin. Kernel-tason muistissa ajettavalla ohjelmalla on vapaa pääsy kaikkialle keskusmuistissa, kun taas käyttäjätason muisti on rajoitetumpaa. (Perla & Oldani 2011, Introducing the Kernel and the World of Kernel Exploitation.)

### 7.3.1 Moduulit

Kernelin moduuleilla pystytään lisäämään kernelin ominaisuuksia. Moduuli voi olla esimerkiksi laitteistoajuri, jota kerneli tarvitsee tietokoneeseen liitetyn laitteen, kuten näppäimistön käyttämiseen. Moduuleita pystytään lisäämään esimerkiksi luomalla kernel-ajuri-tyyppinen palvelu SCM:n (Service Control Manager) kautta tai kutsumalla suoraan alemman tason Windows API:n NtLoadDriver-funktiota, jolloin SCM ohitetaan ja palvelulokiin ei jää merkintää tapahtumasta. (Ligh ym. 2014, 25, 370-371.)

Haittaohjelmatkin kykenevät lataamaan omia moduuleitaan kerneliin. Moduuleita suoritetaan kernel-tason muistissa, joten haitallinen moduuli pystyy vaikuttamaan suoraan käyttöjärjestelmän toiminnallisuuteen, jolloin se voi esimerkiksi piilottaa omista toiminnoistaan jääneitä jälkiä. Tämäntyyppistä havaitsemista välttävää kernel-tasossa ajettavaa haittaohjelmaa kutsutaan piilohallintaohjelmaksi. Luvussa 5.2 käsiteltiin piilohallintaohjelmia laajemmin. (Ligh ym. 2014, 367; Monnappa 2018, 432-433.) Volatility 2:sta löytyy useita lisäosia muistivedoksesta löytyvien moduulien analysoimiseen. Lisäosia on kuvailtu taulukkoon 2.

Taulukko 2. Volatility 2:en lisäosia kernelin moduulien tutkimiseen (Command reference 2020)

| Lisäosa | Kuvaus |
|---|---|
| modules | Listaa kerneliin ladatut moduulit käymällä LDR_DATA_TABLE_ENTRY-tietorakenteita läpi. Näyttää moduuleiden latausjärjestyksen. Ei löydä piilotettuja tai käytöstä poistettuja moduuleita. |
| modscan | Listaa kerneliin ladatut moduulit, joihin kuuluvat myös mahdollisesti piilohallintaohjelmien piilottamat tai käytöstä poistetut moduulit. Ei näytä moduulien latausjärjestystä, toisin kuin modules-lisäosa. |
| moddump | Tallentaa muistivedoksesta valitun kernel-moduulin tiedostoon. |
| driverscan | Etsii DRIVER_OBJECT-tietorakenteita muistivedoksesta ja listaa löydetyt ajurit. |
| unloadedmodules | Listaa äskettäin irrotetut moduulit. |

### 7.3.2 SSDT

SSDT (System Service Descriptor Table) on kernelin funktioiden kahvojen muistiosoitteita sisältävä tietorakenne Windows-käyttöjärjestelmässä. Kun käyttäjätason ohjelma esimerkiksi aloittaa uuden prosessin kutsumalla tarvittavaa kernelin funktiota, ohjelmalle haetaan SSDT:stä kyseisen funktion muistiosoite. Haittaohjelma voi muokata SSDT:tä ylikirjoittamalla siitä löytyvien funktioiden muistiosoitteita omalla muistiosoitteellaan, joka voi esimerkiksi osoittaa haitalliseen moduuliin. Näin korvattujen muistiosoitteiden funktioita hakevat ohjelmat saadaan suorittamaan haittakoodia. Tekniikka toimii 32-bittisissä Windows-järjestelmissä, mutta ei enää 64-bittisissä järjestelmissä, sillä niistä löytyvä PatchGuard-teknologia kaataa koko tietojärjestelmän, jos SSDT:tä on muokattu. (Ligh ym. 2014, 393-395; Matrosov, Rodionov & Bratus 2019, 25.)

## 7.4 Windows-rekisteri

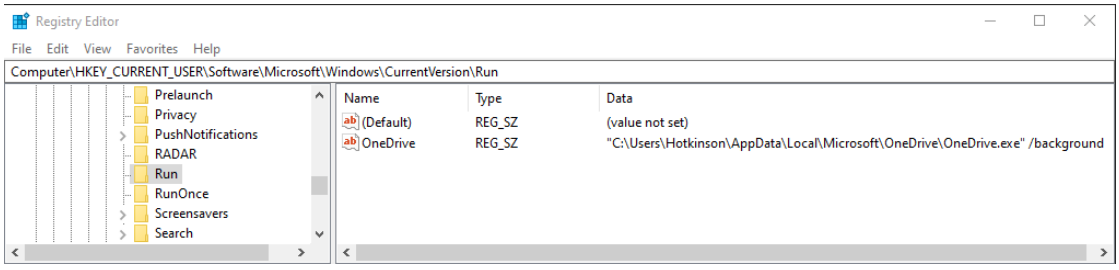
Yksi Windows-järjestelmän ydinkomponenteista on Windows-rekisteri. Se sisältää paljon asetuksia sekä tietoa järjestelmästä ja sen käytöstä. Rekisteristä voi esimerkiksi etsiä tietoa käyttäjien lataamista ohjelmista sekä käyttäjien salasanojen tarkistussummia (engl. hash). Koska rekisteri on tärkeä osa Windows-käyttöjärjestelmää, sitä käytetään jatkuvasti tietokoneen ollessa päällä. Tämän takia osa tai kaikki rekisteritiedostojen sisällöistä on tallennettu tietokoneen muistiin, jotta tietojen hakeminen olisi nopeampaa. (Carvey 2016, 1-2; Ligh ym. 2014, 281.)

Windows-rekisteri on jaettu viiden juuriavaimen (engl. root key) alle. Taulukko 3 sisältää kuvauksen jokaisesta juuriavaimesta. Juuriavaimet sisältävät avaimia (hakemistoja), joiden alla on lisää avaimia sekä merkintöjä (engl. value entry), jotka koostuvat yksittäisen nimen ja arvon parista. Merkintöjä voi verrata ohjelmoinnista tuttuun muuttuja-käsitteeseen. Merkinnät sisältävät kaikki rekisterin arvot. (Sikorski & Honig 2012, 139-140.)

Taulukko 3. Windows-rekisterin juuriavainten yleiskuvaus (Russinovich, Solomon & Ionescu 2012, 280-287)

| Juuriavain | Kuvaus |
|---|---|
| HKEY_LOCAL_MACHINE (HKLM) | Sisältää kaikkien tietokoneen käyttäjien asetuksia. |
| HKEY_CURRENT_USER (HKCU) | Sisältää sisään kirjautuneen käyttäjän profiiliasetuksia. |
| HKEY_CLASSES_ROOT | Sisältää tiedostopäätteisiin liittyvää tietoa. |
| HKEY_CURRENT_CONFIG | Sisältää järjestelmän laitteistoprofiiliasetukset. |
| HKEY_USERS | Sisältää kaikkien käyttäjäprofiilien asetuksia. |
| HKEY_PERFORMANCE_DATA | Sisältää järjestelmän suorituskykyyn liittyvää tietoa. |

Windows-rekisteri tarjoaa haittaohjelmille erilaisia mahdollisuuksia. Rekisteri sisältää avaimia, joiden merkinnät määrittelevät, mitkä ohjelmat käynnistetään tietokoneen käynnistyksen yhteydessä tai käyttäjän kirjautuessa sisään. Yksi tällainen aliavain on *HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*. Kuviossa 10 on esimerkki kyseisen avaimen merkinnöistä. Ainoastaan OneDrive.exe-niminen EXE-tiedosto suoritetaan käyttäjän kirjautuessa sisään. Haittaohjelma voi lisätä kyseiseen tai vastaaviin rekisteriin merkintöjä, jolloin haittaohjelman komponentteja suoritetaan tietokoneen käynnistyksen yhteydessä. (Ligh ym. 2014, 289-292; Sikorski & Honig, 140-143.)



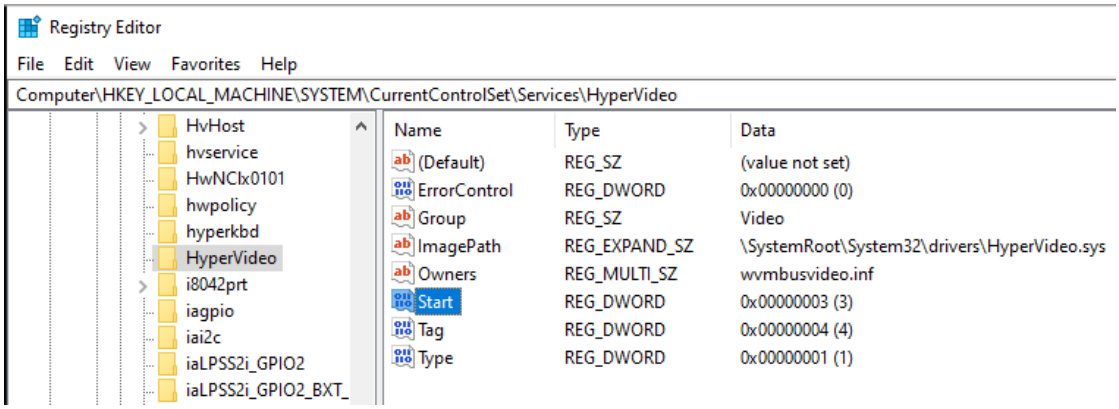

Kuvio 10. Sisäänkirjautuessa käynnistettävät ohjelmat Windows-rekisterissä

## 7.5 Windows-palvelut

Windows-palvelut ovat sovelluksia, jotka toimivat taustalla järjestelmän ollessa käynnissä. Yleensä palvelut eivät tarjoa minkäänlaista käyttöliittymää käyttäjälle, jolloin käyttäjä ei pysty vuorovaikuttamaan palveluihin muuten kuin pysäyttämällä ja käynnistämällä niitä palveluiden hallintapaneelista. Palveluita ovat esimerkiksi aikapalveluprosessi (engl. time daemon) ja Windows Defender. (Carvey 2016, 81-83; Ligh ym. 2014, 343-345.)

Kaikilla Windows-järjestelmän palveluilla on oma aliavain rekisterissä *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services*-avaimen alla. Näiden aliavainten merkinnät määrittelevät palveluihin liittyviä asetuksia, kuten käynnistetäänkö se automaattisesti järjestelmän käynnistyessä vai manuaalisesti jonkun toisen sovelluksen toimesta. (Ligh ym. 2014, 343-345.)

Windows-järjestelmän käynnistyessä SCM, eli services.exe-niminen prosessi käynnistää järjestelmän palvelut. Prosessi lukee palveluiden aliavainten merkinnöistä Start-nimisen merkinnän arvon, joka määrittelee, käynnistetäänkö palvelu järjestelmän käynnistyksen yhteydessä. Kuviossa 11 näkyy HyperVideo-palvelun rekisterimerkintöjä. Start-merkinnän arvo on 3, mikä tarkoittaa, että palvelu pitää käynnistää manuaalisesti. Arvo 2 tarkoittaisi automaattista käynnistystä. (Monnappa 2018, 277-281.)




Kuvio 11. HyperVideo-palvelun rekisterimerkinnät

### 7.5.1 Haittaohjelmien luomat palvelut

Myös haittaohjelmat voivat lisätä omia palveluitaan Windows-järjestelmiin. Palveluita voi lisätä useilla tavoilla, joista yleisimmät ovat Windows API -funktiot, komentojonot (engl. batch script) ja palvelun lisääminen manuaalisesti. Palveluihin liittyviä Windows API -funktioita ovat muun muassa CreateService, joka lisää SCM:ään uuden palvelun sekä valitun palvelun käynnistävä StartService. Manuaalisesti komentoriviltä käsin palveluita voi lisätä *sc create* -komennolla ja käynnistää *sc start* -komennolla. Sc-komennot käyttävät Windows API -funktioita palveluiden käsittelemiseen. Komentojonotiedostoihin voidaan sisältää tarvittavat komennot, jotka luovat palvelun ja lisäävät rekisteriin palvelun avaimet ja merkinnät, kun komentojonotiedosto suoritetaan. (Ligh ym. 2014, 345; Monnappa 2018, 279-280.)

Windows-palvelut voivat olla erityyppisiä. Haittaohjelmien käytetyimmät palvelutyypit ovat: jaettu prosessi (.dll), itsenäinen prosessi (.exe) ja kernel-ajuri (.sys). Itsenäisten prosessien taustalla on suoritettava tiedosto, jonka SCM suorittaa. Kernel-ajurilla kernelin muistialueella voidaan ajaa koodia. Jaetut prosessit ovat svchost.exe-nimisiä tiedostoja, joihin on ladattu palvelun DLL-tiedosto. (Ligh ym. 2014, 343-347; Monnappa 2018, 277-279; ServiceType Enum n.d.)

Jotkut tietoturvaohjelmistot saattavat tarkkailla prosesseja, jotka esimerkiksi kutsuvat tiettyjä Windows API -funktioita putkeen luodakseen uuden palvelun. Palveluiden luonti näillä funktioilla luo tapahtumalokiin merkintöjä, jotka voivat paljastaa haittaohjelman. Haittaohjelmat voivatkin käyttää hiljaisempia metodeja karttaakseen tietoturvaohjelmistojen heuristiikkaa, kuten luomalla rekisterimerkinnät manuaalisesti ja kutsumalla funktioita, joiden jäljiltä ei luoda tapahtumalokiin merkintöjä. (Ligh ym. 2014, 346.)

### 7.5.2 Palvelun kaappaaminen

Haittaohjelmat kykenevät kaappaamaan järjestelmän muita palveluja. Jotkin haittaohjelmat kohdistavat kaappauksen Windowsin omaan palveluun, jota käytetään harvoin, mutta se toimii taustalla järjestelmän ollessa käynnissä. Tällöin järjestelmään ei luoda uutta palvelua ollenkaan, vaan olemassa olevaa palvelua muokataan ajamaan haittaohjelman koodia. Tämän kaltaisen palvelun korvaaminen ei aiheuta ongelmia järjestelmässä, koska palvelua käytetään erittäin harvoin tai ei koskaan. (Ligh ym. 2014, 356; Monnappa 2018, 281.)

Kaappaaminen voidaan toteuttaa kahdella eri tavalla. Ensimmäinen tapa on muokata palvelun rekisterimerkintöjä. Esimerkiksi jaetun prosessin DLL-tiedoston polun merkintää (ServiceDll) voidaan muuttaa rekisteristä niin, että haittaohjelma määrittelee sen osoittamaan omaa DLL-komponenttiaan. Seuraavan kerran kun palvelu käynnistetään, haittaohjelman DLL-tiedostoa ajetaan palveluna alkuperäisen DLL-tiedoston sijaan. (Ligh ym. 2014, 356-357.)

Toinen tapa on korvata tiedosto, josta palvelu on luotu. Alkuperäinen tiedosto johon palvelun rekisterimerkintä osoittaa, voidaan siirtää muualle tai poistaa, minkä jälkeen samanniminen haittakoodia sisältävä tiedosto lisätään tilalle. Tämä tapa ei vaadi palvelun rekisterimerkintöjen, kuten jaetun prosessin ServiceDll-merkinnän arvon vaihtamista. Vain alkuperäinen tiedosto korvataan toisella tiedostolla. (Ligh ym. 2014, 358-362; Monnappa 2018, 281.)

## 7.6 Verkkotapahtumat

Suurimmalta osalta haittaohjelmista löytyy verkkoyhteyksiin liittyvää toiminnallisuutta. Haittaohjelmat voivat käyttää verkkotoimintojansa muun muassa ottaakseen yhteyttä CnC-palvelimeen (Command-and-Control) tai levitäkseen verkkoja pitkin muihin laitteisiin. Toiminnallisuus riippuu haittaohjelman tyypistä ja tarkoituksesta. CnC-yhteyksien kautta voidaan viestiä käyttäen erilaisia protokollia, kuten IRC:iä tai haittaohjelman omaa protokollaa. (Ligh ym. 2014, 309-313.)

Haittaohjelmat voivat myös avata tietokoneeseen takaportteja ohittaen tietoturvakontrollit. Vaikka haittaohjelma menettäisi komponenttejaan tietokoneesta, se voi ladata niitä uudelleen huomaamattomasti takaporttien kautta. Takaporteista pyritäänkin tekemään mahdollisimman huomaamattomia, koska takaportin löytyminen voisi johtaa sen menettämiseen. Tämän takia haittaohjelmat yrittävät piilottaa yhteyksiään käyttämällä erilaisia tekniikoita. Eräs yleinen tapa on käyttää suosittuja kommunikaatioprotokollia, kuten HTTP:tä (Hypertext Transfer Protocol), HTTPS:ää (Hypertext Transfer Protocol Secure) tai DNS:ää (Domain Name System). Kyseisten protokollien liikenne sekoittuu helposti massaan, mikä tekee haittaliikenteen huomaamisesta vaikeampaa. (Sikorski & Honig, 140-143.)

DNS-protokollaa voidaan hyödyntää esimerkiksi lähettämällä saastuneesta järjestelmästä DNS-kyselyitä, joiden aliverkkotunnukseksi on lisätty dataa, kuten osa tiedostoa tai viesti CnC-palvelimelle. Dataa ei lähetetä selkotekstisenä, vaan se voidaan esimerkiksi salata, kompressoida ja

muuttaa koodimuotoon käyttäen Base64-koodausta, jolloin kaikki merkit lähetettävässä tiedossa ovat aliverkkotunnuksen nimelle sopivia (Hyväksyttävät merkit ovat a-z, A-Z, 0-9 sekä viivat ja pisteet). CnC-palvelin voi purkaa aliverkkotunnukseen kätketyn datan DNS-kyselyn tultua perille. Tekniikkaa käyttämällä dataa voidaan lähettää CnC-palvelimelle pienissä erissä. (Das, Shen, Shashanka & Wang 2017, 738.)

## 7.7 Master File Table

NTFS-tiedostojärjestelmän osio sisältää tiedoston, joka on nimeltään Master File Table (MFT). MFT pitää sisällään merkinnät kaikille osiossa oleville tiedostoille ja hakemistoille. Merkinnät sisältävät tietoa, kuten tiedoston tai hakemiston sijainnin tiedostojärjestelmässä, aikaleimoja, datan tyypin ja nimen. MFT kasvaa sitä mukaan kuin dataa lisätään tiedostojärjestelmään ja poistetun datan merkintöjen viemä tila varataan uusille merkinnöille. MFT:n tärkeyden takia NTFS varaa 12,5 prosenttia osion kokonaistilasta MFT:lle, kunnes jäljelle jäänyt tila on käytetty loppuun. (How NTFS reserves space for its Master File Table (MFT) 2020; Ligh ym. 2014, 477-481.)

Volatility sisältää mftparser-lisäosan, joka etsii MFT:n merkintöjä muistista ja tulostaa merkinnöissä olevien attribuuttien tietoja komentoriville. Liitteessä 1 näkyy esimerkki lisäosan käytöstä Pythonilla. Lisäosalle voidaan myös määritellä vaihtoehtoisia parametrejä, kuten muistiosoite, josta yksittäinen MFT-merkintä haetaan. (Command reference 2020.)

**Alternate Data Stream**

Yksi NTFS:n tiedostokohtaisista ominaisuuksista on vaihtoehtoinen datavirta (engl. alternate data stream) (ADS). Ominaisuus lisättiin alun perin yrityksenä tukea kolmannen osapuolen, kuten Applen tiedostojärjestelmiä Windowsilla. Käytännössä ADS:n avulla tiedostoon voidaan kirjoittaa vaihtoehtoinen sisältö, mikä ei näy käyttäjälle käyttöliittymän tiedostolistauksissa, ellei piilotettua sisältöä erikseen haeta. (Messier & Mackay 2016, Alternate Data Streams.)

Liite 2 sisältää esimerkin datan lisäämisestä PowerShell-tiedoston ADS:ään Windows 10 -käyttöjärjestelmän komentorivillä. Ensimmäiseksi ajettavalla dir-komennolla saadaan tulostettua lista tämänhetkisen hakemiston sisällöstä ja ”/r”-parametri komennossa paljastaa vaihtoehtoiset datavirrat, jos niitä on (Messier & Mackay 2016, Alternate Data Streams). Hakemisto sisältää vain skripti.ps1-nimisen tiedoston. Seuraavana liitteessä ajetaan komento:

```
echo ADS > skripti.ps1:datavirta
```

Komento lisää datavirta-nimisen datavirran skripti.ps1-tiedostoon. Datavirran nimi on vain esimerkki ja sen pystyy nimeämään haluamallaan tavalla. Kaksoispiste erottaa datavirran nimen tiedostonimestä. Datavirran sisältöön lisätään merkkijono ”ADS”. Seuraavaksi liitteessä 2 ajetaan dir-komento kahdesti. Ensin ilman ”/r”-parametriä ja toisella kerralla sen kanssa. Ensimmäisenä suoritetun komennon tulosteesta voidaan huomata, että datavirtaa ei näy tulosteessa. Toisena suoritetun komennon tulosteessa taas äsken luotu datavirta näkyy, eli tiedostojen vaihtoehtoisia datavirtoja saa erikseen hakea, sillä niitä ei näy oletuksena. (Messier & Mackay 2016, Alternate Data Streams.)

Vaihtoehtoisiin datavirtoihin voidaan lisätä kokonaisia tiedostoja ja jotkut haittaohjelmat hyödyntävätkin datavirtoja komponenttiensa piilottamiseen. Microsoft onkin vaikeuttanut ohjelmien suorittamista suoraan vaihtoehtoisista datavirroista käyttäjien suojelemiseksi. (Ligh ym. 2014, 482-484; Messier & Mackay 2016, Alternate Data Streams.)

# 8 Toteutuksen suunnitelma

Ennen varsinaisen teknisen toteutuksen aloittamista, toimeksiantajan kanssa suunniteltiin teknologioita, ominaisuuksia ja toimintoja, joita teknisestä toteutuksesta toivottaisiin löytyvän. Valittujen teknologioiden pohjalta toteutettiin yleisluontoinen suunnitelma toteutuksen olennaisimmista ominaisuuksista.

## 8.1 Käytettäviä teknologioita

Osana teknistä toteutusta käytetään useita eri teknologioita, joista oleellisimmat on listattu tämän luvun aliluvuissa sekä perustelut miksi kyseiset teknologiat valittiin.

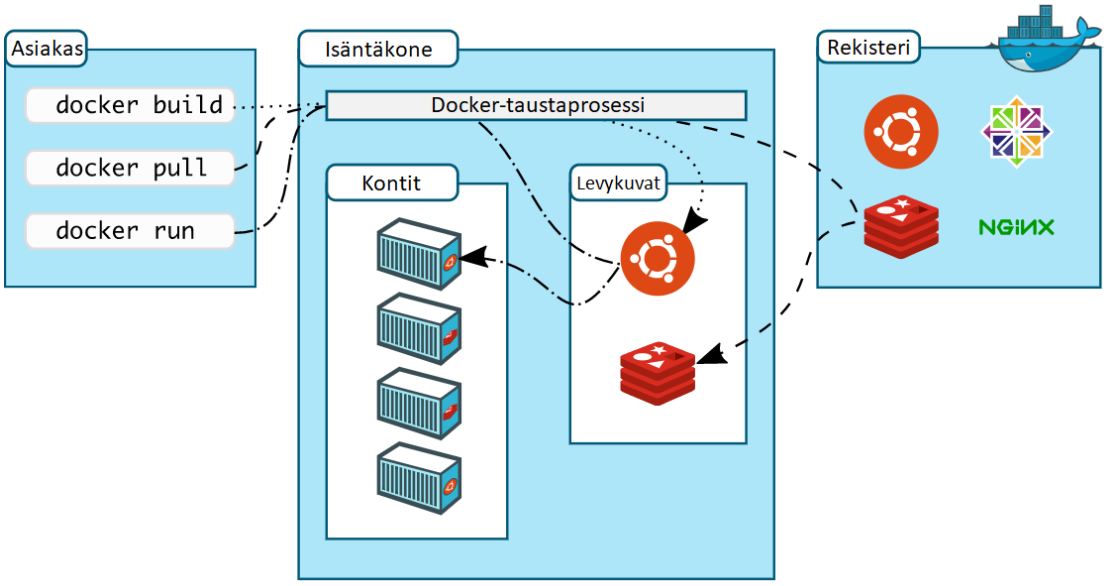
### 8.1.1 Docker

Docker on Go-ohjelmointikielellä kirjoitettu avoimen lähdekoodin ympäristö, johon käyttäjä voi pystyttää omia palveluitaan käyttämällä kontteja. Kontit ovat kevyitä virtualisoituja ympäristöjä, jotka simuloivat aitoa käyttöjärjestelmää. Kontteja hallitaan komentoriviltä tai API:sta käsin, joista

konteille voidaan toteuttaa erilaisia toimintoja, kuten niiden käynnistäminen, sammuttaminen tai poistaminen. (Docker overview n.d.)

Kontteja käynnistetään levykuvista (engl. image), jotka sisältävät valmiiksi esimerkiksi Ubuntu-käyttöjärjestelmän ja Apache-palvelinohjelman. Palveluita ja konfiguraatioita voidaan lisätä kyseiseen levykuvaan, jolloin siitä voidaan luoda uusi päivitetympi versio. Päivitetyn levykuvan pohjalta voidaan käynnistää uusi kontti, joka sisältää uudet palvelut ja konfiguraatiot. Levykuvia voidaan säilyttää Dockerin omassa julkisessa rekisteripalvelussa tai paikallisessa rekisterissä, jossa levykuvat ovat helposti saatavilla. (Docker overview n.d.)

Docker perustuu asiakas-palvelin-arkkitehtuuriin, jota luonnehditaan kuviossa 12. Asiakas-laatikon sisällä olevat palkit sisältävät Docker-komentoja, joilla asiakas on yhteydessä isäntäkoneella pyörivään Docker-taustaprosessiin. Asiakas voi syöttää komentoja suoraan isäntäkoneelta käsin tai etänä. Kuviossa komennoista lähtevät erilaiset viivat osoittavat, mihin Dockerin komponentteihin mikäkin komento taustaprosessin kautta vaikuttaa. (Docker overview n.d.)




Kuvio 12. Docker-arkkitehtuuri (Docker overview n.d, muokattu)

Kuviossa 12 näkyvät komennot selitettyinä (Docker overview n.d):

- **docker build**: Rakentaa levykuvan ajamalla Dockerfilestä löytyviä käskyjä.
- **docker pull**: Hakee levykuvan rekisteristä isäntäkoneelle.
- **docker run**: Hakee levykuvan rekisteristä isäntäkoneelle, jos se ei jo ole siellä sekä luo kontin käyttäen kyseistä levykuvaa.

Konttien asetuksia pystytään muokkaamaan laaja-alaisesti. Konteille voidaan esimerkiksi määritellä verkot, joihin ne saavat olla yhteydessä sekä kuinka paljon kiintolevytilaa ne saavat käyttää. Docker tarjoaa myös työkalun nimeltään Compose, jolla pystytään konfiguroimaan useita kontteja sisältävä ympäristö. Konttien asetukset, kuten kontin käyttämä levykuva ja palvelun nimi kirjoitetaan YAML-merkintäkieltä (YAML Ain't Markup Language) käyttävään tiedostoon docker-compose.yml. Ajamalla *docker compose up* -komennon, kaikki docker-compose.yml-tiedostossa määritellyt palvelut käynnistyvät käyttäjän tiedostoon antamien konfiguraatioiden kanssa. (Docker overview n.d; Overview of Docker Compose n.d.)

Zhangin, Liun, Pun, Doun, Wun ja Zhoun (2018) toteuttamassa tutkimuksessa vertailtiin virtuaalikoneiden ja konttiteknologioiden eroja Big Data -ympäristössä. Tutkimuksen tuloksista käy ilmi, että kontit olivat virtuaalikoneita parempia melkein kaikilla testatuilla mittareilla, kuten käynnistymisen nopeudessa sekä prosessorin ja muistin hyötykäytössä ennalta määritellyn työkuorman alla. Tutkimuksen perusteella konttiteknologia vaikutti paremmalta ratkaisulta käytettäväksi toteutuksessa verrattuna virtuaalikoneisiin. Kovács (2017) vertaili eri konttiteknologioita keskenään Linux-järjestelmässä. Hän suoritti näille konttiteknologioille erilaisia testejä, joista saatujen tulosten perusteella konttiteknologioiden välillä ei ollut mittavia eroja. Docker kuitenkin vaati ylimääräisiä muutoksia verkkokonfiguraatioihin, jotta lähetetyn datamäärän keskihajontaa saatiin laskettua. (Kovács 2017, 47-50; Zhang, Liu, Pu, Dou, Wu & Zhou 2018, 1, 4-8.)

Vaikka edellisessä kappaleessa suoritetun vertailun perusteella Docker vaati ylimääräistä työtä järjestelmän verkkoasetusten suhteen, erot olivat muuten pieniä. Dockerin helppokäyttöisyyden, tehokkuuden, suosion ja keveyden takia se valittiin käytettäväksi toteutuksessa. Teknologian suosion ansiosta siitä on enemmän dokumentaatiota saatavilla. Docker-kontteja on helppo poistaa, lisätä ja konfiguroida tarpeen mukaan. Koko tekninen toteutus voidaan rakentaa kontteihin ja kaikkia kontteja pystytään hallitsemaan Compose-työkalun avulla. Levykuviin pystytään lisäämään palve-

luita ja työkaluja helposti jälkeenpäin, jos muutoksille ilmenee tarvetta. Konttien ja Docker-ympäristön yleisiä asetuksia voidaan myös kätevästi muokata docker-compose.yml-tiedostosta tarpeen mukaan.

### 8.1.2 MongoDB

MongoDB on relaatio- ja NoSQL-tietokantojen ominaisuuksia yhdistelevä dokumenttitietokantaohjelmisto. NoSQL-ominaisuuksiensa ansiosta, MongoDB tarjoaa joustavan tietomallin erilaisten datarakenteiden säilyttämiseen tietokannassa. Tietokantaan syötettävä data tallennetaan relaatiotietokantojen rivejä vastaaviin dokumentteihin BSON-formaatissa (Binary JSON), joka perustuu JSON-formaattiin (JavaScript Object Notation). Dokumentit taas varastoidaan kokoelmiin, jotka vastaavat relaatiotietokannoista tuttuja tauluja, ja dokumentin kentät taas ovat samanlaisia relaatiotietokantojen sarakkeiden kanssa. (MongoDB Architecture Guide 2015, 2-4.)

MongoDB valittiin palvelun tietokantaratkaisuksi, koska dokumentin kenttään syötettävä data voi viedä paljon tilaa ja datan rakenne voi olla kompleksi. MongoDB:hen pystytään säilömään kaikenlaisia tietorakenteita ja tietokantaa voi helposti skaalata suuremmaksi, jos kiintolevytilasta tulee puutetta. MongoDB:ssä tietokantarakenteen ei tarvitse olla valmiiksi tiedossa. Esimerkiksi dokumentteihin luodaan kenttiä sitä mukaan, kun kenttiin syötetään dataa, eikä kaikkien dokumenttien tarvitse sisältää samoja kenttiä, vaikka ne olisivat samassa kokoelmassa. (MongoDB Architecture Guide 2015, 1-5, 8.)

Ennen MongoDB:n valintaa palvelun tietokannaksi, tutkittiin muita mahdollisia tietokanta vaihtoehtoja. Lopullinen vertailu tehtiin MongoDB:n ja avoimen lähdekoodin PostgreSQL-relaatiotietokannan välillä. Vertailun apuna käytettiin Jungin, Younin, Baen ja Choin (2015) suorittamaa tutkimusta MongoDB:n ja PostgreSQL:n eroista suorituskyvyssä. Tutkimuksessa vertailtiin näiden tietokantaohjelmistojen suorittamiin tietokantaoperaatioihin kuluvaa aikaa. Tutkimuksessa kävi ilmi, että MongoDB on yleisesti PostgreSQL:ää suorituskykyisempi tietokantaohjelmisto ja ottaen vielä huomioon edellisessä kappaleessa mainitut hyödyt, MongoDB vaikutti paremmalta vaihtoehdolta järjestelmässä käytettäväksi tietokantaratkaisuksi. (Jung, Youn, Bae & Choi 2015, 14-17.)

### 8.1.3 Apache HTTP Server

Apache HTTP Server on avoimen lähdekoodin HTTP-palvelin, mikä on osa Apache Software Foundation -yhtiötä. Apachea käytetään toteutuksessa React-sovelluksen alustana ja Python-rajapintaa ylläpitävänä HTTP-palvelimena. Apache siirtää tietoa käyttäjän tietokoneen ja toteutetun järjestelmän eri komponenttien välillä, käyttäen HTTP-protokollaa, eli Apache yhdistää käyttäjän toteutuksen tarjoamaan palveluun. (What is the Apache HTTP Server Project n.d.)

Toinen varteenotettava vaihtoehto toteutuksessa käytettäväksi avoimen lähdekoodin HTTP-palvelimeksi olisi ollut NGINX. Käännän (2013) suorittamassa Apachen ja NGINXin välisessä suorituskykyvertailussa NGINX pärjäsi Apachea paremmin palvelemalla enemmän asiakaslaitteita tietyn aikarajan puitteissa. NGINX on Apachea kevyempi palvelinohjelmisto, joka käyttää myös Apachea vähemmän resursseja. Apache valikoitui käytettäväksi kuitenkin modulaarisuutensa takia. Modulaarisuus mahdollistaa ulkopuolisten lisäosien liittämisen Apacheen. Tämä on hyödyllinen ominaisuus, jos toteutettavaan HTTP-palvelimeen halutaan tulevaisuudessa lisätä erilaisia toiminnallisuuksia. Tällöin Apachesta löytyy suuremmalla todennäköisyydellä tuki halutulle toiminnolle. Myös toteutuksen kehitysvaiheessa tulee todennäköisesti olemaan vähemmän riippuvuuksiin liittyviä ongelmia Apacheen löytyvien lisäosien ansiosta. (Kääntä 2013, 10-11, 13, 33-37.)

### 8.1.4 React

React on komponentteihin ja niiden tilojen muutoksiin perustuva JavaScript-kirjasto, jolla rakennetaan tietokoneille ja mobiililaitteille sopivia käyttöliittymiä verkkosovelluksiin (React 2021). Reactia käytetään toteutuksessa käyttäjälle selaimessa näkyvän käyttöliittymän rakentamiseen. React-kirjaston lisäksi käyttöliittymässä käytetään muitakin JavaScript-kirjastoja, mutta verkkosovellusta hallitseva kirjasto on React.

Käyttöliittymän rakentamiseen käytettävää JavaScript-kirjastoa valikoidessa apuna käytettiin Mattias Levlinin pro gradu -tutkielmaa. Levlin (2020) vertaili neljän eri JavaScript-kirjaston: React, Angular, Vue ja Svelte suorituskykyä niiden käsitellessä dokumenttioliomallia (engl. Document Object Model). Levlin esittelee työssään JavaScript-kirjastoille suoritettujen erilaisten testien tuloksia. Tulosten perusteella React oli monessa vertailussa paras kirjasto, eikä se ollut millään mittarilla heikoin kirjasto. Levlin sijoittikin Reactin ensimmäiselle sijalle vertailussa. Levlinin mukaan Reactissa

ei esiintynyt huomattavia heikkouksia verrattuna muihin kirjastoihin ja sillä oli useita vahvuuksia, kuten hyvä suorituskyky sekä React oli vertailluista kirjastoista käytetyin. (Levlin 2020, 58-70.)

Reactin kiitettävän suorituskyvyn ohella tunnettavuus oli yksi syy, minkä takia kirjasto valittiin käytettäväksi toteutuksessa. Koska Reactilla on enemmän käyttäjiä, kirjastolla ohjelmoidessa kohdatuissa ongelmatilanteissa voi löytyä enemmän materiaalia käyttäjiltä, jotka ovat kohdanneet samanlaisia ongelmia. Vähemmän käytetyissä kirjastoissa ongelmatilanteiden ratkaisuihin liittyvää aineistoa voi löytyä vähemmän tai sitä voi olla vaikeammin saatavilla, mikä voi hidastaa teknisen toteutuksen valmistumista.

### 8.1.5 Python ja Flask

Python on korkean tason tulkittava (engl. interpreted) ohjelmointikieli, jonka tekijänoikeudet omistaa Python Software Foundation -organisaatio. Kielen syntaksi on yksinkertainen ja se sopii hyvin aloitteleville ohjelmoijille. Python sisältää monipuolisen standardikirjaston ja kielelle löytyy myös paljon kolmannen osapuolen kirjastoja, jotka mahdollistavat tuen useille eri protokollille ja ohjelmistoille. (General Python FAQ 2021.)

Purer (2009) vertaili PHP-, Python- ja Ruby-ohjelmointikielien ominaisuuksia ja suorituskykyä keskenään. Purerin mukaan suorituskykyyn liittyvissä vertailuissa ohjelmointikielet olivat hyvin lähellä toisiaan, eikä mikään kielistä ollut huomattavasti muita parempi. Jos kuitenkin otetaan kaikkien vertailujen tulokset huomioon, Python ja Ruby pärjäsivät parhaiten. Purer suositteli näistä kahdesta Python-kieltä käytettäväksi, uutta verkkosovellusta rakennettaessa. PHP-kielellä on rakennettu valmiita sisällönhallintajärjestelmiä (engl. content management system) ja Purer suositteli-kin PHP:tä käytettäväksi tilanteissa, joissa valmista PHP:llä rakennettua sisällönhallintajärjestelmää voidaan hyödyntää verkkosovelluksen kehityksessä, eikä järjestelmän koodikanta tarvitse huomattavia muokkauksia. (Purer 2009, 8-15.)

Monipuolisen kirjastovalikoiman, integrointimahdollisuuksien, helpon syntaksin ja edellisessä kappaleessa mainittujen vertailujen tulosten takia Python valittiin eri sovellusten rajapintoja yhdistäväksi tekijäksi. Kuten luvussa 6 mainitaan, myös Volatility on kirjoitettu Python-kielellä, joten Pythonia on käytettävä, jotta Volatility-kirjastoa voidaan hyödyntää. Pythonia käytetään teknisessä

toteutuksessa muistivedosten analysoimiseen, Analyysi-konttien käskyttämiseen, tiedostojen kirjoittamiseen levylle sekä MongoDB-tietokannan yhdistämiseen muiden palveluiden kanssa.

Pythonia voitaisiin käyttää esimerkiksi vain muistivedosten analysoimiseen, jolloin muut komponentit toteutettaisiin jollain toisella kielellä. Pythonin hyödyntäminen kaikissa komponenteissa kuitenkin mahdollistaa koodin uudelleenkäyttämisen komponenttien välillä, vähentäen ohjelmointiin sekä uuden teknologian opiskelemiseen kuluvaa aikaa, tehden koko teknisen toteutuksen prosessista sujuvamman.

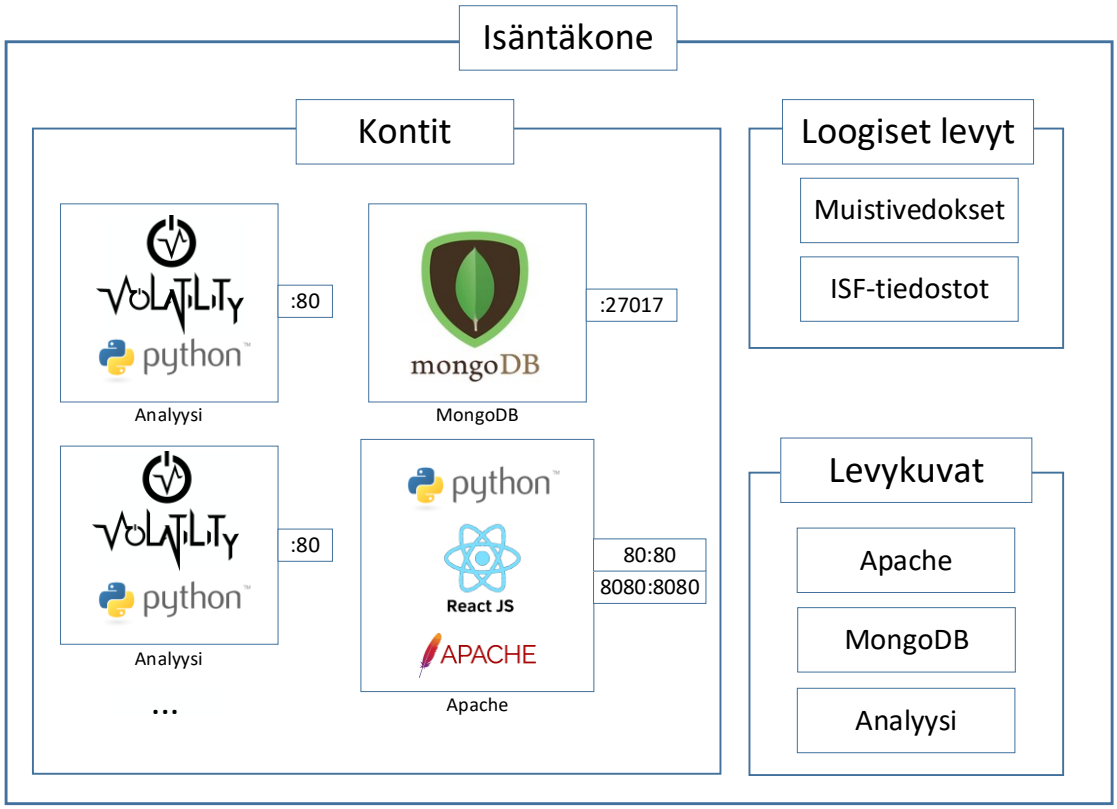
Järjestelmän komponenttien välisien HTTP-rajapintojen rakentamiseen käytettävästä Python-verkkosovelluskehyksestä (engl. web application framework) suoritettiin myös vertailua. Lopullinen päätös tehtiin Flask- ja Django-kehysten välillä. Päätöksen apuna käytettiin muun muassa Ghimiren (2020) suorittamaa vertailua näistä verkkosovelluskehyksistä. Ghimire toteaa Flaskin sopivan paremmin pienen mittakaavan projekteihin ja Djangon taas suurempiin kokonaisuuksiin. Ghimiren mukaan Flask on myös helpompi opetella, mutta sen ylläpitäminen on vaikeampaa. Crawford ja Hussain (n.d) vertailivat eri verkkosovellusteknologioita keskenään. Django ja Flask kuuluivat testattavien teknologioiden joukkoon. Crawfordin ja Hussainin suorittamissa suorituskykyvertailuissa Flask pärjäsi Djangoa paremmin kaikissa kolmessa testissä. Muissa testeissä Flask ei kuulunut testattavien teknologioiden joukkoon. Vertailujen perusteella Flask-verkkosovelluskehyksen katsottiin sopivan paremmin toteutukseen. Jos toteutettava järjestelmä olisi ollut vielä laajempi, Django olisi voitu valita Flaskin sijaan. (Crawford & Hussain n.d, 74; Ghimire 2020, 21-22, 32.)

## 8.2 Palvelun kuvaus

Toteutustavaksi valittiin verkkosovellus, jonne analysoitavia muistivedoksia ja ISF-tiedostoja (Intermediate Symbol File) voidaan ladata. Sovellus tukee Windows- ja Linux-käyttöjärjestelmien analyysia. Muistivedokset analysoidaan automaattisesti ajamalla niihin Volatilitysta löytyviä lisäosia, joiden tuloksia käyttäjä pystyy jälkeenpäin tarkastelemaan ja suodattamaan. Volatility-versioista käytettäväksi valittiin kolmas versio, eli tämän hetken uusin, niin kuin luvussa 6.4 mainitaan. Sovellus ei itsessään etsi merkkejä haittaohjelmista, vaan jättää analysoinnin käyttäjälle. Muistivedokset ja ISF-tiedostot jäävät talteen verkkosovellusta ylläpitävälle palvelimelle, ja vedosten analysointia voi jatkaa milloin vain.

Toteutettavan palvelun nimeksi valittiin Autovola, jonka infrastruktuuria kuvataan kuviossa 13. Palvelulle riittää yksi isäntäkone, joka ylläpitää levykuvia, loogisia levyjä sekä Docker-kontteja, jotka sisältävät palvelun varsinaiset komponentit. Kaikki Dockeriin liittyvät komponentit ovat listattuina isäntäkoneen sisälle omiin lokeroihinsa. Levykuvat-niminen lokero sisältää kolme eri levykuvaa: Apache, MongoDB ja Analyysi, joiden pohjalta kontteja suoritetaan.

Kuvion 13 Kontit-lokero sisältää isäntäkoneen ylläpitämät kontit. Jokainen kontti on omassa lokerossansa, minkä alla näkyy kontin käyttämä levykuva. Kontin oikealla puolella olevat palikat kertovat kyseisen kontin avoimet portit. Esimerkiksi palikan arvo ”:80” tarkoittaa, että kontissa on avoinna portti numero 80. Jos ennen kaksoispistettä on jokin luku, kyseinen isäntäkoneen portti tunneloidaan kontin porttiin, joka näkyy kaksoispisteen jälkeen. Esimerkiksi merkintä ”80:8080” tarkoittaisi, että isäntäkoneen porttiin 80 saapuva liikenne ohjataan kontin porttiin 8080.




Kuvio 13. Palvelun infrastruktuuri selitettynä

Eri kontit ja niiden tarkoitus selitettyinä:

- **Apache**: Portissa 80 on JavaScriptin React.js-kirjastolla rakennettu verkkosovellus, josta käyttäjä pystyy lataamaan palveluun muistivedoksia ja ISF-tiedostoja, ajamaan Volatility 3:n lisäosia muistivedoksiin sekä analysoimaan lisäosien tuloksia. Portissa 8080 on Flask-rajapinta, jota käytetään tietokantayhteyksien luomiseen MongoDB:hen sekä muistivedosten ja ISF-tiedostojen kirjoittamiseen palvelimelle. Tämä toimii AJAX-pyynnöillä (Asynchronous Javascript and XML), joita React-sovellus lähettää palvelimen porttiin 8080. Molemmissa porteissa olevien sovellusten pohjalla toimii luvussa 8.1.3 kuvailtu Apache.
- **MongoDB**: Kontin portissa 27017 on MongoDB-tietokantapalvelu, jonne säilötään tietoa muistivedoksista, kuten Volatility 3 -lisäosien tuloksia. Tietoa myös haetaan tietokannasta esimerkiksi käyttäjän alkaessa analysoimaan muistivedosta.
- **Analyysi**: Analyysi-kontit ajavat muun muassa Volatility 3:n lisäosia muistivedoksiin ja lähettävät tulokset MongoDB-tietokantaan. Konttien porteissa 80 on Flask-rajapinta, mitä kautta Apache-kontti käskyttää Analyysi-kontteja lähettämällä niihin HTTP-pyyntöjä. Analyysi-kontteja voi olla yksi tai useita. Jos isäntäkoneella on paljon resursseja käytettävissään, Analyysi-kontteja voidaan pystyttää enemmän, jolloin muistivedoksia pystytään analysoimaan nopeammin.

Loogiset levyt -lokero (ks. kuvio 13) sisältää hakemistoja, jotka jaetaan Apache-kontin ja Analyysi-konttien kesken. Hakemistojen jakaminen toteutetaan käyttämällä Dockerin Volumes-mekanismia, mikä mahdollistaa tiedostojärjestelmään luotavan pysyvän levytilan, johon kontit voivat kirjoittaa dataa ja lukea sitä (Use volumes n.d). Levytilaa käytetään käyttäjän lataamien muistivedosten ja ISF-tiedostojen säilyttämiseen. Jaetun levytilan ansiosta tiedostot kirjoitetaan vain yhteen sijaintiin säästäen levytilaa.

# 9 Toteutus

Toimeksiantajan kanssa sovittiin tietyin väliajoin palavereita, jolloin sovelluksen sen hetken tilaa sekä uusia edellisen palaverin jälkeen lisättyjä toimintoja käytiin läpi. Tätä jatkettiin koko teknisen toteutuksen ajan pitäen toimeksiantaja tietoisena sovelluksen kehityksestä. Toteutusta varten projektille tehtiin oma repositorio toimeksiantajan GitLab-versionhallintapalveluun, mistä käsin projektia päivitettiin.

Projekti suunniteltiin niin, että kaikki Autovola-sovelluksen komponentit löytyvät yhdestä repositoriosta, eikä käyttäjän tarvitse ladata sovellukseen osia muista ulkoisista lähteistä. Autovolalla on kuitenkin joitain vaatimuksia, jotka liittyvät isäntäkoneen käyttöjärjestelmään ja sen palveluihin.

Käyttöjärjestelmän tulisi olla Linux-pohjainen ja ympäristöstä tulisi löytyä Docker ja Docker Compose -sovellukset. Autovolan repositoriosta löytyy ohjeistus, mistä kyseiset sovellukset saadaan asennettua.

Autovola-repositorion rakennetta kuvataan liitteessä 3. Hakemistojen nimet ovat kirjoitettu englanniksi, niin kuin ne ovat projektissakin. Ylimmän hakemiston alla on kolme päähakemistoa: autovolagui, testing ja containers. Autovolagui sisältää Apache-kontin porteissa 80 ja 8080 olevien verkkosovelluksien lähdekoodin. Testing-hakemistossa on ympäristön yksinkertaiseen testaamiseen käytettäviä Python-tiedostoja ja containers-hakemiston alta löytyy jokaiselle kontille oma kansionsa sekä images-kansio, mikä sisältää konttien tarvitsemat Docker-levykuvat.

## 9.1 Docker-kontit

Autovola-projektin päähakemistossa on docker-compose.yml-tiedosto, joka sisältää Autovolan konttien konfiguraatiot. Compose-työkalua kuvaillaan tarkemmin luvussa 8.1.1. Päähakemiston alla on myös .env-niminen ympäristömuuttujia sisältävä tiedosto, minkä sisältö näkyy liitteessä 4. Tiedostossa rivit, jotka alkavat #-merkillä, ovat kommentteja, kun taas muilla riveillä =-merkki toimii ympäristömuuttujan nimen ja arvon erottimena, jossa muuttujan nimi tulee ensin ja muuttujan arvo sen jälkeen (Declare default environment variables in file n.d). Docker-compose.yml sekä konttien Dockerfile-tiedostot käyttävät kyseisiä ympäristömuuttujia tiedostoista löytyvien samannimisillä ympäristömuuttujilla merkittyjen kenttien täyttämiseen, muuttujien arvoilla.

Docker-compose.yml-tiedosto sisältää määritykset konttien tarvitsemalle jaetulle levytilalle sekä verkkoasetukset konteille. Nämä konfiguraatiot näkyvät kuviossa 14. Kuvion yläosassa näkyvän volumes-avaimen alla olevat sisäkkäiset arvot ovat jaettujen levytilojen nimet muistivedoksille (dumps) sekä Linux (linux_symbols) ja Windows (windows_symbols) ISF-tiedostoille. Kaikki samassa docker-compose.yml-tiedostossa olevat palvelut voivat lukea tai kirjoittaa tietoa näihin levytiloihin, jos levytiloille on määritelty sijainti palvelun omassa konfiguraatiossa. (Use volumes n.d.)

Networks-avaimen alla olevat määritykset luovat palvelun konteille oman yksityisen sisäverkkonsa ja sille oletusyhdyskäytävän (Docker network create n.d). Aliverkko ja oletusyhdyskäytävä saavat arvonsa .env-tiedoston ympäristömuuttujista (ks. liite 4). Jokaisen ympäristömuuttujan nimi on

suljettu kaarisulkeilla, sekä $-merkillä ennen ensimmäistä kaarta, jotta Compose-työkalu ymmärtää kyseisen merkkijonon viittaavan ympäristömuuttujaan (Environment variables in Compose n.d).

```
volumes:
  dumps:
  windows_symbols:
  linux_symbols:

networks:
  private:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: ${SUBNET}
          gateway: ${GATEWAY}
```

Kuvio 14. Docker-compose.yml-tiedoston jaetut hakemistot ja verkkoasetukset

### 9.1.1 MongoDB

MongoDB-kontissa sijaitsee Autovolan käyttämä MongoDB-tietokanta. Tietokannassa on kaksi kokoelmaa, joiden nimet ovat *windows_pluginresults* ja *linux_pluginresults*. Jokaiselle Autovola-palveluun ladatulle muistivedokselle tehdään oma dokumentti toiseen näistä kokoelmista. Tämä riippuu laitteen käyttöjärjestelmästä, josta muistivedos on otettu. Yksittäisen muistivedoksen dokumenttiin luodaan kenttiä, jotka sisältävät tietoa vedoksesta, kuten Volatility 3:n lisäosien tulosteita, tietoa käyttöjärjestelmästä sekä React-sovelluksen käyttöliittymässä näkyvät kyseisen muistivedoksen raportin tiedot. React-käyttöliittymää esitellään luvussa 9.2.

MongoDB-palvelun docker-compose.yml-konfiguraatio näkyy kuviossa 15. Palvelu käyttää kolmea .env-tiedostossa määriteltyä ympäristömuuttujaa (ks. liite 4), jotka ovat *MONGO_USER* (tietokannan käyttäjätunnus), *MONGO_PASSWORD* (käyttäjätunnuksen salasana) ja *MONGO_IP* (kontin IP-osoite). Kontin portti 27017 on määritelty avoinna olevaksi, koska MongoDB-tietokantapalvelu käyttää sitä.

```
  mongodb:
    image: autovola/mongo:latest
    restart: always
    container_name: mongodb
    privileged: true
    environment:
      - MONGO_INITDB_ROOT_USERNAME=${MONGO_USER}
      - MONGO_INITDB_ROOT_PASSWORD=${MONGO_PASSWORD}
      - MONGO_INITDB_DATABASE=autovola
    networks:
      private:
        ipv4_address:
          ${MONGO_IP}
    expose:
      - "27017"
    volumes:
      - ./containers/mongodb/mongo-init.sh:/docker-
entrypoint-initdb.d/mongo-init.sh
      - ./containers/mongodb/mongod.conf:/etc/mongod.conf
```

Kuvio 15. MongoDB-palvelun konfiguraatio docker-compose.yml-tiedostossa

Kuviossa 15 volumes-avaimen alla on kaksi eri arvoa. Ne määrittelevät isäntäkoneen tiedostoja, jotka kopioidaan konttiin, kun sitä rakennetaan (Use volumes n.d). Isäntäkoneen tiedostosijainti erotetaan kaksoispisteellä kohdesijainnista kontissa (Use volumes n.d). Toinen tiedostoista on mongod.conf, joka sisältää MongoDB-tietokantaan liittyviä asetuksia. Tiedoston sisältö näkyy liitteessä 5. Tiedosto siirretään kontissa polkuun */etc/mongod.conf*, jolloin se ylikirjoittaa MongoDB:n saman nimisen oletuskonfiguraatiotiedoston, mikä antaa käyttäjälle mahdollisuuden muuttaa oletusasetuksia haluamallaan tavalla (Configuration File Options n.d).

Volumes-avaimen toinen tiedosto on mongo-init.sh. Se kopioidaan kontissa olevaan */docker-entrypoint-initdb.d/*-polkuun. Hakemistossa olevat *.sh* ja *.js*-päätteiset tiedostot suoritetaan kontin käynnistyessä ensimmäisen kerran (Initializing a fresh instance n.d). Mongo-init.sh-tiedoston sisältö näkyy kuviossa 16. Se on komentorivi-skripti, joka lisää uuden käyttäjän tietokantaan, käyttäen kuviossa 15 määriteltyjä ympäristömuuttujia, jotka ovat määritelty environment-avaimen alla. Kyseinen skripti ajetaan kontin käynnistyessä, koska se on kappaleessa aiemmin mainitun *docker-entrypoint-initdb.d*-hakemiston sisällä.

```
#!/bin/bash
set -e

mongo <<EOF
db.auth('$MONGO_INITDB_ROOT_USERNAME', '$MONGO_INITDB_ROOT_PASSWORD')
db = db.getSiblingDB('$MONGO_INITDB_DATABASE')
db.createUser(
        {
            user: "$MONGO_INITDB_ROOT_USERNAME",
            pwd: "$MONGO_INITDB_ROOT_PASSWORD",
            roles: [
                {
                    role: "readWrite",
                    db: "$MONGO_INITDB_DATABASE"
                }
            ]
        }
);
EOF
```

Kuvio 16. Mongo-init.sh-tiedoston sisältö

### 9.1.2 Apache

Apache-kontti ylläpitää Autovola-palvelun verkkosovelluksia porteissa 80 ja 8080. Porttien palvelut toimivat niin kuin luvun 8.2 suunnitelmassa kuvaillaan. Kyseiset portit ovat myös ainoat käyttäjälle avoinna olevat Autovola-palvelussa. Liitteessä 6 näkyy Apache-kontin konfiguraatio docker-compose.yml-tiedostossa. Konfiguraatiossa käytetään useita .env-tiedostossa (ks. liite 4) määriteltyjä ympäristömuuttujia. Muuttujia käytetään muun muassa määrittelemään Apachen ja MongoDB:n IP-osoitteisiin yhdistetyt verkkotunnukset sekä hakemistojen sijainnit Autovola-pääkansiolle, muistivedoksille ja ISF-tiedostoille. Volumes-avaimen arvot määrittelevät kuviossa 14 näkyvien levytilojen polut tässä kontissa (Use volumes n.d).

Liitteessä 6 näkyvän build-avaimen alla on dockerfile-niminen avain, jonka arvo on Apache-palvelun Dockerfile-tiedoston sijainti isäntäkoneessa. Dockerfile sisältää käskyjä, joita isäntäkone suorittaa rakentaessaan levykuvaa. Käsky voi olla esimerkiksi määräys kopioida jokin tiedosto isäntäkoneelta konttiin tai komentorivillä ajettava komento, joka suoritetaan kontissa. Dockerfileä

käyttämällä nämä käskyt ajetaan automaattisesti levykuvaan sen rakennusvaiheessa, eikä käyttäjän tarvitse kirjoittaa niitä aina manuaalisesti uudelleen. (Compose file version 3 reference n.d; Dockerfile reference n.d.)

Apachen Dockerfile-tiedoston sisältö näkyy liitteessä 7. Tiedostossa jokaisella rivillä #-merkillä kommentoituja ja tyhjiä rivejä lukuun ottamatta on ensin avainsana (FROM, RUN, COPY, ARG, CMD jne.), joka määrittelee käskyn tyypin. Esimerkiksi RUN-avainsana tarkoittaa komentorivikomennon suorittamista. Avainsanaa seuraa lauseke, joka määrittelee, mitä käskyllä tehdään. Esimerkiksi *COPY autovolagui/build/ /var/www/html/* -käsky kopioi isäntäkoneesta *autovolagui/build/*-hakemiston alta kaikki tiedostot ja kansiot kontin hakemistoon */var/www/html/*. (Compose file version 3 reference n.d; Dockerfile reference n.d.)

Dockerfile-tiedoston ensimmäisen käskyn tulisi olla FROM- tai ARG-käsky. Ennen FROM-käskyä Dockerfilestä saa löytyä vain ARG-käskyjä. FROM-käsky määrittelee levykuvan, jota käytetään pohjana rakennusvaiheessa. Apache-kontin Dockerfile-tiedoston käyttämä levykuva on *autovola/apache:latest* (ks. liite 7). ARG-käsky lisää Dockerfileen ympäristömuuttujia, jotka ovat käytössä vain rakennusvaiheessa. Liitteessä 7 määritellään neljä ympäristömuuttujaa, joille haetaan arvot liitteessä 6 näkyvän Apachen Compose-konfiguraation args-avaimen samannimisistä arvoista. Esimerkiksi Dockerfilen *AUTOVOLA_DIRECTORY*-muuttujan arvoksi määräytyy Compose-tiedostossa args-avaimen alla olevan *AUTOVOLA_DIRECTORYn* arvo, eli *${AUTOVOLA_DIRECTORY}*-ympäristömuuttuja. (Compose file version 3 reference n.d; Dockerfile reference n.d.)

ARG-käskyjen jälkeen (ks. liite 7) konttiin luodaan hakemistorakenne käyttäen juuri määriteltyjä ympäristömuuttujia. Muun muassa portin 8080 Flask-rajapinnalle, ISF-tiedostoille ja muistivedoksille tehdään omat kansionsa. Apachen www-data käyttäjästä tehdään hakemistojen omistaja. Seuraavaksi isäntäkoneen Autovola-projektista kopioidaan COPY-käskyillä Flask-rajapintaan liittyvät tiedostot */var/www/api/*-hakemiston alle. Sen jälkeen Dockerfilessä konfiguroidaan Apache-palvelu toimintavalmiuteen ja lisätään portissa 80 toimivan React-verkkosovelluksen koodi isäntäkoneesta */var/www/html/*-hakemiston alle. Viimeinen Dockerfilen käsky on CMD-tyyppinen, mikä määrittelee kontissa ajettavan komennon sen käynnistyessä (Dockerfile reference n.d). Tässä Dockerfilessä komennolla käynnistetään Apache-palvelu (Apache Core Features n.d).

Dockerfilessä kopioidaan (ks. liite 7) *COPY containers/apache/apache/sites/ /etc/apache2/sites-available/* -käskyllä isäntäkoneelta konttiin porteissa 80 ja 8080 toimivien web-sivujen konfiguraatiot. Apache sisältää oletuksena konfiguraatiotiedoston nimeltään 000-default.conf, joka ylikirjoitetaan samannimisellä tiedostolla, jonka sisältö näkyy kuviossa 17. Konfiguraatiota sovelletaan portin 80 React-sovellukseen saapuviin yhteyksiin. Porttiin yhteyttä ottavalle selaimelle tarjotaan */var/www/html/*-hakemiston alla olevia tiedostoja, sillä polku on määritelty DocumentRoot-direktiivillä osoittamaan React-sovellukseen. (Apache Core Features n.d.)

```
<VirtualHost *:80>

    ServerName ${APACHE_HOSTNAME}
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    <Directory /var/www/html/>
        AllowOverride All
        Order Allow,Deny
        Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Kuvio 17. 000-default.conf-tiedoston sisältö

Kuviossa 18 on kontin portin 8080 Apache-konfiguraatio. Directory-direktiivi antaa verkkosovelluksen käyttäjän käyttää */var/www/api/*-hakemiston alla olevia tiedostoja. Flask-rajapintaa varten konfiguraatiossa on määritelty useita WSGI-spesifikaatioon (Python Web Server Gateway) liittyviä asetuksia, kuten WSGIScriptAlias, WSGIDaemonProcess ja WSGIProcessGroup (Quick Configuration Guide n.d).

```
Listen 8080
<VirtualHost *:8080>

    ServerAdmin webmaster@localhost
    ServerName ${APACHE_HOSTNAME}
    WSGIScriptAlias / /var/www/api/flaskapi.wsgi
    WSGIDaemonProcess api user=www-data group=www-data threads=5
    WSGIProcessGroup api

    <Directory /var/www/api/>
        Require all granted
        AllowOverride all
    </Directory>

    ErrorLog /dev/stdout
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Kuvio 18. Api.conf-tiedoston sisältö

WSGI:tä käytetään antamaan Apachelle tuki Python-pohjaisten verkkosovellusten ylläpitämiseen. Python-verkkosovellusta varten WSGIScriptAlias-direktiivillä määritellään WSGI-tiedosto, josta verkkosovellus asennetaan. Kuviosta 18 nähdään, että tiedostoksi on valittu flaskapi.wsgi, jonka sisältö näkyy kuviossa 19. (Quick Configuration Guide n.d.)

```
#!/usr/bin/python
import os
import sys
import logging

logging.basicConfig(stream=sys.stderr)
sys.path.insert(0, '/var/www/api/')
from flaskapi import api as application
application.secret_key = os.getenv("FLASK_SECRET_KEY")
```

Kuvio 19. Flaskapi.wsgi-tiedoston sisältö

### 9.1.3 Analyysi

Analyysi-konttien pääasiallinen tehtävä on Volatility 3:n lisäosien suorittaminen muistivedoksiin ja lisäosista saatujen tulosten lähettäminen tietokantaan, josta ne voidaan hakea ja esittää käyttäjälle. Liitteessä 3 analyse-hakemiston alla näkyvä volatility3-hakemisto sisältää Volatility 3 -projektin, joka asennetaan kontin rakennusvaiheessa. Liitteessä 8 näkyy Analyysi-kontin konfiguraatio docker-compose.yml-tiedostossa. Kontin portissa 80 on Python-kielellä rakennettu Flask-rajapinta, josta kontti vastaanottaa käskyjä. Kaikki Analyysi-kontit jakavat samat volumes-avaimen alla näkyvät levytilat yhdessä Apache-kontin kanssa (ks. liite 6).

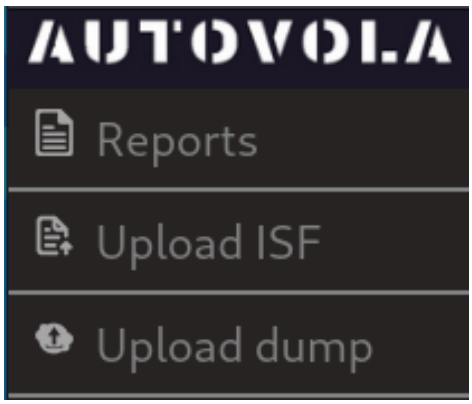
Konfiguraatiossa (ks. liite 8) deploy-avaimen alla olevan mode-avaimen arvoksi on annettu *replicated*, mikä mahdollistaa usean Analyysi-kontin suorittamisen samaan aikaan. Replicas-avaimella taas määritetään suoritettavien Analyysi-konttien määrä, mikä haetaan ”.env”-tiedoston *ANALYSE_MACHINE_AMOUNT*-muuttujasta (ks. liite 4). Limits-avaimen alla olevan memory-avaimen arvo määrittelee, kuinka paljon keskusmuistia yksi Analyysi-kontti voi enintään käyttää. Reservations-avaimen alla oleva memory-avain taas määrittelee, kuinka paljon muistia yhdelle kontille on aina varattuna. (Compose file version 3 reference n.d.)

Liitteessä 9 näkyy Analyysi-konttien käyttämän Dockerfile-tiedoston sisältö. Dockerfilessä FROM- ja ARG-käskyjen jälkeen luodaan hakemistorakenne liitteessä 8 volumes-avaimen alla oleville levytiloille sekä kontin käyttämille Python-tiedostoille. Volatility 3 -projekti ja muu Python-koodi kopioidaan konttiin Volatility Foundationin tarjoamien valmiiden Windows ISF-tiedostojen kera. Lopuksi Volatility 3 -Python kirjasto asennetaan konttiin, ja kontin käynnistyessä CMD-käskyllä määritetään suoritettavaksi portissa 80 toimiva Python-palvelu. (Dockerfile reference n.d.)

## 9.2 Käyttöliittymä

Autovola-palvelun käyttöliittymässä on kolme pääsivua, jotka näkyvät kuviossa 20 näkyvässä verkkosovelluksen sivupalkissa. Eri painikkeiden tarkoitus selitettynä:

- Reports-painiketta painamalla käyttäjä ohjataan sivulle, josta avautuu näkymä kaikkiin palveluun ladattuihin muistivedoksiin.
- Upload ISF -painiketta painamalla käyttäjä ohjataan sivulle, josta palveluun voi ladata ISF-tiedostoja.
- Upload dump -painiketta painamalla käyttäjä ohjataan sivulle, josta palveluun voi ladata muistivedoksia.

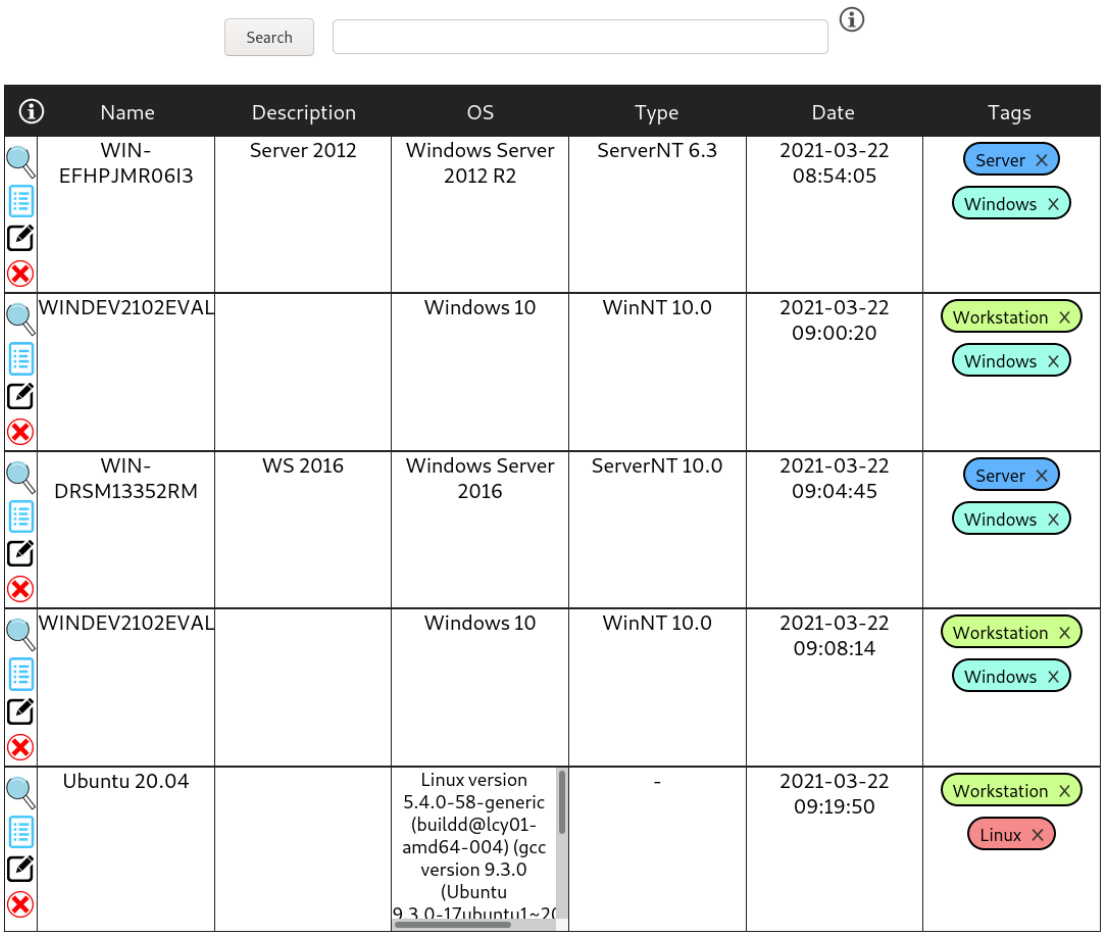



Kuvio 20. Autovolan käyttöliittymän sivupalkki

Verkkosovelluksen eri sivuja ja niiden toiminnallisuutta esitellään tarkemmin tämän luvun alilu-vuissa.

### 9.2.1 Muistivedosten raportit

Jokaiselle Autovola-palveluun ladatulle muistivedokselle luodaan oma lohkonsa Reports-sivulle. Kuviossa 21 on esimerkki näkymä Reports-sivun raporteista, kun palveluun on ladattu viisi eri muistivedosta. Jokainen rivi mustan yläpalkin alla edustaa yhtä muistivedosta. Yläpalkki sisältää sarakkeita, jotka ovat jaettu vedoksen nimeen, kuvaukseen, käyttöjärjestelmään, käyttöjärjestelmän tyyppiin (vain Windows-muistivedoksissa), aikaleimaan muistivedoksen lataamisesta palveluun sekä tageihin. Palkit muistivedosten lohkoissa vastaavat näihin sarakkeisiin.



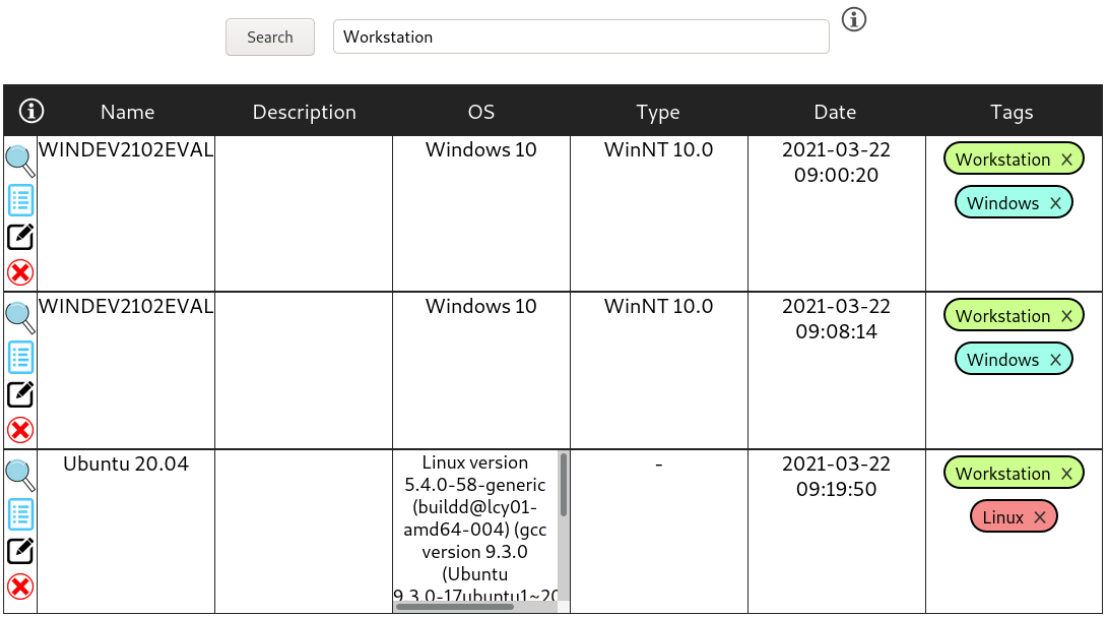

Kuvio 21. Esimerkki käyttöliittymässä näkyvistä muistivedoksista

Jokaisen lohkon vasemmalla puolella on neljä kuvaketta allekkain (ks. kuvio 21). Kuvakkeet toimivat painikkeina, joista avautuu erilaisia toiminnallisuuksia. Painikkeet selitettynä järjestyksessä ylhäältä alas:

- Vie sivulle, josta pystytään tutkimaan ja suodattamaan Volatility 3:n lisäosien tuloksia, jotka ovat otettu kyseisestä muistivedoksesta. Sivun toiminnallisuutta kuvataan luvussa 9.2.4.
- Avaa ikkunan, josta voidaan valita, mitä lisäosia muistivedokseen suoritetaan. Ikkunaa ja lisäosien suorittamisen prosessia kuvaillaan luvussa 9.2.3.
- Avaa ikkunan, josta voidaan muuttaa lohkossa näkyvää järjestelmän nimeä, kuvausta tai tageja.
- Avaa ikkunan, josta muistivedos ja kaikki siihen liittyvä data tietokannasta voidaan poistaa Autovola-järjestelmästä. Toiminto vapauttaa levytilaa ja poistaa lohkon raporteista, koska muistivedoksen dokumenttia ei enää ole tietokannassa.

Kuvion 21 yläosassa näkyvästä hakupalkista pystytään suodattamaan käyttäjälle näkyviä raportteja. Hakupalkkiin voi kirjoittaa raportissa näkyviä avainsanoja, kuten tunnisteiden (engl. tag) nimiä, järjestelmän nimen tai päiväyksen, jolloin näkyville jää vain raportit, jotka sisältävät kyseisen

avainsanan. Esimerkiksi kuviossa 22 raportteja on suodatettu sanalla *Workstation* ja käyttöliittymään näkyville ovat jääneet vain kyseisen sanan sisältävät raportit. Jos käyttäjä hakisi raportteja merkkijonolla *Workstation Linux*, vain kuvion 22 raporteista viimeisin näkyisi käyttäjälle, sillä haku toimii konjunktiolla, eli välilyönnillä erotetut sanat ovat erillisiä hakusanoja ja raportista on löydyttävä kaikki erilliset hakusanat, jotta se näytetään käyttäjälle.



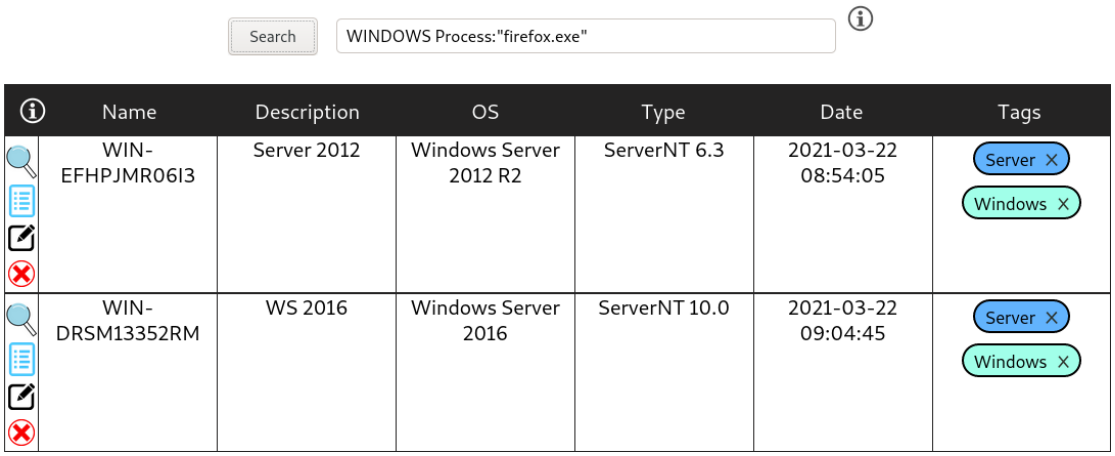

Kuvio 22. Raporttien suodatuksen tulos

Raportteja pystytään myös suodattamaan Volatilityn lisäosien tulosten perusteella. Tämä toimii kirjoittamalla yhden kolmesta avainsanasta (WINDOWS, LINUX tai GLOBAL) haun alkuun. Avainsanojen merkitys:

- **WINDOWS**: Suodata Windows-käyttöjärjestelmän raportteja.
- **LINUX**: Suodata Linux-käyttöjärjestelmän raportteja.
- **GLOBAL**: Suodata Windows- ja Linux-käyttöjärjestelmien raportteja.

Avainsanan jälkeen haun syntaksi noudattaa luvussa 9.2.4 esiteltävää hakusyntaksia. Kuviossa 23 raportit on suodatettu WINDOWS-avainsanalla koskemaan pelkästään Windows-käyttöjärjestelmän raportteja, jolloin Linux-muistivedosten raportteja ei näytetä tuloksissa. Avainsanan jälkeen tuleva *Process:"firefox.exe"*-hakusana tarkoittaa, että Windows-raporttien Volatility 3:n lisäosien tuloksista etsitään Process-kolumnia, jonka arvo on *firefox.exe*. Jos raporttia vastaavan muistive-

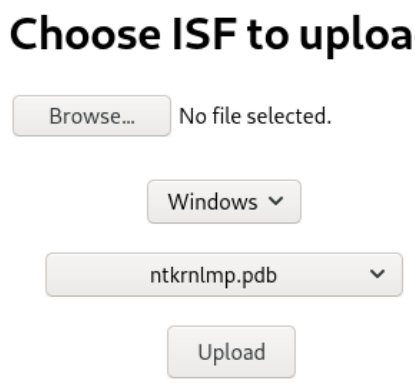
doksen tuloksista löytyy kyseinen osuma, raportti näytetään käyttäjälle. Muussa tapauksessa raportti piilotetaan käyttäjältä. Jos WINDOWS-avainsanan tilalla olisi GLOBAL, firefox.exe-nimistä prosessia etsittäisiin myös Linux-raporteista.




Kuvio 23. WINDOWS-raporttien suodattaminen prosessin nimen perusteella

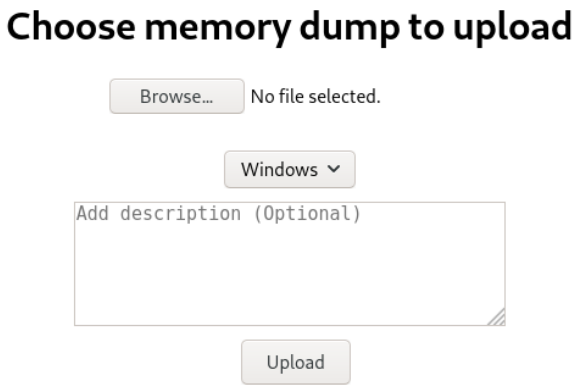
### 9.2.2 Muistivedoksen ja ISF-tiedoston lähettäminen Autovolalle

Kuvion 20 Upload ISF -painiketta painamalla käyttäjä ohjataan sivulle, jossa näkyy lomake (ks. kuvio 24), josta käyttäjä pystyy lataamaan palveluun ISF-tiedoston. Lomake sisältää pudotusvalikon, josta voi valita joko Linuxin tai Windowsin riippuen siitä, mille käyttöjärjestelmälle ISF-tiedosto on tarkoitettu. Jos käyttäjä valitsee Windows-vaihtoehdon, hänelle tulee näkyviin toinen pudotusvalikko, josta valitaan vaihtoehto alkuperäisen PDB-tiedoston (engl. Program database) nimen mukaan. Sivu sisältää myös ohjeistuksen sopivien ISF-tiedostojen luomiseen. Painamalla Upload-painiketta, valittua ISF-tiedostoa aletaan lähettämään palvelimelle pienissä osissa kerrallaan.




Kuvio 24. ISF-tiedoston latauslomake

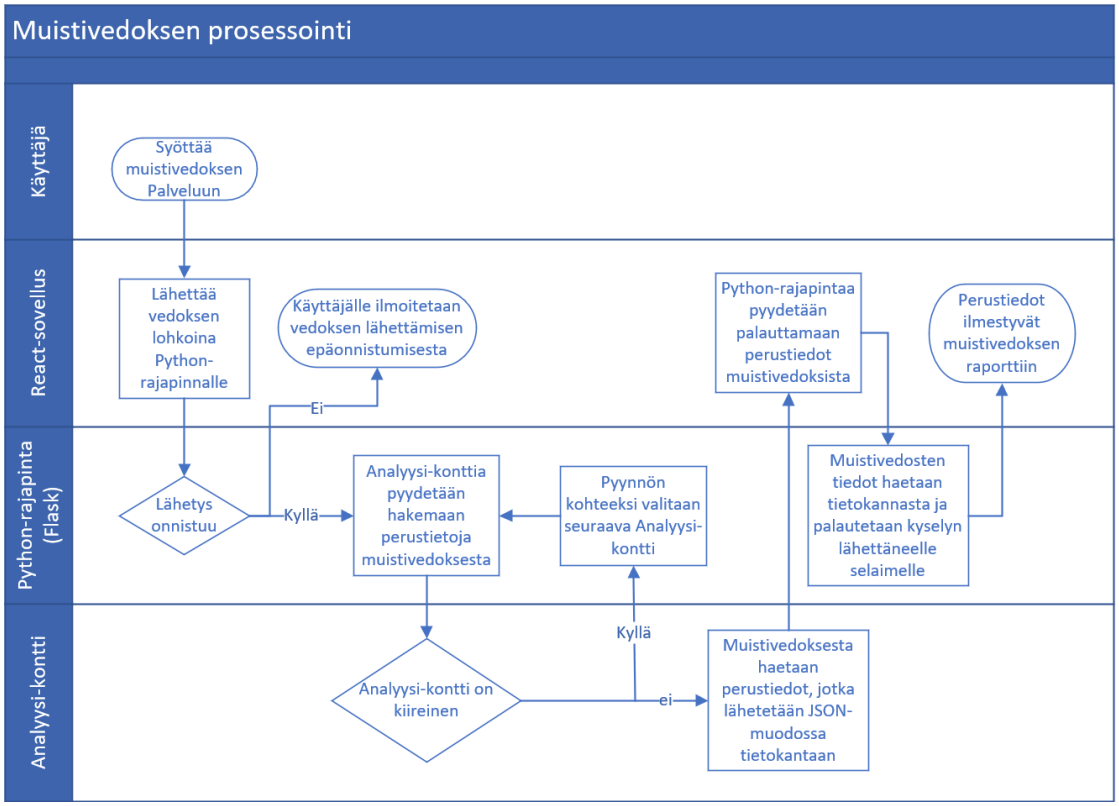
Kuvion 20 Upload dump -painikkeesta käyttäjä ohjataan sivulle, josta hän pystyy lataamaan palveluun muistivedoksen. Sivu sisältää kuviossa 25 näkyvän muistivedoksen latauslomakkeen. Lomakkeessa on samanlainen pudotusvalikko kuin ISF-tiedostojen latauslomakkeessa (ks. kuvio 24), josta voi valita Windows- ja Linux-käyttöjärjestelmän väliltä. Valinnan pitäisi olla sama kuin käyttöjärjestelmän, josta muistivedos on otettu. Käyttäjä voi vaihtoehtoisesti kirjoittaa kuvauksen muistivedokselle, joka näkyy vedoksen raportissa (ks. kuvio 21). Painamalla Upload-painiketta, valittu muistivedos lähetetään palvelimelle pienissä osissa, niin kuin ISF-tiedostotkin.




Kuvio 25. Muistivedoksen latauslomake

Muistivedokset ja ISF-tiedostot lähetetään Autovolalle käyttäen HTTP-protokollaa. Muistivedoksen lähettämistä on kuvailtu kuviossa 26 näkyvässä prosessikaaviossa. Kuviossa näkyvä React-sovellus viittaa Autovolan käyttöliittymään käyttäjän selaimessa. Python-rajapinta taas on Apache-kontin portissa 8080 toimiva palvelu, joka on rakennettu käyttäen Pythonin Flask-kirjastoa. Jos Analyysi-

kontti on kiireinen, se on todennäköisesti suorittamassa jotain muuta Volatility 3:n lisäosaa johonkin muistivedokseen, eikä ole saatavilla ennen kuin on suorittanut kyseisen prosessin loppuun asti. Tätä vaihetta kuvaillaan tarkemmin luvussa 9.2.3. Perustietoja ovat kuvion 21 taulukossa olevissa raporteissa näkyvät tiedot, kuten käyttöjärjestelmä ja aikaleima.



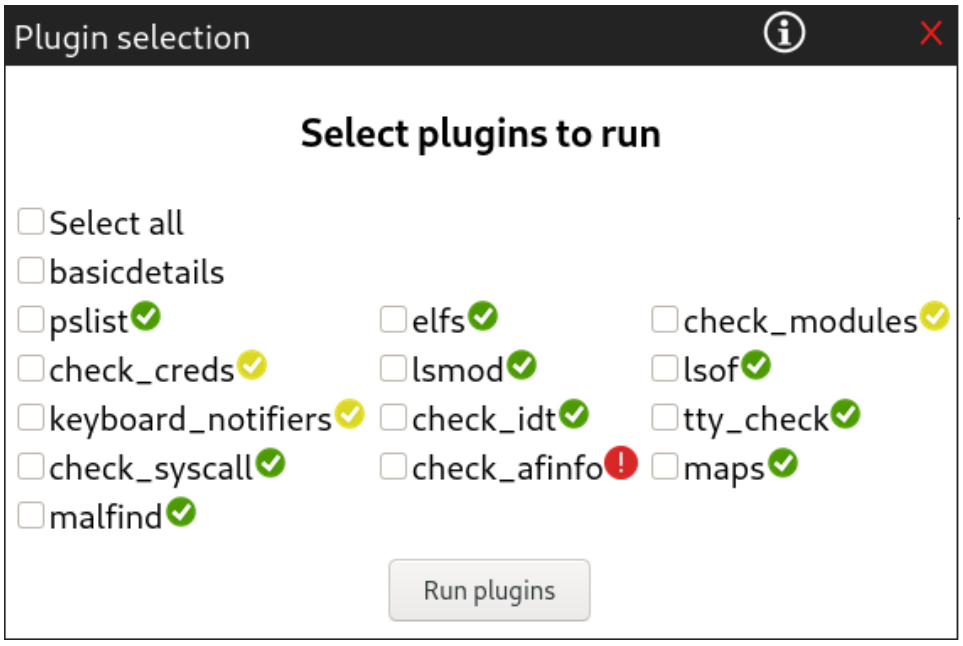

Kuvio 26. Prosessikaavio muistivedoksen käsittelystä

ISF-tiedostojen lataaminen Autovola-palveluun on tiedoston lähetyksen osalta samanlainen kuin kuviossa 26. Kun ISF-tiedosto on onnistuneesti ladattu palvelimelle, jokaiselle Analyysi-kontille lähetetään tästä viesti. Kaikki Analyysi-kontit poistavat tiedostojärjestelmästänsä Volatility 3:n välimuistin, jolloin uudet ja vanhat ISF-tiedostot lisätään välimuistiin, kun kontti seuraavan kerran käyttää Volatility 3:a.

### 9.2.3 Volatility 3 -lisäosien käyttäminen muistivedokseen

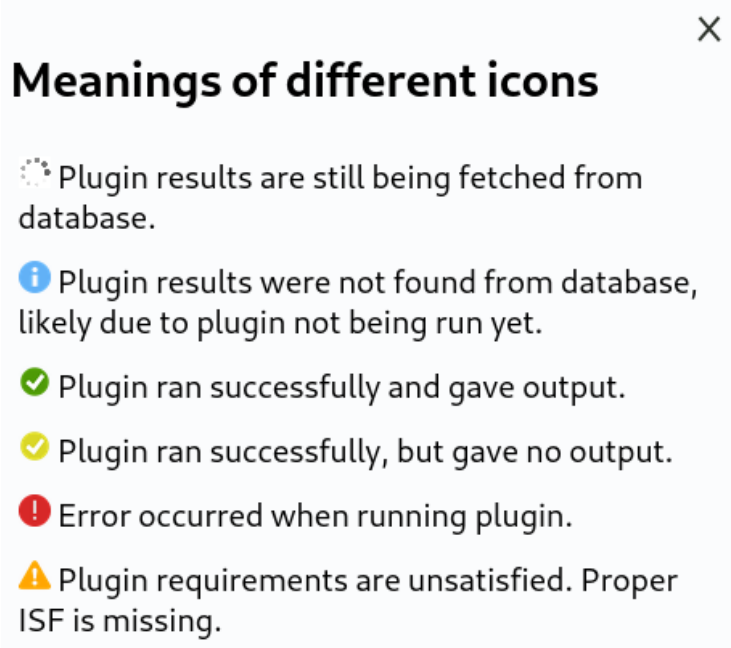
Käyttöliittymästä voidaan määritellä, mitä Volatility 3:n lisäosia mihinkin muistivedokseen suoritetaan. Luvussa 9.2.1 kerrottiin muistivedoksen raporttiin liittyvästä ikkunasta, joka näkyy kuviossa

27. Ikkuna sisältää kaikkien lisäosien nimet, mitä muistivedokseen voidaan ajaa. Kaikkia mahdollisia Volatility 3:n lisäosia ei pystytä suorittamaan Autovolan kautta, koska ne saattavat paljastaa arkaluontoista tietoa, kuten käyttäjien salasanoja. Lisäosat eivät ole samoja Windows- ja Linux-käyttöjärjestelmille, eli ikkunassa näkyvät lisäosat määräytyvät muistivedoksen käyttöjärjestelmän mukaan. Kuviossa 27 näkyvät lisäosat ovat Linux-käyttöjärjestelmälle suunnattuja, eli kyseinen ikkuna kuuluu Linux-raportille.




Kuvio 27. Linux-muistivedoksen Volatility 3 -lisäosat

Kuvion 27 lisäosien nimien perässä on kuvake, basicdetails-lisäosaa lukuun ottamatta. Kuvake kertoo tietoa lisäosasta, kuten onko kyseistä lisäosaa suoritettu muistivedokseen tai onko lisäosaa suoritettaessa tapahtunut jokin virhe. Tarkemmat tiedot kuvakkeiden tarkoituksesta saa painamalla kuvion 27 ylälaidassa oikealla näkyvää valkoista painiketta, joka paljastaa kuviossa 28 näkyvän ikkunan.




Kuvio 28. Lisäosien yhteydessä olevien kuvakkeiden tarkoitus

Prosessikaavio liitteessä 10 kuvastaa tapahtumasarjaa, joka käynnistyy käyttäjän valitessa joukon lisäosia kuvion 27 ikkunasta ja painaessa Run plugins -painiketta sekä, mitä tapahtuu käyttäjän avatessa Plugin selection -ikkunan lisäosien suorittamisen jälkeen. React-sovellus käyttäjän selaimessa lähettää tarvittavat tiedot kohteesta ja lisäosista Python-rajapintaan, joka pyytää aina yhtä Analyysi-konttia suorittamaan yhden lisäosan. Jos Analyysi-kontti on suorittamassa jotain toista lisäosaa pyynnön saapuessa, se vastaa olevansa kiireinen, jolloin seuraavaan Analyysi-konttiin ollaan yhteydessä. Jos Analyysi-kontteja on esimerkiksi vain kaksi, kaikkia lisäosia ei pystytä suorittamaan muistivedokseen samaan aikaan, vaan Analyysi-konttien vapautumista täytyy odottaa.

Kaikki Autovolaan ladattavat muistivedostiedostot nimetään käyttäjän antaman käyttöjärjestelmän ja siitä tietoa sisältävän MongoDB-dokumentin ID-kentän mukaan. Analyysi-kontille syötetään ajettavan lisäosan lisäksi käyttöjärjestelmä ja aiemmin mainittu dokumentin ID, jotta se osaa löytää oikean muistivedoksen, johon lisäosa suoritetaan. ID:tä käytetään myös löytämään muistivedoksen tietoja sisältävä dokumentti, mihin lisäosan tiedot tallennetaan.

Lisäosan dataa varten dokumenttiin luodaan kenttä, joka nimetään lisäosan mukaan. Näin lisäosan tuloksia haettaessa tiedetään, mistä kentästä tieto löytyy. Jos lisäosan vaatimuksia ei pystytä toteuttamaan puuttuvan ISF-tiedoston takia, sitä vastaavaan kenttään dokumentissa kirjoitetaan *UNSATISFIED*. Mikäli lisäosan suorittamisen aikana tapahtuu virhe, dokumentin kenttään kirjoitetaan *ERROR*. React-sovellus muun muassa käyttää kyseisiä arvoja selvittääkseen, mikä kuvake lisäosan yhteydessä käyttäjälle näytetään (ks. kuvio 28).

Kun prosessikaaviossa (ks. liite 10) lisäosat ovat ajettu muistivedokseen ja käyttäjä avaa muistivedoksen Plugin selection -ikkunan (ks. kuvio 27), selaimesta lähetetään Python-rajapintaan kysely, jossa kaikkien muistivedokseen ajettujen lisäosien tulokset pyydetään palauttamaan. Rajapinnalle kerrotaan muistivedoksen käyttöjärjestelmä ja muistivedoksen tietoja sisältävän dokumentin ID-kentän arvo, jotta rajapinta pystyy kohdistamaan tietokantaan oikeanlaisen kyselyn tiedon saavuttamiseksi. Kuten luvussa 9.1.1 kerrotaan, MongoDB-tietokanta sisältää kaksi eri kokoelmaa: *windows_pluginresults* ja *linux_pluginresults*. Python-rajapintaan syötetyn käyttöjärjestelmän perusteella rajapinta osaa pyytää tietokannalta tietoa oikeasta kokoelmasta. Pelkkä dokumentin ID ei riitä, sillä dokumentit ovat jaettu kappaleessa aiemmin mainittuihin kokoelmiin käyttöjärjestelmän perusteella.

Python-rajapinnan noudettua lisäosien tulokset tietokannasta, ne lähetetään käyttäjän selaimelle, joka aiemmin pyysi tietoja rajapinnasta. Prosessin lopussa (ks. liite 10) käyttäjän kuviossa 27 näkyvän Plugin selection -ikkunan lisäosien vieressä olevat kuvakkeet päivittyvät kuviossa 28 näkyvillä kuvakkeilla. Näin käyttäjä näkee, pystyttiinkö kaikki kymmenen prosessikaavion alussa valittua lisäosaa suorittamaan onnistuneesti.

### 9.2.4 Muistivedoksen tulosten tutkiminen

Kuten luvussa 9.2.1 kerrotaan, painamalla kuviossa 21 raportin painikkeista ylintä, käyttäjälle rakennetaan sivu, josta hän pystyy tutkimaan muistivedokseen suoritettujen Volatility 3 -lisäosien tuloksia. Tässä dokumentissa kyseistä sivua kutsutaan analysointisivuksi. Liitteessä 11 näkyy yleisnäkymä Windows-järjestelmästä otetun muistivedoksen analysointisivun sisällöstä. Sivun yläreunassa on palkki, josta käyttäjä pystyy vaikuttamaan kaikkiin sivulla näkyviin lisäosiin. Yläpalkki on kiinnitetty sivuun, joten käyttäjän selatessa sivua alaspäin yläpalkki pysyy näkyvissä koko ajan.

Sivun varsinainen sisältö ovat Volatility 3:n lisäosien tulokset. Jos muistivedokseen on suoritettu jokin lisäosa onnistuneesti, ja kyseinen lisäosa on palauttanut tuloksia, lisäosan tuloksille luodaan oma elementtinsä sivulle. Kuviossa 29 näkyy esimerkki envars-lisäosasta analysointisivulla. Jokaisessa lisäosalaatikossa on yläpalkki, jossa näkyy lisäosan nimi, info-painike ja JSON-latauspainike. Info-painiketta painamalla käyttäjälle paljastetaan ikkuna, jossa selitetään, mitä lisäosa tekee ja mitä taulukon kolumnit tarkoittavat. JSON-painiketta painamalla käyttäjä voi ladata kaiken tiedon taulukosta JSON-muodossa.

envars

| PID | Process | Block | Variable | Value |
|---|---|---|---|---|
| 552 | wininit.exe | 0x1c9afb316f0 | PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC |
| 552 | wininit.exe | 0x1c9afb316f0 | PROCESSOR_ARCHITECTURE | AMD64 |
| 552 | wininit.exe | 0x1c9afb316f0 | PROCESSOR_IDENTIFIER | Intel64 Family 6 Model 158 Stepping 9, GenuineIntel |
| 552 | wininit.exe | 0x1c9afb316f0 | PROCESSOR_LEVEL | 6 |
| 552 | wininit.exe | 0x1c9afb316f0 | PROCESSOR_REVISION | 9e09 |

Show all 5 10 20 50 200 1 2 3 4 5 .. 358

Kuvio 29. Envars-lisäosan tulostetta Windows-muistivedoksen analysointisivulla

Kuvion 29 lisäosalaatikon sisällä olevan taulukon sarakkeiden nimien vieressä olevia ylä- ja alanuolipainikkeita painamalla käyttäjä voi määrittää näytetäänkö taulukon tulokset laskevassa vai nousevassa järjestyksessä jonkun sarakkeen arvojen mukaisesti. Lisäosalaatikon alapalkissa on painikkeita, joiden toiminnoilla voidaan vaikuttaa taulukossa näkyviin riveihin. Taulukossa näkyy vain viisi riviä kaikesta lisäosan datasta. Painamalla alapalkissa näkyvää Show all -painiketta (ks. kuvio 29), kaikki lisäosan rivit tulevat käyttäjän näkyville. Tällöin myös Show all -painikkeen tilalle tulee Hide-painike, jota painamalla näkyvien rivien määrä palaa takaisin viiteen.

Lisäosalaatikon alaosasta pystytään myös määrittelemään, kuinka monta taulukon riviä käyttäjälle näytetään kerralla. Kuviossa 29 Show all -painikkeen oikealla puolella on lukuja 5-200. Lukuja painamalla vaihdetaan käyttäjälle näytettävien rivien määrää. Kuviossa 29 käyttäjälle näytetään viisi riviä dataa, mutta painamalla esimerkiksi lukua 20, käyttäjälle näytettävien rivien määrä nousee 20:een. Aiempaan viiden rivin määrään voidaan palata painamalla lukua viisi.

Käyttäjälle näytettävä data on jaettu sivuihin. Alapalkin (ks. kuvio 29) oikeassa laidassa näkyy, millä sivulla tällä hetkellä ollaan ja mikä on sivujen maksimimäärä. Kuvion taulukossa ollaan sivulla kolme, mikä voidaan havaita luvun kolme tummemmasta väristä. Painamalla lukuja numeron kolme oikealla ja vasemmalla puolella, käyttäjä voi vaihtaa taulukon sivua. Taulukon rivien kokonaismäärä jaetaan taulukossa näkyvien rivien määrällä, mikä vaikuttaa sivujen maksimimäärään. Esimerkiksi kuviossa 29 maksimisivujen määrä on 358, kun rivejä näytetään kerralla viisi. Jos käyttäjälle näytettävien rivien määrä nostetaan 20:een, maksimisivujen määrä pienentyy 90:een.

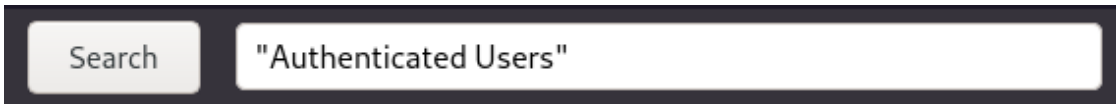
Liitteessä 11 näkyvän yläpalkin elementit ovat jaettu omiin värikoodattuihin laatikoihin kuviossa 30. Elementtien toiminnot kiteytettyinä:

- **Punainen**: Tätä kuvaketta painamalla käyttäjä viedään takaisin raportteja sisältävälle sivulle (ks. kuvio 21).
- **Vihreä**: Alasvetovalikko, josta voidaan vaihtaa kaikkien lisäosalaatikoiden taulukoiden rivimäärä tiettyyn arvoon. Oletuksena kaikki lisäosat näyttävät 20 riviä, mikä on myös alasvetovalikon oletusarvo.
- **Sininen**: Hakupainike ja -palkki, josta käyttäjä voi suodattaa lisäosien tuloksia, mikä vaikuttaa lisäosalaatikoissa näkyvään dataan.
- **Keltainen**: Alasvetovalikko, jossa jokaisen lisäosan nimi on omassa laatikossansa. Sivulla näkyviä lisäosia (ks. liite 11) voi poistaa ja lisätä painamalla samannimistä laatikkoa alasvetovalikosta.
- **Oranssi**: Eri kategorioita sisältävä alasvetovalikko, josta voidaan määritellä, mitä lisäosia käyttäjälle näytetään. Oletuskategoria on *All*, eli käyttäjälle näytetään kaikki lisäosat. Jos kategoriaksi valitaan *Processes*, käyttäjälle näytetään sivulla vain prosesseihin liittyvät Volatilityn lisäosat.
- **Ruskea**: Info-kuvake, jota painamalla käyttäjälle näytetään ikkuna, joka sisältää ohjeistuksen yläpalkin toimintojen käytöstä.


Kuvio 30. Analysointisivun yläpalkki jaettuna osiin

Hakutoiminto on yksi Autovolan tärkeimmistä ominaisuuksista. Toiminnon avulla käyttäjä pystyy suodattamaan lisäosalaatikoissa näkyvää tietoa haluamallaan tavalla sekä hakemaan täsmälleen tietynlaista tietoa. Tämä onnistuu soveltamalla hakusyntaksista löytyviä erilaisia ominaisuuksia.

Kaikkien lisäosien dataa voidaan suodattaa kirjoittamalla hakusana kahden lainausmerkin väliin tai ilman lainausmerkkejä, jolloin hakusana ei saa sisältää välilyöntejä, koska välilyönnit erottaisivat hakusanat toisistaan. Kuviossa 31 näkyvällä hakutermillä haetaan kaikista lisäosalaatikoista merkkijonoa *”Authenticated Users”*. Jos jonkin lisäosalaatikon taulukosta (ks. kuvio 29) löytyy kyseinen merkkijono, merkkijonon sisältävä rivi näytetään käyttäjälle. Jos merkkijonoa ei joltain riviltä löydy, kyseinen rivi poistetaan taulukosta. Jos lisäosalaatikon taulukosta ei löydy ainuttakaan merkkijonoon täsmäävää riviä, lisäosalaatikko poistetaan käyttäjän näkyvistä. Kyseistä käytäntöä noudatetaan kaikissa hauissa. Jos rivi tai koko taulukko ei sisällä haluttua tietoa, se suodatetaan pois käyttäjän näkyviltä.




Kuvio 31. Lainausmerkein ympäröity hakusana analysointisivulla

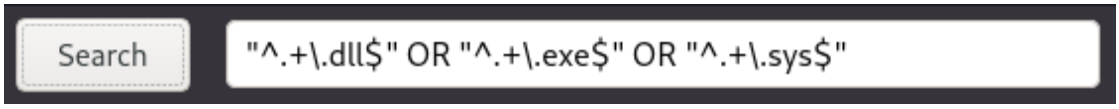
Jos kuviossa 31 ei olisi lainausmerkkejä, haku sisältäisi kaksi eri hakutermiä. Eri hakutermit erotetaan toisistaan välilyönnillä niiden välissä. Jos lainausmerkkejä ei käytettäisi kuvion 31 haussa, taulukkojen rivien täytyisi toteuttaa molemmat hakutermit, eli taulukon riviltä täytyisi löytyä sanat *Authenticated* ja *Users*. Eli pelkkä välilyönti hakutermien välissä tarkoittaa konjunktiota (looginen ”ja”). Hakusanojen välille on myös mahdollista luoda disjunktio (looginen ”tai”) kirjoittamalla sanojen väliin *OR*, niin kuin kuviossa 32. Jos jonkin taulukon rivi sisältää sanan *Authenticated* tai *Users*, rivi näytetään käyttäjälle. Eli vain toisen hakutermeistä täytyy löytyä. Hakutermien määrälle ei ole asetettu maksimirajaa, eli käyttäjä voisi lisätä vielä kuviossa 31 tai 32 näkyviin hakulausekkeisiin lisää hakutermejä.


Kuvio 32. Kaksi disjunktiolla yhdistettyä hakusanaa analysointisivun hakupalkissa

Yksittäiset hakutermit noudattavat säännöllisen lausekkeen (engl. Regular expression) (Regex) merkkijonokieltä. Kielen avulla hakutermien sisälle voidaan kirjoittaa erilaisia rakenteita, joiden avulla taulukoissa olevien rivien dataa voidaan suodattaa kattavammin. Regex-kieli sisältää useita sen syntaksiin kuuluvia erikoismerkkejä, jotka täytyy kommentoida \-merkillä, jos merkit halutaan sisällyttää itse hakusanaan. (Fitzgerald 2012, 1, 5-7.)

Kuviossa 33 on kolme disjunktiolla yhdistettyä hakutermiä. Jokaisen hakutermin alussa on merkkijono ”^.+\.” ja sanan lopussa $-merkki. Nämä ovat Regex-kielen erikoismerkkejä, joiden merkitys on selitettynä alle (Fitzgerald 2012, 6-8, 22):

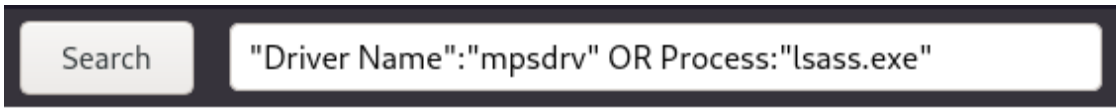
- ^-merkki tarkoittaa merkkijonon alkua.
- .-merkki vastaa yleensä kaikkia merkkejä.
- +-merkki tarkoittaa, että yhden merkin tai useamman joukon merkkejä täytyy esiintyä ainakin kerran.
- Kommentoitu piste ”\.” tarkoittaa pisteen säännölliseen lausekkeeseen kuuluvan merkityksen poistumista, jolloin se on kuin mikä tahansa normaali merkki, joka ei kuulu säännöllisen lausekkeen merkkijonokielen lauseoppiin.
- $-merkki tarkoittaa merkkijonon päättymistä.




Kuvio 33. Regex-kielen soveltamista analysointisivun hakupalkissa

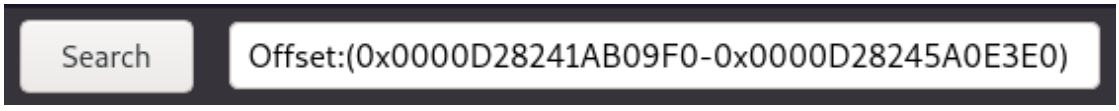
Jotta lisäosalaatikossa olevan taulukon rivi toteuttaisi hyväksyttävästi kuviossa 33 näkyvän hakulausekkeen, ainakin yhden rivin kolumneista tulisi muodostaa merkkijono, joka sisältää vähintään yhden kirjaimen ennen *.dll*-, *.exe*- tai *.sys*-päätettä. Hyväksyttäviä rakennetta noudattavia merkkijonoja olisivat esimerkiksi: *Windows**SysWOW64**ntdll.dll* ja ” ksecdd.sys”. Epäkelvollisia merkkijonoja taas olisivat muun muassa *ufxsynopsy*, *lsass.exec*, *.sys* ja ” ntdll. dll”. Kuvion 33 hakulauseketta voitaisiin käyttää haettaessa lisäosista tiedoston nimiä, joilla on tietty tiedostopääte.

Hakutermejä on mahdollista kohdistaa tiettyihin kolumneihin lisäämällä kolumnin nimen ennen varsinaista hakusanaa. Kuvion 34 hakulausekkeessa on kaksi hakutermiä, jotka ovat yhdistetty disjunktiolla. Ensimmäisessä hakutermissä haetaan merkkijonoa *mpsdrv*, Driver Name -nimisestä kolumnista. Kolumnin nimi sisältää välimerkin, joten koko nimi on ympäröity lainausmerkeillä, sillä muuten välimerkki toimisi nimen sanojen konjunktionaalisena erottimena. Toisessa hakutermissä kolumnin nimeä ei ole ympäröity lainausmerkeillä. Termin merkitys pysyisi kuitenkin samana, vaikka niin olisi tehty. Jos lisäosalaatikon taulukko sisältää toisen tai molemmat hakulausekkeen kolumneista, laatikko voi olla käyttäjälle näkyvillä, jos kolumnia vastaava merkkijono sisältää kolumnin nimen jälkeen määritellyn arvon.




Kuvio 34. Kaksi kolumneja suodattavaa hakutermiä analysointisivun hakupalkissa

Hakutoimintoa voi käyttää myös numeeristen arvojen hakemiseen tietyn alueen sisältä. Numeeriset arvot voivat olla desimaali- tai heksadesimaalimuodossa. Ominaisuutta käytettäessä on myös aina määriteltävä kolumni, josta tietoa haetaan. Kuviossa 35 on esimerkki datan hakemisesta tietyn heksadesimaalialueen sisältä kolumnista Offset. Kolumnin merkkijonon on oltava arvojen *0x0000D28241AB09F0* ja *0x0000D28245A0E3E0* välissä, jotta merkkijonon rivi esitetään käyttäjälle.




Kuvio 35. Numeeristen arvojen hakemista tietyltä alueelta Offset-kolumnista

Tietoa on myös mahdollista suodattaa tuloksista pois käyttämällä negaatiota (looginen ”ei”). Toimintoa voi käyttää lisäämällä miinusmerkin hakutermin eteen. Kuvion 36 ensimmäisessä hakutermissä lisäosalaatikot, joiden taulukot sisältävät kolumnin PID, haetaan rivit, joissa kolumnia vastaava arvo on välillä 1-500. Tämä hakutermi yhdistetään konjunktiolla hakutermiin, joka poistaa

PID-kolumnin tuloksista kaikki rivit, joiden PID-kolumnia vastaava arvo on 108. Negaatiota käyttämällä taulukoiden tuloksista voidaan poistaa tietoa, joka ei ole analyysille olennaista.


Kuvio 36. Negaation soveltamista analysointisivun hakupalkissa

Hakutermejä pystytään ryhmittämään keskenään käyttämällä sisäkkäisiä hakuja. Sisäkkäisessä haussa eri hakutermit ympäröidään suluilla, jolloin sulkujen sisällä olevista hakutermeistä tulee yksi ryhmä. Kuviossa 37 hakutermeistä on luotu kaksi eri ryhmää, jotka ovat yhdistetty keskenään disjunktiolla. Ensimmäisessä ryhmässä on kaksi konjunktiolla yhdistettyä hakutermiä, jotka hakevat tuloksia taulukosta, joka sisältää Module- ja Detail-nimiset kolumnit. Toisessa ryhmässä on myös kaksi konjunktiolla yhdistettyä hakutermiä. Tuloksia haetaan taulukosta, josta löytyy kolumnit LocalPort ja Owner. Owner-kolumnissa on käytetty negaatiota, jolloin kolumnia vastaava rivin merkkijono ei saa olla *System*.


Kuvio 37. Hakutermien ryhmittämistä analysointisivun hakupalkissa

Jos lisäosalaatikon taulukon rivi täyttää toisen kuviossa 37 näkyvien ryhmien ehdoista, rivi näytetään käyttäjälle. Mahdollisuus ryhmittää hakutermejä on hyödyllinen, jos eri lisäosien taulukot sisältävät eri nimisiä kolumneja ja käyttäjä haluaa nähdä erilaista tietoa useista eri lisäosista samaan aikaan. Kaikki hakutoiminnon ominaisuudet ovat dokumentoitu analyysisivulle. Dokumentaation saa esille painamalla kuviossa 30 näkyvää ruskealla merkittyä painiketta.

## 9.3 Lokien kerääminen

Autovola-palvelun Docker-kontit keräävät erilaisista kontin tapahtumista tietoa lokeihin. Jokaisella kontilla on oma lokinsa, josta näkee kontissa tapahtuneita virhetilanteita sekä muuta informaatiota. Lokeja voi tarkastella käyttämällä Dockerin omia työkaluja. (Docker logs n.d.)

Jos yksittäisen kontin, kuten ensimmäisen Analyysi-kontin lokia halutaan tarkastella, voidaan ajaa komento (Docker logs n.d):

```
docker logs autovola_analyse_1
```

Jos taas kaikkien Analyysi-konttien lokidata halutaan nähdä, voidaan ajaa Compose-työkalun komento (Docker-compose logs n.d):

```
docker-compose logs analyse
```

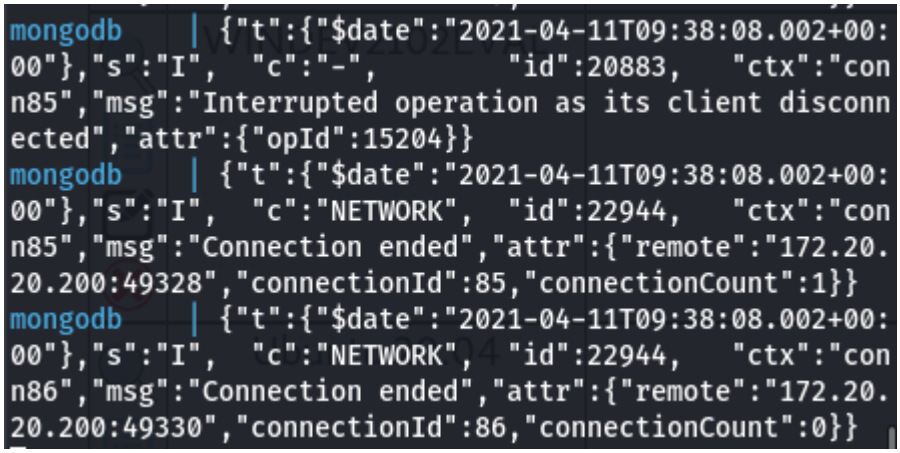
Kuviossa 38 näkyy kahden eri Analyysi-kontin lokitietoa komennon tulosteesta. Molempien Analyysi-konttien nimet ovat värikoodattu, mikä helpottaa niiden lokidatan erottamista toisistaan. Jokainen lokirivi aloitetaan kontin nimellä, jota seuraa lokirivin tietosisällön tyyppi, kuten kuviossa 38 näkyvät INFO ja ERROR. Jos lokirivin tietosisällöllä kuvaillaan kontissa tapahtuvaa normaalia toimintaa, tietosisällön tyyppinä käytetään avainsanaa INFO. Virhetilanteen tapahtuessa lokirivin tietosisältöä taas kuvaillaan avainsanalla ERROR.

```
analyse_1   | INFO - fetching OS details from dump /etc/autovola/dumps/w
indows.60585e59596015ff7a37b3a3.new-dump.raw and determining OS version
analyse_1   | INFO - running plugin <class 'volatility3.framework.interf
aces.plugins.windows.info.Info'> on dump /etc/autovola/dumps/windows.60
585e59596015ff7a37b3a3.new-dump.raw
analyse_1   | INFO - fetching Windows system name from dump /etc/autovol
a/dumps/windows.60585e59596015ff7a37b3a3.new-dump.raw
analyse_1   | INFO - calculating dump /etc/autovola/dumps/windows.60585e
59596015ff7a37b3a3.new-dump.raw size
analyse_1   | INFO - finished running plugin basicdetails on file window
s.60585e59596015ff7a37b3a3.new-dump.raw
analyse_2   | INFO - received request to run plugin check_afinfo on file
 linux.60585fa9596015ff7a37b3b7.5.4.0-58-generic.lime
analyse_2   | INFO - running plugin <class 'volatility3.framework.interf
aces.plugins.linux.check_afinfo.Check_afinfo'> on dump /etc/autovola/du
mps/linux.60585fa9596015ff7a37b3b7.5.4.0-58-generic.lime
analyse_2   | ERROR - Volatility gave this error when running <class 'vo
latility3.framework.interfaces.plugins.linux.check_afinfo.Check_afinfo'
>: StructType has no attribute: vmlinux1!tcp_seq_afinfo.seq_fops
```

Kuvio 38. Analyysi-konttien lokitietoa

Kuvion 38 viimeisellä lokirivillä kuvaillaan toisessa Analyysi-kontissa tapahtunutta virhetilannetta. Kontti on yrittänyt suorittaa Volatility 3:n lisäosaa Linux-muistivedokseen epäonnistuen. Tästä tapahtumasta ja Volatilityn antamasta virheilmoituksesta: *StructType has no attribute: vmlinux1!tcp_seq_afinfo.seq_fops* on lisätty merkintä lokiin. Jos tapahtuneen virhetilanteen syytä halutaan tutkia tarkemmin, lokissa näkyvä virheilmoitus voi kertoa suoraan, mitä on sattunut.

Apache- ja Analyysi-konttien lokirakenne on räätälöity niin, että lokidata sisältäisi konttien tärkeimpiä tapahtumia, jolloin lokidataa on myös vähemmän. MongoDB-kontti käyttää levykuvaansa kuuluvaa MongoDB-palvelun lokirakennetta, joka kirjoittaa lokeihin enemmän dataa alhaisemman tason tapahtumista, kuten yhteyden päättymisestä johonkin IP-osoitteeseen (ks. kuvio 39).




Kuvio 39. MongoDB-kontin lokitietoa

## 9.4 Palvelun käyttöönottaminen

Koko Autovola-palvelun saa käyttöön ajamalla Autovolan GitLab-projektista löytyvän rebuild.sh-nimisen skriptin (ks. kuvio 40), joka sisältää tarvittavat komentorivikomennot palvelun vaatiman ympäristön pystyttämiseen. Jokainen ajettava komento on omalla rivillään tiedostossa, ja komentoja suoritetaan rivi kerrallaan järjestyksessä ylhäältä alas (Blum & Bresnahan 2015, 269-270). Tiedostoa voidaan käyttää uuden ympäristön rakentamiseen sekä poistamaan ensin vanha ympäristö ja sitten rakentamaan uusi ympäristö vanhan tilalle.

```
#!/bin/bash

docker-compose stop
docker-compose rm -f
docker rmi autovola/apache:latest autovola/analyse:latest
autovola/mongo:latest mongo:latest
docker load --input containers/images/mongodb.tar
docker tag mongo:latest autovola/mongo:latest
docker load --input containers/images/debian_apache.tar
docker load --input containers/images/ubuntu_analyse.tar
docker-compose up --build -d
```

Kuvio 40. Rebuild.sh-tiedoston sisältö

Skripti (ks. kuvio 40) yrittää ensimmäiseksi poistaa vanhan ympäristön kontit ja levykuvat, vaikka Autovola-ympäristöä ei järjestelmään olisi aiemmin asennettukaan. Kun mahdollinen vanha ympäristö on poistettu, skripti lataa kaikkien Docker-palveluiden levykuvat järjestelmään *docker load* -alkuisilla komennoilla (Docker load n.d). MongoDB-kontin levykuvalle luodaan uniikki tunniste, joka vastaa docker-compose.yml-tiedostossa olevaan image-avaimen arvoon (ks. kuvio 15). Apache- ja Analyysi-konteilla on jo palveluita kuvaavat merkinnät ja molempien palveluiden docker-compose-konfiguraation image-avainta käytetään myös kyseisten palveluiden Dockerfile-tiedostoissa oleviin FROM-avainsanan arvoihin (ks. liitteet 7 ja 9). Näin Docker osaa varmasti käyttää oikeita levykuvia rakentaessaan kontteja (Docker tag n.d). Skriptin viimeinen komento rakentaa levykuvat valmiiksi ja käynnistää Autovola-ympäristön kontit taustapalveluiksi järjestelmään (Docker-compose up n.d).

# 10 Toteutuksen käytännöllinen testaaminen

## 10.1 Järjestelmän asentaminen

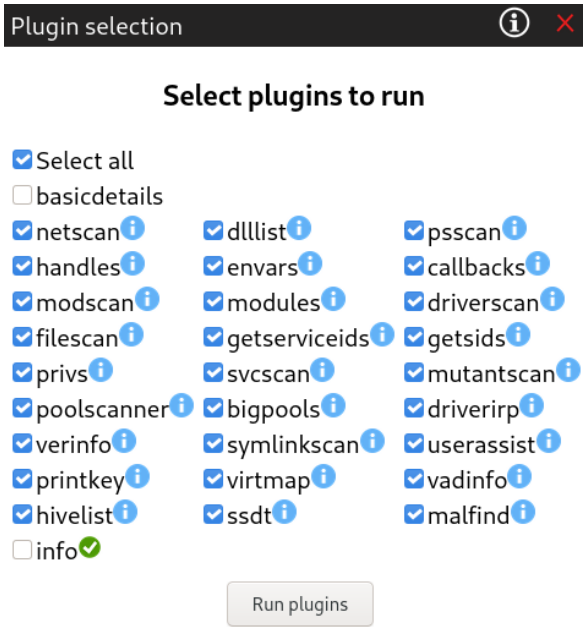
Autovola-palvelu pystytettiin Linux-järjestelmään, jonka tarkka versio oli *Linux 5.10.0-kali3-amd64 #1 SMP Debian 5.10.13-1kali1 (2021-02-08) x86_64 GNU/Linux*. Järjestelmää ylläpidettiin VirtualBoxin virtuaalikoneessa, jolle oli varattu 6144 Mt DDR4-keskusmuistia ja neljä virtuaalista prosessorin ydintä. Isäntäkoneen prosessorina toimi Intel i7-7700k. Autovola rakennettiin järjestelmään ajamalla kuviossa 40 näkyvä skripti. Skripti rakensi järjestelmään Apache- ja MongoDB-kontin lisäksi kaksi Analyysi-konttia. Tähän kului aikaa 6 minuuttia ja 53 sekuntia.

## 10.2 Muistivedoksen analysointia

Järjestelmään ladattiin kuviossa 25 näkyvää lomaketta käyttäen Windows 10 -käyttöjärjestelmästä otettu muistivedos. Muistivedoksen koko oli noin 4,83 gigatavua ja sen lataamiseen kului aikaa 2 minuuttia ja 22 sekuntia. Muistivedokselle löytyi ISF-tiedosto valmiiksi Autovolasta, joten erillistä ISF-tiedostoa ei tarvinnut ladata Autovolaan. Ennen muistivedoksen kaappausta, Windows 10 -järjestelmässä oli suoritettu WannaCry-kiristyshaittaohjelman (engl. ransomware) komponentti tasksche.exe, jonka 256-bittinen SHA-2-tarkistussumma on *ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA*.

Tasksche.exe muun muassa kirjoittaa laitteen levyasemalle lisää haittaohjelman komponentteja, suorittaa näitä komponentteja ja salaa tiedostojärjestelmästä tiedostot, joilla on tietty tiedostopääte, kuten *.txt* tai *.docx*. Alkuperäiset tiedostot poistetaan jättäen jäljelle vain salatut tiedostot, minkä jälkeen käyttäjälle näytetään lunnasvaatimus @wanadecryptor@.exe-nimisen ohjelman käyttöliittymässä. Muistivedos otettiin vasta tiedostojen salaamisen jälkeen lunnasvaatimuksen ilmaannuttua näkyville. (Berry, Homan & Eitzman 2017; Kumar, Ben-Othman & Srinivasagan 2018, 1201-1202.)

Muistivedokseen ajettiin kaikki mahdolliset Autovolan käyttämät Volatility 3:n lisäosat (ks. kuvio 41). Info- ja basicdetails-lisäosat suoritetaan automaattisesti jokaiseen Autovola-palveluun ladattuun Windows-muistivedokseen, joten lisäosat jätettiin valitsematta. Kaikkien lisäosien suorittamiseen kului yhteensä aikaa 28 minuuttia ja 5 sekuntia kahdelta Analyysi-kontilta. Vertailun vuoksi samat lisäosat ajettiin myös eri ajankohtana isäntäkoneella käyttäen Volatility 3:n komentoriviohjelmaa. Lisäosien tulokset putkitettiin JSON-muodossa suoraan tiedostoihin ja koko prosessiin kului aikaa 19 minuuttia 32 sekuntia.




Kuvio 41. Windows-muistivedoksen Volatility 3 -lisäosat

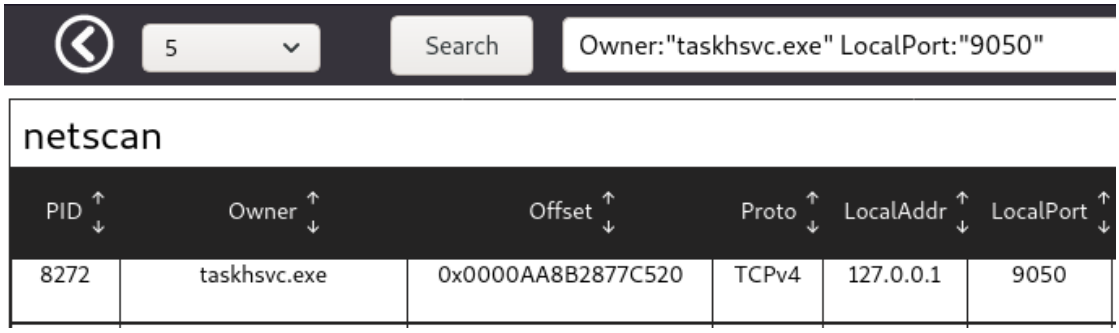
Vadinfo-nimisen lisäosan suoritus onnistui, mutta lisäosan dataa ei pystytty lähettämään MongoDB-tietokantaan, koska lähetettävän tiedon koko ylitti 16 Mt:n rajan (ks. kuvio 42). MongoDB tukee enintään 16 Mt:n BSON-dokumentteja, jotta keskusmuistin käyttö ei kasva liian suureksi. MongoDB tarjoaa myös rajapinnan nimeltään GridFS, suurien dokumenttien varastoimiseen, mutta kyseistä rajapintaa ei käytetä Autovolassa. (MongoDB Limits and Thresholds n.d.)

```
analyse_2  | 2021-04-17T11:21:00.978100664Z ERROR - sending data to database autovola
table windows_pluginresults document with ID 607abacdb2a301e47828ca5d caused error: BS
ONObj size: 19759888 (0x12D8310) is invalid. Size must be between 0 and 16793600(16MB)
 First element: _id: ObjectId('607abacdb2a301e47828ca5d'), full error: {'index': 0, 'c
ode': 10334, 'errmsg': "BSONObj size: 19759888 (0x12D8310) is invalid. Size must be be
tween 0 and 16793600(16MB) First element: _id: ObjectId('607abacdb2a301e47828ca5d')"}
```

Kuvio 42. Lokimerkintä MongoDB-virheilmoituksesta

Vadinfoa lukuun ottamatta, muiden lisäosien tulokset saatiin lähetettyä tietokantaan onnistuneesti. WannaCry-haittaohjelmaan liittyy useita eri IoC:eja, kuten IP-osoitteita, verkkotunnuksia, tiedostoja ja rekisterimerkintöjä. Yksi näistä on WannaCryn Tor-palvelin taskhsvc.exe, joka käyttää järjestelmän TCP-porttia 9050. WannaCry suorittaa taskhsvc.exe-tiedoston ja käyttää tiedostosta käynnistynyttä Tor-palvelinta ottaakseen yhteyttä CnC-palvelimeen, jonne lähetetään tietoa käyttäjän Windows-järjestelmästä. (Berry ym. 2017; Kao & Hsiao 2018, 160-161.)

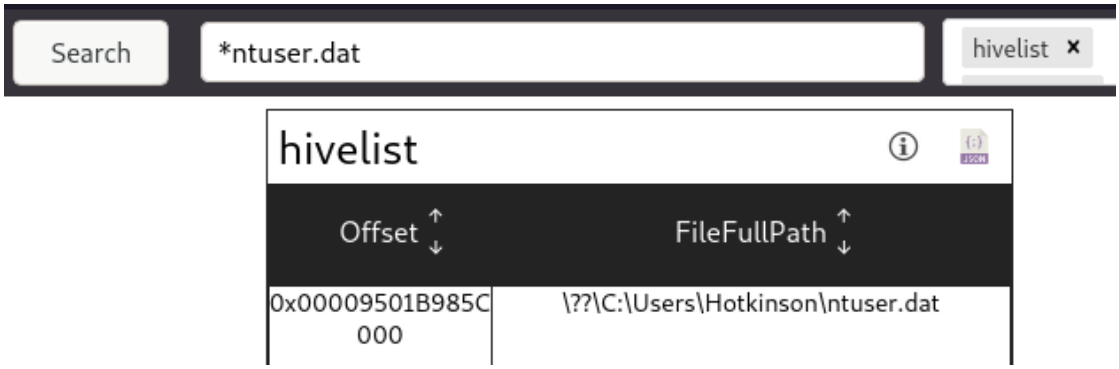
Koska haitallisen prosessin nimi ja sen käyttämä portti on tiedossa, muistivedoksen analysointisivulta voidaan tarkistaa, löytyykö taskhsvc.exe-prosessia vedoksesta. Kuviossa 43 analysointisivun hakutoiminnon avulla on haettu kyseistä prosessia ja siihen liittyvää porttia 9050, mikä on palauttanut yhden osuman netscan-lisäosasta. Netscan-lisäosa palauttaa listan objekteista, jotka liittyvät verkkoyhteyksiin (Volatility 3 Documentation 2021, 508-509).




Kuvio 43. Hakuparametrit ja netscan-lisäosan tulostetta

WannaCry lisää merkinnän Windows-järjestelmän rekisterin polkuun *HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*, mikä varmistaa, että haittaohjelman komponentti tasksche.exe käynnistetään käyttäjän kirjautuessa sisään (Berry ym. 2017). Tätä keinoa kuvailtiin tarkemmin luvussa 7.4. Juuriavain HKCU:n sisältö sijaitsee kokonaisuudessaan profiilidataa sisältävän ntuser.dat-nimisen tiedoston sisällä ja jokaisella Windows-käyttäjällä on oma ntuser.dat-tiedostonsa (Russinovich ym. 2012, 281, 294).

Tiedostoa voidaan etsiä muistivedoksesta käyttämällä analyysisivun hakutoimintoa. Kuviossa 44 kaikista lisäosista on etsitty *ntuser.dat*-päätteeseen päättyvää arvoa. Volatilityn HiveList-lisäosasta on löytynyt yksi hakuun sopiva arvo. HiveList-lisäosa etsii juuriavaimia ja niiden aliavainten ryppäitä muistista. Löydettyjen ryppäiden muistipaikka (engl. offset) sekä sijainti tiedostojärjestelmässä palautetaan käyttäjälle. (Volatility 3 Documentation 2021, 470.)




Kuvio 44. Ntuser.dat-tiedoston muistipaikka HiveList-lisäosassa

Kun juuriavaimen sijainti muistissa tiedetään, sen aliavainten merkintöjä voidaan hakea Volatilityn PrintKey-lisäosalla. PrintKeyllä voidaan myös hakea juuriavaimen aliavainta tietämättä juuriavaimen muistipaikkaa, mutta tällöin PrintKey-lisäosa palauttaa kaikki aliavaimen polkuun täsmäävät polut, eli tuloksia voidaan näyttää usean eri juuriavaimen alta. (Volatility 3 Documentation 2021, 474-475.) Autovola ei tue juuriavainten aliavainten merkintöjen hakemista PrintKeyllä, mutta Linux-järjestelmässä Volatility 3:n komentorivityökalulla aliavaimet voidaan hakea ajamalla komento (Volatility 3 Documentation 2021, 474-475):

```
python3 vol.py -f ~/wannacry.raw windows.registry.printkey.PrintKey --offset
0x00009501B985C000 --key "Software\Microsoft\Windows\CurrentVersion\Run"
```

Komennon eri osat selitettynä (Volatility 3 Documentation 2021, 474-475):

- **vol.py**: Volatility 3:n Python-työkalun nimi.
- **-f ~/wannacry.raw**: Polku muistivedoksen sijaintiin tiedostojärjestelmässä.
- **windows.registry.printkey.PrintKey**: Lisäosan luokkapolku.
- **--offset 0x00009501B985C000**: ntuser.dat-tiedoston muistipaikka (ks. kuvio 44).
- **--key "Software\Microsoft\Windows\CurrentVersion\Run"**: Polku rekisteriavaimeen, jonka sisältö käyttäjälle tulostetaan.

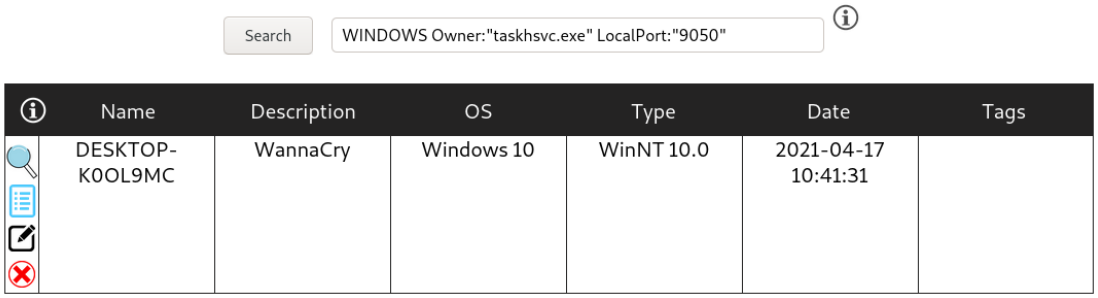
Kuviossa 45 näkyy kyseisen komennon tuloste. Avaimen alta löytyi kaksi merkintää, joista jälkimmäinen on WannaCryn tekemä. WannaCryn luoman merkinnän nimi on *sicvszmqbf870*. Merkit ovat satunnaisesti generoitu, mutta ensimmäiset kymmenen merkkiä ovat aina pieniä kirjaimia ja kolme viimeistä merkkiä taas numeroita, mikä helpottaa merkinnän tunnistamisessa. Merkintä sisältää arvon *C:\Users\Hotkinson\Desktop\oppari_maltsut\Ransomware.WannaCry\tasksche.exe*.

Arvo on polku tasksche.exe-tiedoston sijaintiin tiedostojärjestelmässä. Windows-järjestelmä käyttää tätä arvoa käynnistääkseen tasksche.exe-prosessin aina kun käyttäjä kirjautuu sisään, kuten luvussa 7.4 mainittiin. (Berry ym. 2017.)

```
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
Last Write Time Hive Offset     Type    Key     Name    Data    Volatile

2021-04-17 10:29:09.000000      0x9501b985c000  REG_SZ  \??\C:\Users\Hotkinson\
ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Run OneDrive        ""C:\U
sers\Hotkinson\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background"
        False
2021-04-17 10:29:09.000000      0x9501b985c000  REG_SZ  \??\C:\Users\Hotkinson\
ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Run sicvszmqbf870   ""C:\U
sers\Hotkinson\Desktop\oppari_maltsut\Ransomware.WannaCry\tasksche.exe""  False
```

Kuvio 45. Volatility 3:n PrintKey-lisäosan tuloste

WannaCry-haittaohjelman komponentteja tai muita IoC:eja voidaan etsiä kaikista muistivedoksista käyttämällä Reports-sivun hakutoimintoa. Kuviossa 46 kaikista Windows-muistivedoksista on haettu portissa 9050 toimivaa taskhsvc.exe-nimistä prosessia käyttämällä kuviossa 43 näkyviä hakuparametrejä. Haku on palauttanut yhden muistivedoksen, jota tässä luvussa on analysoitu.




Kuvio 46. Reports-sivulla suoritetun haun palauttama muistivedos

# 11 Pohdinta

## 11.1 Tutkimusongelmaa lähestyminen

Tutkimusongelmana oli JYVSECTECiltä puuttuva muistiforensiikkaa automatisoiva järjestelmä. Ongelmaa lähestyttiin tutkimuskysymysten avulla, miettien käyttäjäystävällisyyttä, käytettäviä teknologioita, raportointia, skaalautuvuutta ja mahdollisimman monen muistiforensiikkaan sisältyvän prosessin automatisointia. Toteutettavaksi konstruktioksi valittiin järjestelmä, joka toimisi muistivedosten säilytys- ja analysointipaikkana. Kyseisen verkkopohjaisen järjestelmän katsottiin tarjoavan tiettyjä etuja tutkimusongelmaan ja -kysymyksiin liittyen.

Tietoverkkoon yhdistetyllä järjestelmällä voi olla useampi käyttäjä samaan aikaan. Käyttäjä voi syöttää palveluun muistivedoksen, jota muut käyttäjät voivat myös analysoida palvelussa, lataamatta muistivedosta omalle tietokoneellensa jotain toista kautta. Järjestelmän verkkosovellusta voidaan käyttää selaimesta käsin, eikä koko palvelua tarvitse ladata erikseen, kuten tilanteessa, jossa palvelua pystyisi käyttämään ainoastaan paikallisesti. Verkkosovelluksen tarjoama graafinen käyttöliittymä on myös käyttäjäystävällisempi ja visuaalisesti havainnollistavampi verrattuna esimerkiksi komentoriviohjelmaan. Verkkosovelluksen katsottiin myös tarjoavan muistivedoksista saatavien tulosten raportointiin hyvän pohjan, sillä verkkosivun sisältöön ja tyyleihin voidaan vaikuttaa tavalla, jolla raportista saadaan käyttäjän tarpeisiin mukautuva, mikä helpottaa raportin sisällön tulkintaa.

Muitakin vaihtoehtoja harkittiin, kuten edellisessä kappaleessa mainittua komentoriviohjelmaa. Komentoriviohjelma ei vaatisi palvelinta tai ylimääräisiä resursseja sen jatkuvaan ylläpitämiseen, sillä se käyttäisi resursseja vain suorituksensa aikana. Ohjelma myös vaatisi vähemmän ulkoisia riippuvuuksia, kuten selaintukea tai graafista käyttöliittymää. Tämän toteutustavan ongelmana oli muistivedoksista saadun raportin esittäminen käyttäjälle. Perusperiaate olisi ollut sama kuin verkkosovelluksessa. Volatility 3:n lisäosia olisi suoritettu muistivedokseen ja niiden tulokset olisivat tallennettu joko tietokantaan tai käyttäjän tiedostojärjestelmään. Tämän jälkeen käyttäjä olisi komentoriviohjelmaa hyödyntäen voinut hakea lisäosien tuloksia. Tulosten suodattaminen olisi kuitenkin vaatinut käyttäjältä ylimääräisiä ponnisteluja ja komentoriviohjelmien käyttöön liittyviä taitoja, mikä olisi tehnyt ohjelman käyttämisestä haasteellisempaa ja analysoinnista hitaampaa. Komentoriviohjelman rakentaminen olisi kuitenkin ollut kontteihin pohjautuvaa verkkosovellusta

nopeampaa. Jos ohjelmasta olisi taas haluttu usealle käyttäjälle skaalautuva, olisi se vaatinut ulkoista tietokantapalvelinta, mikä olisi toiminut muistivedoksista saadun tiedon säilytyspaikkana.

Kolmas vaihtoehto olisi ollut rakentaa ohjelmatiedosto, jossa olisi ollut graafinen käyttöliittymä. Tämä olisi vastannut edellisessä kappaleessa mainitun komentoriviohjelman ongelmaan, jossa raportin esittäminen ja suodattaminen olisi ollut haastavaa käyttäjälle. Sovelluksen käyttöliittymästä olisi voitu rakentaa samantapainen kuin Autovolan verkkosovelluksesta. Paikallisen sovelluksen käyttäminen olisi kuitenkin vaatinut sen kääntämistä usealla eri käyttöjärjestelmälle, minkä katsottiin tuovan ylimääräistä työtä. Verkkosovelluksen kohdalla mahdolliset selainpohjaiset eriäväisyydet voidaan korjata muokkaamalla sovelluksen koodia, jolloin siitä ei tarvitse rakentaa useaa eri versiota. Ohjelmatiedostoon perustuvan sovelluksen ei katsottu tuovan erityisiä hyötyjä verrattuna verkkosovellukseen, minkä johdosta verkkosovelluksella toteutettavaa graafista käyttöliittymää ehdotettiin toimeksiantajalle, joka hyväksyi toteutustavan.

## 11.2 Saadut tulokset

Edellisessä luvussa mainitun harkinnan pohjalta päädyttiin rakentamaan Docker-konttien päällä toimiva järjestelmä nimeltään Autovola. Järjestelmään kuuluu muistivedoksista saatua tietoa varastoiva MongoDB-kontti, käyttäjän valitsema määrä muistivedoksia analysoivia Analyysi-kontteja ja järjestelmän käyttöliittymän sisältävä Apache-kontti.

Valmiin toteutuksen käytännön testausta käsiteltiin luvussa 10. Järjestelmän asentaminen vei 6 minuuttia ja 53 sekuntia. Osittain asennuksen kestoon vaikuttivat Dockerfile-tiedostoissa (ks. liitteet 7 ja 9) olevat komennot, jotka on suoritettava asennuksen yhteydessä. Komennoissa suoritettavia toimintoja, kuten eri hakemistorakenteiden luomista, olisi voitu sisällyttää valmiisiin levykuviin, jolloin komentoja ei olisi tarvinnut enää suorittaa Dockerfileissä. Näin ei kuitenkaan haluttu toimia, koska järjestelmän käyttäjille haluttiin antaa mahdollisimman paljon valtaa vaikuttaa konttien rakennusprosessiin. Levykuvia voidaan aina muokata jälkeenpäin, jos niihin halutaan lisätä jotain ylimääräistä, mitä ei haluta enää Dockerfileissä suoritettavan. Levykuviin oli suoritettu vain komennot, jotka vaativat internet-yhteyden, koska järjestelmää tullaan käyttämään ympäristössä, jossa internet-yhteyttä ei välttämättä ole saatavilla ja asennuksen vaatima internet-yhteys voisi aiheuttaa ongelmia.

4,83 gigatavun kokoisen muistivedoksen lataaminen Autovolaan vei 2 minuuttia ja 22 sekuntia, kuten luvussa 10.2 mainittiin. Prosessi voisi olla nopeampi, jos esimerkiksi luvussa 8.1.3 mainitun Apachen ja NGINXin vertailun päätteeksi olisi päädyttä valitsemaan NGINX, joka vaikutti olevan näistä kahdesta suorituskykyisempi WWW-palvelin. Myös HTTP-rajapintojen rakentamiseen käytettävää verkkosovelluskehystä valittaessa (ks. luku 8.1.5) olisi voitu tehdä enemmän vertailuja eri kehysten välillä, ennemmin kuin valita ensin ohjelmointikieli ja sitten vasta kehys, joka on rakennettu kyseisellä ohjelmointikielellä. Valintoja tehdessä henkilökohtainen mieltymys teknologioihin ja mukavuudenhaluisuus saivat ehkä liikaa arvoa.

Windows-muistivedokseen käytettyjen lisäosien suoritus vei Autovolalta aikaa 28 minuuttia ja 5 sekuntia, kun taas Volatility 3:n komentoriviohjelmalla näiden lisäosien suorittamiseen kului 19 minuuttia ja 32 sekuntia (ks. luku 10.2). Autovolan aikaan sisältyy kuitenkin myös lisäosien tulosten lähettäminen tietokantaan sekä joidenkin lisäosien antamien tulosten muokkaaminen MongoDB:lle sopivaksi ja Apache-palvelimesta saapuvien käskyjen viive. Aikaero on silti huomattava. Autovolassa Volatility 3:n lisäosien suorittamisen nopeuteen voidaan vaikuttaa muuttamalla Analyysi-konttien ja konteille annettavien resurssien määrää. Volatility-kirjaston koodia olisi voitu mahdollisesti optimoida nopeuden kerryttämiseksi, mutta se olisi vaatinut paljon aikaa ja perehtymistä, eikä toiminnalla olisi välttämättä edes saavutettu haluttuja tuloksia.

Testaamisen aikana tapahtui virhe lisäosan tuloksia lähetettäessä tietokantaan, koska lähetettävän datan määrä oli liian suuri (ks. kuvio 42). MongoDB:n dokumenttikohtaisiin rajoitteisiin olisi pitänyt kiinnittää enemmän huomiota järjestelmän kehitysvaiheessa, jotta virhetilanteelta olisi vältytty tai siihen olisi voitu varautua. Virheeseen johtava tilanne vaikuttaisi kuitenkin olevan harvinainen ja sen sattuessa käyttäjä voi halutessaan käyttää Volatility 3:n komentorivityökalua lisäosan suorittamiseen.

Toimeksiantaja antoi valmiista tuotteesta ja koko kehitysvaiheeseen liittyvästä prosessista palautetta. Palautteen kritiikistä suurin osa liittyi järjestelmän käyttöönottamiseen liittyviin seikkoihin. Järjestelmän vaatimien resurssien, kuten levytilan tai keskusmuistin määrää ei ollut kirjattu dokumentaatioon. Dokumentaatiosta olisi myös toivottu löytyvän muita perusasioita, kuten palvelun vaatima Docker-versio sekä arvio järjestelmän rakentumiseen kuluvasta ajasta ja API-

dokumentaatio. API-dokumentaatiosta jouduttiin luopumaan opinnäytetyön aikarajan tullessa täyteen, koska siitä ei koettu saavan jäljellä olevan ajan puitteissa asianmukaista. Aiemmin kappaleessa mainitut perusasiat olisi kuitenkin ollut hyvä kirjoittaa projektin GitLab-repositorioon, jotta järjestelmään liittyvät oleelliset tiedot olisivat olleet käyttäjälle selkeästi näkyvillä.

Toimeksiantaja huomasi järjestelmää ensimmäistä kertaa käytettäessä kaksi virhetilannetta, joiden takia Windows-muistivedoksia ei pystytty lataamaan palveluun ja tiettyä muistivedosta ei pystytty analysoimaan. Ongelmat vaikuttivat sovelluksen käyttökokemukseen niin paljon, että korjasin toimeksiantajan pyynnöstä virhetilanteita aiheuttavat ohjelmointivirheet saman päivän aikana niiden huomaamisesta. Tilanteen syntymiseen vaikutti henkilökohtaisen ohjelmistotestausosaamisen puute. Ohjelmointivirheiltä olisi voitu välttyä, jos järjestelmälle olisi suoritettu enemmän testausta sen kehitysvaiheessa. Käyttötestausta olisi myös voitu suorittaa jonkun toisen henkilön toimesta, jolloin testausprosessiin olisi saatu enemmän näkökulmia.

Toimeksiantaja oli tyytyväinen siihen, että palvelut rakennettiin Dockerin varaan ja sovelluskehityksen prosessi onnistui hänen mielestään hyvin. Toimeksiantaja koki, että hänen mielipiteensä ja toiveensa järjestelmän kehitykseen liittyen otettiin huomioon ja lopputuloksesta tuli toivotunlainen. Erikseen toimeksiantaja mainitsi järjestelmän hyviksi puoliksi sen käytettävyyden ja skaalautuvuuden, joka mahdollistaa usean käyttäjän käyttää järjestelmää samanaikaisesti sekä käyttöliittymästä löytyvät käyttöohjeet kaikkiin saatavilla oleviin toimintoihin. Autovola on tällä hetkellä sisäisessä käytössä toimeksiantajan organisaatiossa. Järjestelmää käytetään muun muassa JYVSECTECin järjestämissä kyberharjoituksissa ja työvälinekoulutuksissa. Ulkopuoliset toimijat voivat myös käyttää järjestelmää JYVSECTECin tarjoamien palveluiden kautta. Autovolaa tullaan kehittämään toimeksiantajan toimesta järjestelmään liittyvien tarpeitten ja käyttökokemusten perustella.

Tulosten luotettavuutta kyseenalaistavat tässä luvussa mainitut ohjelmointitestaukseen liittyvät puutteet. Autovolan komponenteista voi vielä jälkeenpäin löytyä ohjelmointivirheitä tai muita käytettävyyteen negatiivisesti vaikuttavia asioita, joita järjestelmää suunniteltaessa ja kehitettäessä ei otettu huomioon. Ohjelmointivirheet voivat kuitenkin olla helposti korjattavissa, mutta järjestelmän logiikkaan tai teknologioihin liittyvät puutteet saattavat vaatia suurempia muutoksia koko kokonaisuuteen. Kuten useiden luvun 8.1 alilukujen teknologioiden perusteluissa mainitaan, teknologian valintaan vaikutti monesti sen helppokäyttöisyys, suosio ja käyttäjämäärät. Jos teknologioihin

liittyviä ongelmia ilmenee myöhemmin, toimeksiantajalla pitäisi olla hyvä mahdollisuus löytää ongelman ratkaisemiseen tukea muilta samaa teknologiaa käyttäviltä henkilöiltä tai yrityksiltä erilaisten internetin tiedonjakopalstojen kautta. Jos teknologioita valittaessa muille ominaisuuksille olisi annettu enemmän merkitystä, niihin liittyvien ongelmatilanteiden ratkaisemiseen voitaisiin joutua käyttämään huomattavasti enemmän aikaa.

## 11.3 Opinnäytetyön tavoitteet

Työn tavoitteita kuvailtiin luvun 1 johdannossa. Yhtenä tavoitteena oli kehittää tuotteeseen toimintoja, joiden avulla käyttäjät pystyvät tekemään yhteistyötä toistensa kanssa. Kuten luvussa 11.1 mainitaan, järjestelmän skaalautuminen usealla käyttäjälle katsottiin tärkeäksi ominaisuudeksi. Autovola toimii keskitettynä arkistona kaikille muistivedoksille, jolloin ne ja Volatility 3:n lisäosien tulokset löytyvät samasta paikasta. Kaikki sovelluksen käyttäjät pääsevät käsiksi kaikkiin muistivedosten raportteihin, ja jos yksi käyttäjä lataa palveluun jonkin muistivedoksen tarvitseman ISF-tiedoston, muiden käyttäjien ei sitä enää tarvitse tehdä. Tavoitteen voidaankin katsoa toteutuneen, sillä järjestelmä edesauttaa käyttäjien välistä yhteistyötä.

Tavoitteena oli myös muistiforensiikan prosessien nopeuttaminen tuotteen avulla. Kuten luvun 11.2 tuloksista käy ilmi, Volatility 3:n lisäosien suorittaminen on huomattavasti nopeampaa komentoriviohjelmalla kuin Autovolalla. Myös muistivedoksen lataaminen Autovolaan vie aikaa. Jos yksittäisen käyttäjän on tarkoitus suorittaa nopeaa tutkimusta muistivedokselle ja ajaa siihen joitain Volatility 3:n lisäosia, on todennäköisesti parempi käyttää vain Volatilityn komentoriviohjelmaa. Jos taas odotettavissa on pitkäkestoinen analyysi yhdestä tai useasta muistivedoksesta usean käyttäjän suorittamana, Autovolan käyttäminen analyysin tukena voi olla hyödyllistä ja aikaa säästävää. Vain yhden käyttäjistä tarvitsee ladata muistivedokset Autovolaan, jolloin kaikki käyttäjät pääsevät suorittamaan Autovolan kautta lisäosia muistivedoksiin ja analysoimaan lisäosien tuloksia. Lisäosien tuloksia pystytään myös lataamaan muistivedoksen raportista, kuten luvussa 9.2.4 mainitaan, jolloin käyttäjien ei tarvitse erikseen suorittaa lisäosia komentoriviohjelmaa käyttäen, jos he haluaisivat esimerkiksi tallentaa tuloksia omaan järjestelmäänsä. Käyttäjät kykenevät myös hakemaan samoja IoC:eja kaikista Autovolaan ladatuista muistivedoksista Reports-sivun hakutoiminnon avulla, kuten luvusta 9.2.1 käy ilmi.

Autovolalla voidaan nopeuttaa joitain muistiforensiikan prosesseja, mutta kuten edellisen kappaleen esimerkeistä voidaan todeta, se riippuu paljon käyttötapauksesta. Autovolalla ei voida korvata Volatility 3:n komentoriviohjelmaa, koska sillä ei esimerkiksi pystytä suorittamaan kaikkia Volatility 3:n lisäosia, kuten luvussa 9.2.3 mainitaan. Autovolan kautta ei myöskään pystytä antamaan lisäosille parametreja, minkä johdosta joidenkin lisäosien kaikkia ominaisuuksia ei pystytä hyödyntämään. Tavoite prosessien nopeuttamiseen liittyen toteutuu osittain, koska joitain prosesseja Autovolalla voidaan nopeuttaa, kuten käyttäjien välistä yhteistyötä. Järjestelmällä kyettäisiin tehostamaan useampiakin muistiforensiikkaan liittyviä prosesseja, kunhan siihen lisättäisiin luvussa 11.4 mainittavia ominaisuuksia.

Kaikista tavoitteista tärkein oli käyttövalmiin tuotteen rakentaminen JYVSECTECille, jota myös käytetään organisaation toimesta. Jos tuote olisi todettu käyttökelvottomaksi tai turhaksi, muilla tavoitteilla ei olisi ollut paljoa merkitystä, koska järjestelmää ei välttämättä koskaan tultaisi hyödyntämään. Järjestelmä on kuitenkin tällä hetkellä JYVSECTECillä käytössä, kuten luvussa 11.2 mainitaan. Luvussa 2 esitetyn markkinatestin perusteella valmista konstruktiota voidaan arvioida kolmella eri tasolla. Autovola läpäisee ensimmäisen tason, koska se on kohdeorganisaatiolla käytössä. Toinen taso kuitenkin vaatisi järjestelmän käyttöönottoa useassa eri organisaatiossa, mikä ei Autovolan kohdalla toteudu. Autovolan voitaisiin katsoa sijoittuneen tasojen yksi ja kaksi väliin, sillä muiden organisaatioiden henkilöstö voi kuitenkin käyttää Autovolaa JYVSECTECin tarjoamien palveluiden, kuten kyberharjoitusten kautta.

## 11.4 Tuotteen jatkokehitys

Autovolaan voidaan tulevaisuudessa lisätä vielä uusia ominaisuuksia. Jotkut ominaisuuksista vaativat paljon aikaa ja työtä, eivätkä sen johdosta mahtuneet tähän Autovolan versioon. Toimeksiantajalla on kuitenkin aikomus jatkokehittää tuotetta tulevaisuudessa, ja mietimmekin yhdessä erilaisia toiminnallisuuksia, mitä Autovolaan voisi vielä lisätä.

Tällä hetkellä järjestelmässä ei ole toiminnallisuutta, joka mahdollistaisi käyttäjän identiteetin todentamisen. Sen johdosta Autovolaan ei sisällytetty kaikkia Volatility 3:n lisäosia ja käyttäjät eivät pysty lataamaan muistivedoksia palvelusta. Kuten luvussa 9.2.3 todetaan, joidenkin Volatility 3:n lisäosien paljastama data sisältää käyttäjien turvallisuutta vaarantavia riskejä. Myös kaikki muisti-

vedosten sisältämä tieto voisi joutua vääriin käsiin, jos palveluun ladattuja muistivedoksia pystyttäisiin lataamaan palvelusta takaisin. Käyttäjän identiteetin todentaminen voitaisiinkin toteuttaa yksilöllisillä käyttäjätunnuksilla. Käyttäjätunnuksille voisi jakaa käyttöoikeuksia muistivedoksiin, jolloin kaikilta käyttäjätunnuksilta ei pystyttäisi analysoimaan kaikkia muistivedoksia. Muistivedoksen palveluun lataama käyttäjätunnus voisi määritellä muut käyttäjätunnukset, joilla olisi oikeus analysoida muistivedosta. Käyttöoikeuksia voitaisiin myös luokitella niin, että tiettyjen oikeuksien haltijat voisivat myös ladata muistivedoksen palvelusta, eivätkä vain analysoida sitä.

Tällä hetkellä Autovolalla kyetään analysoimaan Windows- ja Linux-muistivedoksia. Volatility 3:lla voidaan myös analysoida macOS:tä otettuja muistivedoksia (Volatility 3 Documentation 2021, 433). Autovolaan voitaisiinkin lisätä tuki macOS-muistivedoksille. Autovolalla ei myöskään pystytä kommentoimaan lisäosalaatikoiden sisältämien taulukoiden (ks. kuvio 29) tuloksia. Lisäosan tuloksista tehtyjen havaintojen kommentointi esimerkiksi suoraan kyseiseen lisäosaan voisi helpottaa havaintojen muistamista ja edesauttaa käyttäjien välistä yhteistyötä, kun he pystyisivät lukemaan toistensa kommentteja.

Eräs käytännöllinen ominaisuus olisi käyttäjän mahdollisuus ajaa suoraan Volatility 3:n komentorivikomentoja verkkosovelluksen käyttöliittymästä. Tällöin käyttäjän ei välttämättä edes tarvitsisi käyttää Volatility 3:n komentoriviohjelmaa omassa järjestelmässään, koska se löytyisi Autovolasta. Luvussa 10.2 PrintKey-lisäosaa jouduttiin suorittamaan Volatility 3:n komentoriviohjelmasta, koska Autovolalla ei pystytä hakemaan tietoa tietyistä rekisteriavaimista. Jos Volatilityn komentorivikomentoja olisi kuitenkin voitu suorittaa käyttöliittymästä, rekisteriavaimen tiedot olisi voitu hakea sen kautta.

Autovolan käyttöliittymän graafisesta näkymästä tullaan vielä muokkaamaan toimeksiantajan näkemyksen mukainen. Tämä kuuluu Autovolan jatkokehitykseen, eikä sitä vaadittu valmiiseen toteutukseen. Jos kaikki tässä luvussa mainitut toiminnallisuudet saataisiin toteutettua, Autovola voisi olla varsinkin tiimityöskentelyä ajatellen hyvä pohja muistiforensiikan harjoittamiseen ja järjestelmä voisi kyetä syrjäyttämään Volatility 3:n komentoriviohjelman useissa eri tilanteissa.

# Lähteet

About Environment Variables. 2020. Artikkeli Windows-käyttöjärjestelmän ympäristömuuttujista Microsoft-verkkosivustolla. Viitattu 2.11.2020. https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables?view=powershell-7.

Apache Core Features. N.d. Dokumentaatio Apache-verkkosivustolla. Viitattu 26.3.2021. https://httpd.apache.org/docs/2.4/mod/core.html.

Berry, A., Homan, J. & Eitzman, R. 2017. WannaCry Malware Profile. FireEye-verkkosivuston Threat Research -blogi. Julkaistu 23.3.2017. Viitattu 18.4.2021. https://www.fireeye.com/blog/threat-research/2017/05/wannacry-malware-profile.html.

Blum, R. & Bresnahan, C. 2015. Linux Command Line and Shell Scripting Bible. Kolmas painos. Indianapolis: John Wiley & Sons.

Brezinski, D. & Killalea, T. 2002. Guidelines for Evidence Collection and Archiving. Dokumentti IETF-verkkosivustolla. Viitattu 9.10.2020. https://tools.ietf.org/html/rfc3227.

Carvey, H. 2016. Windows Registry Forensics. Toinen painos. Cambridge: Elsevier.

Command reference. 2020. Volatilityn wikisivu Github-verkkosivustolla. Viitattu 28.11.2020. https://github.com/volatilityfoundation/volatility/wiki/Command-Reference.

Compose file version 3 reference. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 25.3.2021. https://docs.docker.com/compose/compose-file/compose-file-v3/.

Configuration File Options. N.d. Dokumentaatio MongoDB-verkkosivustolla. Viitattu 24.3.2021. https://docs.mongodb.com/manual/reference/configuration-options/.

Crawford, T. & Hussain, T. N.d. A Comparison of Server Side Scripting Technologies. Viitattu 30.4.2021. https://csce.ucmss.com/cr/books/2017/LFS/CSREA2017/SER3291.pdf.

Das, A., Shen, M., Shashanka, M. & Wang, J. 2017. Detection of Exfiltration and Tunneling over DNS. Viitattu 15.3.2021. https://janet.finna.fi, IEEE Xplore Digital Library.

Declare default environment variables in file. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 24.3.2021. https://docs.docker.com/compose/env-file/.

Digital evidence. N.d. Yhdysvaltojen kansallisen standardi- ja teknologiainstituutin (NIST) verkkosivut. Viitattu 7.10.2020. https://www.nist.gov/topics/digital-evidence.

Docker load. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 11.4.2021. https://docs.docker.com/engine/reference/commandline/load/.

Docker logs. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 11.4.2021. https://docs.docker.com/engine/reference/commandline/logs/.

Docker network create. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 25.3.2021. https://docs.docker.com/engine/reference/commandline/network_create/.

Docker overview. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 16.3.2021. https://docs.docker.com/get-started/overview/.

Docker tag. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 11.4.2021. https://docs.docker.com/engine/reference/commandline/tag/.

Dockerfile reference. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 25.3.2021. https://docs.docker.com/engine/reference/builder/.

Docker-compose up. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 11.4.2021. https://docs.docker.com/compose/reference/up/.

Docker-compose logs. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 11.4.2021. https://docs.docker.com/compose/reference/logs/.

Environment variables in Compose. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 24.3.2021. https://docs.docker.com/compose/environment-variables/.

Fitzgerald, M. 2012. Introducing Regular Expressions. Sebastopol: O'Reilly Media.

Garg, R. & Verma, G. 2017. Operating Systems: A Modern Approach. Dulles: Mercury Learning And Information. Viitattu 27.11.2020. https://janet.finna.fi, Ebook Central (ProQuest).

General Python FAQ. 2021. Dokumentaatio Python-verkkosivustolla. Viitattu 19.3.2021. https://docs.python.org/3/faq/general.html.

Ghimire, D. 2020. Comparative study on Python web frameworks: Flask and Django. Opinnäytetyö, AMK. Metropolia ammattikorkeakoulu, tieto- ja viestintätekniikka. Viitattu 30.4.2021.

https://www.theseus.fi/bitstream/handle/10024/339796/Ghimire_Devndra.pdf?sequence=2&isAllowed=y.

Gupta, A.K., Arora, S.K. & Westcott, J.R. 2017. Industrial Automation and Robotics. Dulles: Mercury Learning And Information. Viitattu 25.11.2020. https://janet.finna.fi, Knovel.

Hosseini, A. 2017. Ten process injection techniques: A technical survey of common and trending process injection techniques. Elastic-verkkosivuston Engineering-blogi. Julkaistu 18.7.2017. Viitattu 4.11.2020. https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process.

How NTFS reserves space for its Master File Table (MFT). 2020. Artikkeli NTFS-tiedostojärjestelmän MFT-osiosta Microsoft-verkkosivustolla. Viitattu 10.11.2020. https://docs.microsoft.com/en-us/troubleshoot/windows-server/backup-and-storage/ntfs-reserves-space-for-mft.

How to disable and re-enable hibernation on a computer that is running Windows. 2020. Artikkeli Windows-käyttöjärjestelmän horrostiedostoista Microsoft-verkkosivustolla. Viitattu 12.10.2020. https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/disable-and-re-enable-hibernation.

Hyvärinen, N. 2017. Threat Hunting For Fileless Malware. F-secure-verkkosivuston Threats & Research -blogi. Julkaistu 11.4.2017. Viitattu 19.10.2020. https://blog.f-secure.com/threat-hunting-for-fileless-malware/.

Initializing a fresh instance. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 24.3.2021. https://hub.docker.com/_/mongo.

Johansen, A.G. N.d. What is fileless malware and how does it work? Artikkeli tiedostottomista haittaohjelmista Norton-verkkosivustolla. Viitattu 19.10.2020. https://us.norton.com/internetsecurity-malware-what-is-fileless-malware.html.

Johansen, G. 2017. Digital Forensics and Incident Response. Birmingham: Packt Publishing.

Jung, M., Youn, S., Bae, J. & Choi, Y. 2015. A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment. Viitattu 24.4.2021. https://janet.finna.fi, IEEE Xplore Digital Library.

Kao, D. & Hsiao, S. 2018. The Dynamic Analysis of WannaCry Ransomware. Viitattu 18.4.2021. https://janet.finna.fi, IEEE Xplore Digital Library.

Koskela, M. 2020. Metadata ja digitaalinen forensiikka. Tietotekniikan kandidaatintutkielma. Jyväskylän yliopisto. Viitattu 8.10.2020. https://jyx.jyu.fi/bitstream/handle/123456789/68932/1/URN%3ANBN%3Afi%3Ajyu-202005123136.pdf.

Kovács, Á. 2017. Comparison of Different Linux Containers. Viitattu 1.5.2021. https://janet.finna.fi, IEEE Xplore Digital Library.

Kumar, M.S., Ben-Othman, J. & Srinivasagan, K.G. 2018. An Investigation on Wannacry Ransomware and its Detection. Viitattu 18.4.2021. https://janet.finna.fi, IEEE Xplore Digital Library.

Kääntä, T. 2013. HTTP-palvelinohjelmien vertailu Raspberry Pi:ssä. Opinnäytetyö, AMK. Oulun ammattikorkeakoulu, tietojenkäsittely. Viitattu 24.4.2021. https://www.theseus.fi/bitstream/handle/10024/69356/Kaanta_Tero.pdf;jsessionid=0D2DE1D4254E085EB44C323D765ABEB6?sequence=1.

Levlin, M. 2020. DOM benchmark comparison of the front-end JavaScript frameworks React, Anglar, Vue, and Svelte. Tietojenkäsittelytieteiden pro gradu -tutkielma. Åbo Akademi. Viitattu 25.4.2021. https://www.doria.fi/bitstream/handle/10024/177433/levlin_mattias.pdf?sequence=2&isAllowed=y.

Ligh, M.H., Case, A., Levy, J. & Walters, A. 2014. The Art of Memory Forensics. Indianapolis: John Wiley & Sons.

Lukka, K. 2001. Konstruktiinen tutkimusote. Artikkeli konstruktiivisesta tutkimuksesta Metodix-verkkosivustolla. Viitattu 11.4.2021. https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Malware Trends. 2016. Dokumentaatio Yhdysvaltojen tietoturvaviraston verkkosivustolla. Viitattu 24.4.2021. https://us-cert.cisa.gov/sites/default/files/documents/NCCIC_ICS-CERT_AAL_Malware_Trends_Paper_S508C.pdf.

Marcho, C. 2019. Understanding Crash Dump Files. Artikkeli kaatumisvedostiedostoista Microsoft-verkkosivustolla. Viitattu 12.10.2020. https://techcommunity.microsoft.com/t5/ask-the-performance-team/understanding-crash-dump-files/ba-p/372633.

Matrosov, A., Rodionov, E. & Bratus, S. 2019. Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats. San Francisco: No Starch Press.

Messier, R. & Mackay, K. 2016. Operating System Forensics. Waltham: Elsevier. Viitattu 10.11.2020. https://janet.finna.fi, Skillsoft Books ITPro.

MongoDB Architecture Guide. 2015. Dokumentaatio MongoDB-verkkosivustolla. Viitattu 17.3.2021. https://jira.mongodb.org/secure/attachment/112939/MongoDB_Architecture_Guide.pdf.

MongoDB Limits and Thresholds. N.d. Dokumentaatio MongoDB-verkkosivustolla. Viitattu 18.4.2021. https://docs.mongodb.com/manual/reference/limits/.

Monnappa, K.A. 2018. Learning Malware Analysis. Birmingham: Packt Publishing.

Nerenberg, D.D. 2007. A Study of Rootkit Stealth Techniques and Associated Detection Methods. Tietotekniikan pro gradu -tutkielma. Yhdysvaltain ilmavoimien yliopisto. Viitattu 24.4.2021. https://apps.dtic.mil/dtic/tr/fulltext/u2/a519999.pdf.

Null, L. & Lobur, J. 2015. The Essentials of Computer Organization and Architecture. Neljäs painos. Burlington: Jones and Bartlett Learning. https://janet.finna.fi, Skillsoft Books ITPro.

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2015. Kehittämistyön menetelmät. 3. – 4. p. Helsinki: Sanoma Pro Oy.

Overview of Docker Compose. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 17.3.2021. https://docs.docker.com/compose/.

Perla, E. & Oldani, M. 2011. A Guide to Kernel Exploitation: Attacking the Core. Burlington: Elsevier. Viitattu 15.11.2020. https://janet.finna.fi, Skillsoft Books ITPro.

Purer, K. 2009. PHP vs. Python vs. Ruby – The web scripting language shootout. Viitattu 2.5.2021. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.457.4634&rep=rep1&type=pdf.

Quick Configuration Guide. N.d. Dokumentaatio readthedocs-verkkosivustolla. Viitattu 26.3.2021. https://modwsgi.readthedocs.io/en/master/user-guides/quick-configuration-guide.html.

React. 2021. ReactJS-verkkosivusto. Viitattu 17.3.2021. https://reactjs.org/.

Reith, M., Carr, C. & Gunsch, G. 2002. An Examination of Digital Forensic Models. Viitattu 10.10.2020. https://www.utica.edu/academic/institutes/ecii/publications/articles/A04A40DC-A6F6-F2C1-98F94F16AF57232D.pdf.

Reynolds, M. & Horvath, C. 2017. THREAT HUNTING: A PROACTIVE TECHNIQUE FOR FINDING SOPHISTICATED CYBER THREATS. Yhdysvaltojen eheyden ja tehokkuuden tarkastajien neuvoston verkkosivut. Viitattu 12.10.2020. https://www.ignet.gov/sites/default/files/files/9_26%20Reynolds%20Horvath.pdf.

Russinovich, M., Solomon, D.A. & Ionescu, A. 2012. Windows Internals Part 1. Kuudes painos. Washington: Microsoft Press.

ServiceType Enum. N.d. Dokumentaatio Microsoft-verkkosivustolla. Viitattu 28.11.2020. https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicetype?view=dotnet-plat-ext-3.1.

Sikorski, M. & Honig, A. 2012. Practical Malware Analysis. San Francisco: No Starch Press.

Souppaya, M. & Scarfone, K. 2013. Guide to Malware Incident Prevention and Handling for Desktops and Laptops. Yhdysvaltojen kansallisen standardi- ja teknologiainstituutin (NIST) verkkosivut. Viitattu 16.10.2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-83r1.pdf.

Specializing in cyber security expertise. N.d. JYVSECTECin verkkosivusto. Viitattu 14.11.2020. https://jyvsectec.fi/about/.

Tahiri, S. 2016. Digital Forensics Models. Artikkeli erilaisista digitaalisen forensiikan malleista Info-Sec Institute -verkkosivustolla. Julkaistu 25.1.2016. Viitattu 8.10.2020. https://resources.infosecinstitute.com/digital-forensics-models/.

Tunggal, A.T. 2020. What is Digital Forensics? UpGuard-verkkosivuston Cybersecurity-blogi. Julkaistu 5.8.2020. Viitattu 5.11.2020. https://www.upguard.com/blog/digital-forensics.

Use volumes. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 22.3.2021. https://docs.docker.com/storage/volumes/.

Vacca, J. R. 2013. Computer and Information Security Handbook. Toinen painos. Waltham: Elsevier. Viitattu 7.10.2020. https://janet.finna.fi, Skillsoft Books ITPro.

Virus. N.d. Yleiskuvaus viruksista F-secure-verkkosivustolla. Viitattu 20.10.2020. https://www.f-secure.com/v-descs/virus.shtml.

Volatility 3 Documentation. 2021. Volatility 3:n 1.0.1-version dokumentaatio readthedocs-verkkosivustolla. Viitattu 23.2.2021. https://volatility3.readthedocs.io/_/downloads/en/v1.0.1/pdf/.

What is the Apache HTTP Server Project. N.d. Dokumentaatio Apache-verkkosivustolla. Viitattu 24.3.2021. https://httpd.apache.org/ABOUT_APACHE.html.

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L. & Zhou, W. 2018. A Comparative Study of Containers and Virtual Machines in Big Data Environment. Viitattu 1.5.2021. http://arxiv-export-lb.library.cornell.edu/pdf/1807.01842.

Zhang, J. & Che, S. 2018. The Research on Linux Memory Forensics. Artikkeli iop-verkkosivustolla. Viitattu 10.10.2020. https://iopscience.iop.org/article/10.1088/1757-899X/322/5/052021/pdf.

Årnes, A. 2018. Digital Forensics. John Wiley & Sons. Viitattu 7.10.2020. https://janet.finna.fi, Skillsoft Books ITPro.

# Liitteet

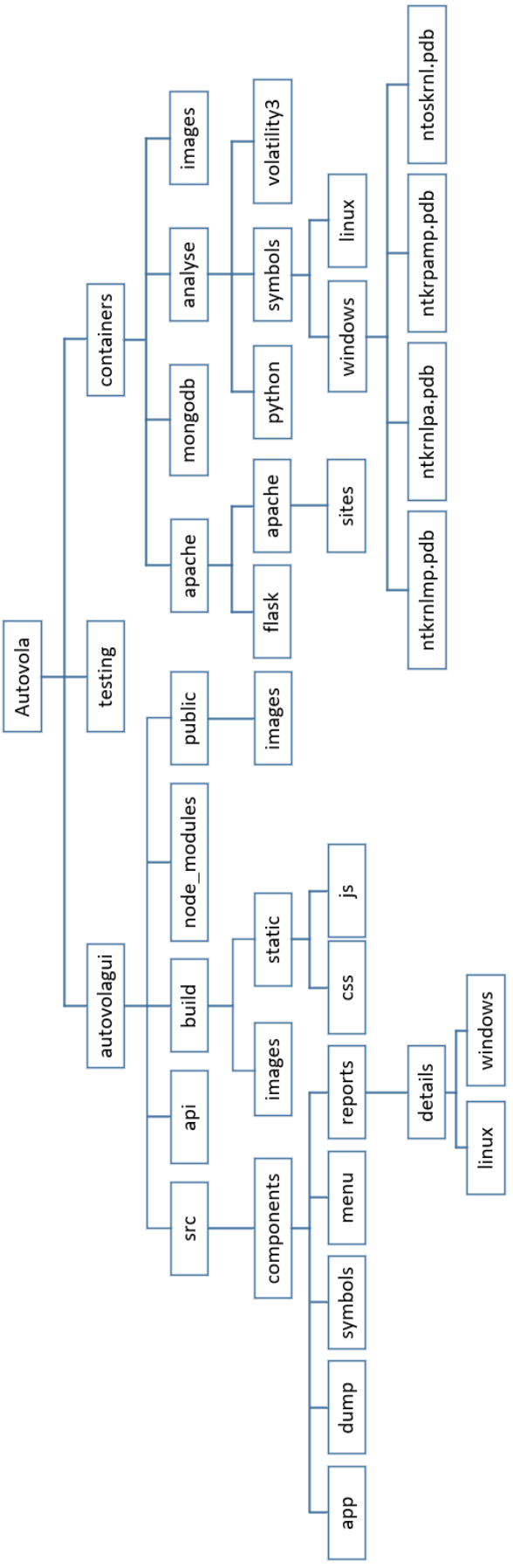
## Liite 1. Volatilityn mftparser-lisäosan tulostetta

```
tuomo@Kala:~/volatility$ sudo python vol.py -f ../vedos.raw --profile=Win10x64_18362 mftparser
Volatility Foundation Volatility Framework 2.6.1
Scanning for MFT entries and building directory, this can take a while
**************************************************************************
MFT entry found at offset 0xa400
Attribute: In Use & File
Record Number: 0
Link count: 1


$STANDARD_INFORMATION
Creation                       Modified                       MFT Altered                    Access Date                    Type
------------------------------ ------------------------------ ------------------------------ ------------------------------ ----
2019-09-18 07:14:26 UTC+0000 2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   Hidden & System

$FILE_NAME
Creation                       Modified                       MFT Altered                    Access Date                    Name/Path
------------------------------ ------------------------------ ------------------------------ ------------------------------ ---------
2019-09-18 07:14:26 UTC+0000 2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   $MFT

$DATA


$OBJECT_ID
Object ID: 40000000-0000-0000-0000-040000000000
Birth Volume ID: 00000400-0000-0000-0000-040000000000
Birth Object ID: 314000c1-0000-0000-b000-000048000000
Birth Domain ID: 01004000-0000-0500-0000-000000000000

**************************************************************************
**************************************************************************
MFT entry found at offset 0xa808
Attribute: In Use & Directory
Record Number: 5
Link count: 1


$STANDARD_INFORMATION
Creation                       Modified                       MFT Altered                    Access Date                    Type
------------------------------ ------------------------------ ------------------------------ ------------------------------ ----
2019-09-18 07:14:26 UTC+0000 2020-02-22 12:18:40 UTC+0000   2020-02-22 12:18:40 UTC+0000   2020-03-10 15:37:34 UTC+0000   Hidden & System

$FILE_NAME
Creation                       Modified                       MFT Altered                    Access Date                    Name/Path
------------------------------ ------------------------------ ------------------------------ ------------------------------ ---------
2019-09-18 07:14:26 UTC+0000 2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   2019-09-18 07:14:26 UTC+0000   .

**************************************************************************
```

Liite 2. Datavirran lisääminen tiedostoon

```
C:\ADS>dir /r
 Volume in drive C has no label.
 Volume Serial Number is B6B6-7EB6

 Directory of C:\ADS

11/11/2020  13.15    <DIR>          .
11/11/2020  13.15    <DIR>          ..
18/09/2019  19.48           131 795 skripti.ps1
               1 File(s)        131 795 bytes
               2 Dir(s)  49 318 166 528 bytes free

C:\ADS>echo ADS > skripti.ps1:datavirta

C:\ADS>dir
 Volume in drive C has no label.
 Volume Serial Number is B6B6-7EB6

 Directory of C:\ADS

11/11/2020  13.15    <DIR>          .
11/11/2020  13.15    <DIR>          ..
11/11/2020  13.16           131 795 skripti.ps1
               1 File(s)        131 795 bytes
               2 Dir(s)  49 318 100 992 bytes free

C:\ADS>dir /r
 Volume in drive C has no label.
 Volume Serial Number is B6B6-7EB6

 Directory of C:\ADS

11/11/2020  13.15    <DIR>          .
11/11/2020  13.15    <DIR>          ..
11/11/2020  13.16           131 795 skripti.ps1
                                  6 skripti.ps1:datavirta:$DATA
               1 File(s)        131 795 bytes
               2 Dir(s)  49 318 100 992 bytes free

C:\ADS>
```

Liite 3. Autovola-projektin hakemistorakenne



Liite 4. .env-tiedoston sisältö

```
# NETWORK
SUBNET=172.20.20.0/24
GATEWAY=172.20.20.1

# APACHE
APACHE_HOSTNAME=apache.autovola.com
APACHE_IP=172.20.20.200

# APACHE FLASK API
FLASK_APP=autovolagui/api/flask-api.py
FLASK_ENV=production
FLASK_SECRET_KEY=admin_secret_key

# ANALYSE
ANALYSE_MACHINE_AMOUNT=2

# MONGODB
MONGO_HOSTNAME=mongo.autovola.com
MONGO_IP=172.20.20.100
MONGO_USER=root
MONGO_PASSWORD=rootpassword1

# AUTOVOLA BASE & DUMPS & SYMBOLS DIRECTORIES
AUTOVOLA_DIRECTORY=/etc/autovola
DUMPS_DIRECTORY=/etc/autovola/dumps
SYMBOLS_DIRECTORY=/etc/autovola/symbols
```

Liite 5. Mongod.conf-tiedoston sisältö

```
# mongod.conf

# Where and how to store data.
storage:
  dbPath: /data/db
  journal:
    enabled: true

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

security:
  authorization: enabled
```

Liite 6. Apache-palvelun konfiguraatio docker-compose.yml-tiedostossa

```
services:
  apache:
    image: autovola/apache:latest
    restart: always
    container_name: apache
    cap_add:
      - NET_ADMIN
    privileged: true
    environment:
      - AUTOVOLA_DIRECTORY=${AUTOVOLA_DIRECTORY}
      - DUMPS_DIRECTORY=${DUMPS_DIRECTORY}
      - SYMBOLS_DIRECTORY=${SYMBOLS_DIRECTORY}
      - FLASK_SECRET_KEY=${FLASK_SECRET_KEY}
      - MONGO_USER=${MONGO_USER}
      - MONGO_PASSWORD=${MONGO_PASSWORD}
      - MONGO_HOSTNAME=${MONGO_HOSTNAME}
      - ANALYSE_MACHINE_AMOUNT=${ANALYSE_MACHINE_AMOUNT}
    networks:
      private:
        ipv4_address: ${APACHE_IP}
    ports:
      - 80:80
      - 8080:8080
    extra_hosts:
      - "${APACHE_HOSTNAME}:${APACHE_IP}"
      - "${MONGO_HOSTNAME}:${MONGO_IP}"
    volumes:
      - dumps:${DUMPS_DIRECTORY}
      - linux_symbols:${SYMBOLS_DIRECTORY}/linux/
      - windows_symbols:${SYMBOLS_DIRECTORY}/windows/
    build:
      dockerfile: ./containers/apache/dockerfile.apache
      context: .
      args:
        - AUTOVOLA_DIRECTORY=${AUTOVOLA_DIRECTORY}
        - DUMPS_DIRECTORY=${DUMPS_DIRECTORY}
        - SYMBOLS_DIRECTORY=${SYMBOLS_DIRECTORY}
        - APACHE_HOSTNAME=${APACHE_HOSTNAME}
```

Liite 7. Apache-palvelun Dockerfile-tiedoston sisältö

```
FROM autovola/apache:latest
# Commented commands below have been run in image
#RUN apt-get update
#RUN apt-get install vim apache2 python3-pip lsof libapache2-mod-wsgi-
py3 python-dev net-tools -y
#RUN pip3 install bson flask Unidecode pyftpdlib python-
dotenv requests pymongo flask-cors waitress threaded

# ARGUMENTS
ARG AUTOVOLA_DIRECTORY
ARG DUMPS_DIRECTORY
ARG SYMBOLS_DIRECTORY
ARG APACHE_HOSTNAME

# Tell Apache its hostname
RUN echo "export APACHE_HOSTNAME=${APACHE_HOSTNAME}" >> /etc/environment

### PYTHON API ###
RUN mkdir ${AUTOVOLA_DIRECTORY}
RUN mkdir -p ${DUMPS_DIRECTORY}
RUN mkdir -p ${SYMBOLS_DIRECTORY}/windows
RUN mkdir -p ${SYMBOLS_DIRECTORY}/linux
RUN chown www-data:www-data ${AUTOVOLA_DIRECTORY}
RUN chown www-data:www-data ${DUMPS_DIRECTORY}
RUN chown -R www-data:www-data ${SYMBOLS_DIRECTORY}
RUN mkdir -p /var/www/api/static
RUN mkdir /var/log/autovola/
COPY autovolagui/api/flask-api.py /var/www/api/flaskapi.py
COPY autovolagui/api/mongoconn.py /var/www/api/mongoconn.py
COPY containers/apache/flask/flaskapi.wsgi /var/www/api/flaskapi.wsgi

### APACHE ###
RUN chmod -R 755 /var/www
COPY containers/apache/apache/sites/ /etc/apache2/sites-available/
COPY containers/apache/apache/apache2.conf /etc/apache2/apache2.conf
COPY containers/apache/apache/envvars /etc/apache2/envvars
COPY autovolagui/build/ /var/www/html/
RUN a2ensite api
RUN a2enmod rewrite
RUN a2enmod headers
RUN update-rc.d apache2 enable
RUN apache2ctl -k graceful
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

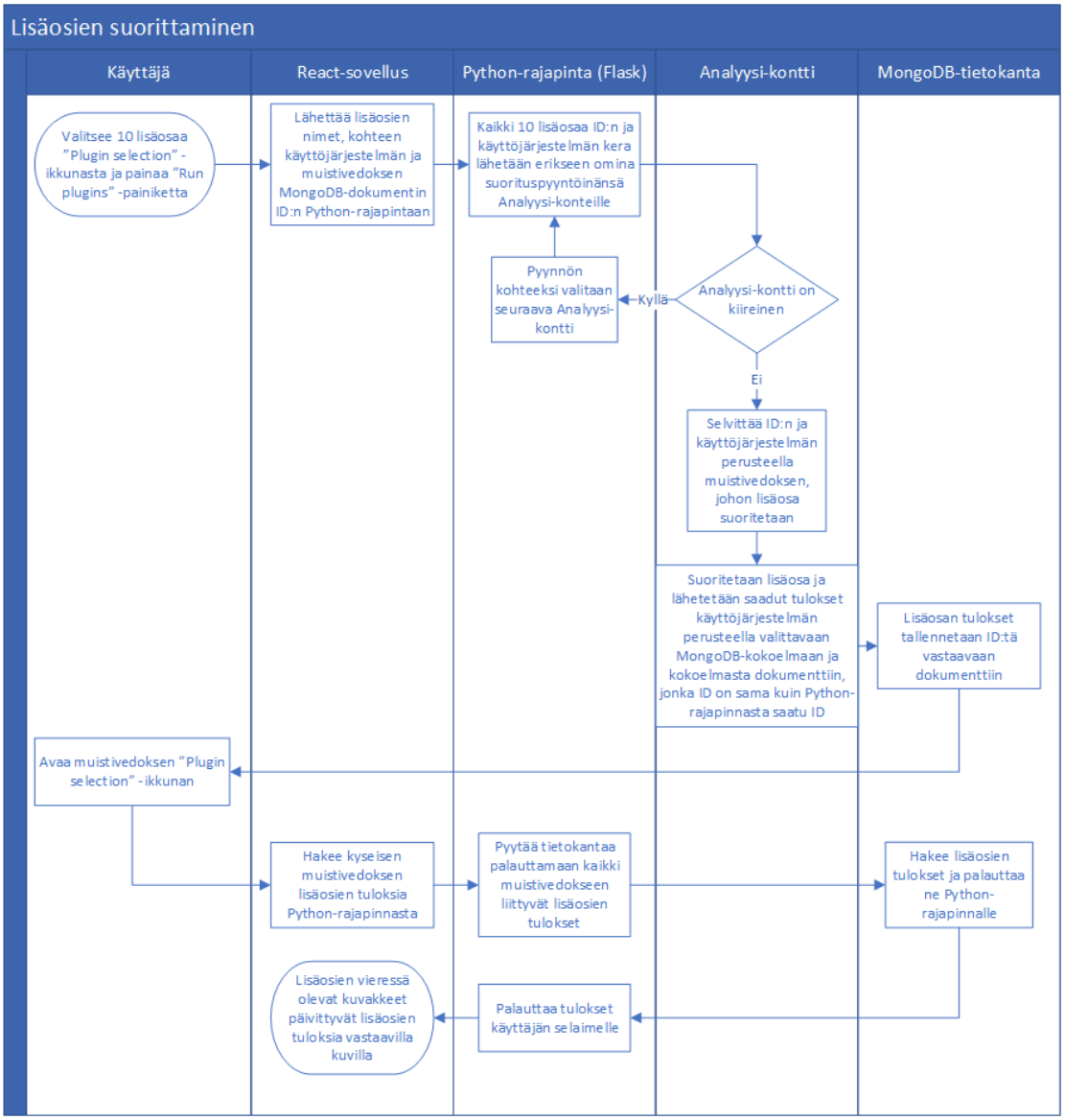
Liite 8. Analyysi-palvelun konfiguraatio docker-compose.yml-tiedostossa

```
analyse:
  image: autovola/analyse:latest
  restart: always
  networks:
    private:
  expose:
    - "80"
  privileged: true
  environment:
    - AUTOVOLA_DIRECTORY=${AUTOVOLA_DIRECTORY}
    - DUMPS_DIRECTORY=${DUMPS_DIRECTORY}
    - SYMBOLS_DIRECTORY=${SYMBOLS_DIRECTORY}
    - MONGO_PASSWORD=${MONGO_PASSWORD}
    - MONGO_USER=${MONGO_USER}
    - MONGO_HOSTNAME=${MONGO_HOSTNAME}
  extra_hosts:
    - "${APACHE_HOSTNAME}:${APACHE_IP}"
    - "${MONGO_HOSTNAME}:${MONGO_IP}"
  volumes:
    - dumps:${DUMPS_DIRECTORY}
    - linux_symbols:${SYMBOLS_DIRECTORY}/linux/
    - windows_symbols:${SYMBOLS_DIRECTORY}/windows/
  build:
    dockerfile: ./containers/analyse/dockerfile.analyse
    context: .
    args:
      - AUTOVOLA_DIRECTORY=${AUTOVOLA_DIRECTORY}
      - DUMPS_DIRECTORY=${DUMPS_DIRECTORY}
      - SYMBOLS_DIRECTORY=${SYMBOLS_DIRECTORY}
  deploy:
    mode: replicated
    replicas: ${ANALYSE_MACHINE_AMOUNT}
    resources:
      limits:
        memory: 4096M
      reservations:
        memory: 256M
```

Liite 9. Analyysi-palvelun Dockerfile-tiedoston sisältö

```
FROM autovola/analyse:latest
# Commented commands below have been run on this image
# RUN apt-get update
# RUN apt-get install net-tools vim python3 python3-pip -y
# RUN pip3 install bson configparser python-
dotenv pymongo flask waitress threaded pycryptodome distorm3 yara-
python urllib3 capstone

# ARGUMENTS
ARG AUTOVOLA_DIRECTORY
ARG DUMPS_DIRECTORY
ARG SYMBOLS_DIRECTORY

# Create Autovola directories and log file
RUN mkdir -p ${AUTOVOLA_DIRECTORY}
RUN mkdir -p ${DUMPS_DIRECTORY}
RUN mkdir -p ${SYMBOLS_DIRECTORY}/linux
RUN mkdir -p ${SYMBOLS_DIRECTORY}/windows
RUN mkdir ${AUTOVOLA_DIRECTORY}/volatility3
RUN touch /var/log/autovola.log

# Copy symbol files, Volatility3 and python code from host to container
COPY containers/analyse/volatility3/ ${AUTOVOLA_DIRECTORY}/volatility3/
COPY containers/analyse/python/ ${AUTOVOLA_DIRECTORY}
COPY containers/analyse/symbols/windows/ ${SYMBOLS_DIRECTORY}/windows
RUN chmod 444 ${AUTOVOLA_DIRECTORY}/plugins.cfg

# Install Volatility 3 python library
WORKDIR ${AUTOVOLA_DIRECTORY}/volatility3/
RUN cd ${AUTOVOLA_DIRECTORY}/volatility3/
RUN python3 setup.py install
WORKDIR ${AUTOVOLA_DIRECTORY}
CMD ["python3","-u","main.py"]
```

Liite 10. Prosessikaavio lisäosien suorittamisesta muistivedokseen Autovolalla



# Liite 11. Yleisnäkymä muistivedoksen analysointisivusta