Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Sumeiden tiivisteiden soveltaminen haittaohjelmien tunnistamisessa

Valtteri Varvikko

23.12.2022

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Batchelor's Programme in Computer Science

Tekijä — Författare — Author

Valtteri Varvikko

Työn nimi — Arbetets titel — Title

Sumeiden tiivisteiden soveltaminen haittaohjelmien tunnistamisessa

Ohjaajat — Handledare — Supervisors

prof. Nikolaj Tatti

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Bachelor's thesis	December 23, 2022	20 pages

Tiivistelmä — Referat — Abstract

Tiivistys on yleisesti käytetty menetelmä haittaohjelmien tunnistamiseen. Erityisesti moderneista haittaohjelmmista levitetään tyypillisesti muunneltuja versioita, joita on vaikeampi tunnistaa. Kryptografisilla tiivisteillä ei voida tunnistaa tällaisia samankaltaisia ohjelmia. Samankaltaisuuksinen tunnistamiseen on kehitetty erilaisia sumeita tiivisteitä, jotka kykenevät yhdistämään suurestikin poikkeavia syötteitä toisiinsa. Sumeita tiivistetiä on paljon, ja paikoitellen ne eroavat toisistaan merkittävästi. Tässä tutkielmassa käsitellään sumeiden tiivisteiden soveltamista haittaohjelmien tunnistamiseen niiden sisältämien samankaltaisuuksien perusteella. Erityisesti analysoidaan eri ohjelmistojen soveltuvuutta ja puutteita sekä sumean tiivistyksen haavoittuvuuksia.

ACM Computing Classification System (CCS)
Security and privacy → Cryptography → Symmetric cryptography and hash functions → Hash functions and message authentication codes
Security and privacy → Intrusion/anomaly detection and malware mitigation → Malware and its mitigation

Avainsanat — Nyckelord — Keywords

similarity hashing, fuzzy hashing, malware detection

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Sisällys

1 Johdanto

Tietojenkäsittelyssä käsitellään valtavia määriä dataa. Laskentakyvyn kasvessa myös tiedonsiirto- ja tallenuskapasiteetti kehittyvät, ja tietojenkäsittelyn kehitys luo myös rikollisia intressejä edistäviä mahdollisuuksia. Sekä tekninen rikostutkinta että kyberrikollisuus ovat riippuvaisia tietotekniikan kehityksestä. Uudet innovaatiot synnyttyävät mahdollisuuksia kehittää rikollisia menetelmiä sekä toisaalta ratkaisuja haitallista toimintaa vastaaan.

Kasvava datavirta helpottaa haittaohjelmien huomaaamatonta leviämistä, ja haittaohjelma saattaa jäädä kokonaan huomaamatta. Haittaohjelmien torjuntaa voidaan tietoisesti pyrkiä johtamaan harhaan aiheuttamalla aiheettomia osumia. Haitallisen datan tunnistaminen kasvavasta tietoliikenteestä edellyttää kohdennettuja resursseja sekä suorituskykyisiä ratkaisuja. Tiedon esittämiselle kompaktimmassa muodossa onkin tarvetta sen tehokkaan tunnistamisen vuoksi.

Dataa voidaan tiivistää muun muassa pakkaus- ja tiivisteohjelmistoilla. Pakkauksesta poiketen tyypillisesti käytetyt kryptografiset tiivistefunktiot tiivistävät datan palauttamattomaan muotoon. Toisaalta tiiviste sisältää lähes uniikin tunnisteen, jolla alkuperäinen data on käytännössä tunnistettavissa. Tiivistystä voidaankin hyödyntää menetelmänä haittaohjelmien havaitsemisessa suuresta datamäärästä. Tyypillisesti tunnistetuista haittaohjelmista on luotu tiiviste, jolloin saman tiivisteeen tuottavaa ohjelmaa voidaan epäillä haittaohjelmaksi. Mikä tahansa muutos tuottaa kuitenkin tunnistamattomaan tiiviisteen perinteisesti tarkoitukseen käytetyillä kryptografisilla tiivistefunktioilla. Rikolliset tahot pyrkivät kiertämään haittaohjelmien torjuntamenetelmiä esimerkiksi tekemällä ohjelmistoon muutoksia. Muutettua haittaohjelmaa ei voida enää tunnistaa alkuperäiseksi tiivisteitä vertaamalla, vaikka toiminnallisuudeltaan ohjelma olisi identtinen.

Seuraavassa luvussa esitellään yleisesti sumean tiivisteen käsite sekä taustoitetaan eri tiivistetyyppien toimintaperiaate. Luvussa 3 käydään läpi kolme sumeiden tiivisteiden prosessointiin käytettyä ohjelmistoa: Ssdeep, Sdhash sekä Mvhash-b. Luvussa 4 käsitellään sumeiden tiivsteiden soveltamista haittaohjelma-analyysissa edellytyksineen. Lisäksi luvussa analysoidaan menetelmiä, joilla haittaohjelmien tunnistamista sumein tiivistein voidaan johtaa harhaan. Luvussa 5 käsitellään sumean tiivistämisen heikkouksia sekä tarkastellaan ohjelmistojen eroavaisuuksien merkitystä haittaohjelma-analyysissa.

2 Sumea tiiviste

Kryptografisia tiivisteitä käytetään laajasti datan tunnistamiseen (Kornblum, 2006). Ne tuottavat samalla syötteellä saman tiivisteen, ja data voidaan lähes varmuudella tunnistaa samaksi yhtäläisen tiivisteen perusteella. Pieninkin muutos syötteessä tuottaa kuitenkin aiemmasta tunnistamattoman tiivisteen. Vertailu kryptografisilla tiivisteillä ei ole soveltuva keino samankaltaisten syötteiden tunnistamiseen (Kornblum, 2006).

Samankaltaisuuden tunnistamista varten kehitetyillä sumeilla tiivisteillä voidaan vaivatta tunnistaa toisistaan eroavia samankaltaisia syötteitä (Kornblum, 2006). Sumeat tiivisteet mahdollistavat suurestikin muutettujen tai puutteellisten tiedostojen yhdistämisen alkuperäiseen versioon. Samanakaltaisuuksien tunnistamisella on merkitystä myös haittaohjelma-analyysissa. Haittaohjelmia kehittävillä tahoilla on taipumus toistaa kaavamaisia piirteitä, joten yhden tunnistetun haittaohjelman avulla voidaan mahdollisesti havaita muitakin haittaohjelmia (Naik, Jenkins ja Savage, 2019).

Tässä tutkielmassa sumeat tiivisteet jaetaan toimintaperiaatteen mukaan luokkiin perustuen luokitukseen, jonka Martín-Pérez, Rodríguez ja Breitinger (2021) esittelevät. Tiivisteet on jaettu ominaisuuksiensa perusteella kolmeen luokkaan. Seuraavassa aliluvussa käsitellään sumeiden tiivistefunktioiden toimintaa yleisellä tasolla. Viimeisissä aliluvuissa käsitellään kaksi sumeiden tiivisteiden luokkaa: tiivistetyt ominaisuusjonot sekä tavujonot.

2.1 Tiivistäminen ja vertailu

Samankaltaisuuksia tunnistavia algoritmeja laatiessa keskitytään koko syötteen tarkastelun sijaan pieniin osioihin. Olennaista ei ole tunnistaa täsmälleen tietynlaista syötettä, vaan löytää syötteestä vertailuun sopivia säännönmukaisuuksia. Näitä rakenteita kutstuaan usein ominaisuuksiksi (Martín-Pérez, Rodríguez ja Breitinger, 2021). Ominaisuuksien valintakriteerit ja vertauslogiikka riippuvat käytetystä algoritmista.

Monet algoritmit muodostavat sumean tiivisteen jonosta, joka sisältää syötteestä valitut ominaisuudet tiivistettynä (Martín-Pérez, Rodríguez ja Breitinger, 2021). Osa tiivisteistä tiivistää koko syötteen, toisaalta osa tiivistää ainoastaan osan. Huomionarvoista on, että ominaisuuksien tiivistämiseen käytetään sumeisssakin tiivisteissä kryptografisia tiivisteitä

(Kornblum, 2006). Datan samankaltaisuuden vertailu sumeilla tiivisteillä etenee vertaillen tiivistettyjä ominaisuuksia koko tiivisteen sijaan. Samankaltaisuuden mittarina käytettävä ominaisuuksien vertailufunktio riippuu käytetystä algoritmista.

2.2 Tiivistetyt ominaisuusjonot

Ominaisuusjonojen tiivistäminen on sumeiden tiivisteiden luokka, johon kuuluu useita laajalti käytettyjä tiivisteitä. Martín-Pérez, Rodríguez ja Breitinger (2021) esittävät tiivistetyt ominaisuusjonot yhtenä kolmesta tiivisteluokasta. Tässä lähestymistavassa tarkastellaan ominaisuuksista muodostuvia jonoja, eli syötteestä etsitään ominaisuuksia, jotka tiivistetään jonoksi. Piirteen määrittely sekä valinta on algoritmikohtaista.

Taulukko 2.1: Merkkijonolle luotu tiiviste, joka on määritelty paloittain.

h	e	i		m	a	a	i	l	m	a	!
7b			7f			e5			53		

Tiivistetty ominaisuusjono voidaan muodostaa määrittelemällä tiiviste paloittain (Martín-Pérez, Rodríguez ja Breitinger, 2021). Syöte jaetaan lohkoihin, jolloin suuresta syötteestä muodostuu jono vertailtavia ominaisuuksia. Yksinkertaisessa paloittaisessa määrittelyssä syöte jaetaan vakiokokoisiin lohkoihin, jotka kuvaavat syötteen ominaisuuksia (Kornblum, 2006). Paloittain määritelty tiiviste muodostuu siis jonosta tiivistettyjä lohkoja $H_i = R(a_j, a_{j-1}, a_{j-2}, \ldots, a_{j-n})$, missä n käytetty lohkokoko ja $j = in$. Menetelmää havainnollistetaan taulukossa 2.1, jossa syötteen lohkoille generoidaan tiivisteet CRC-8-tiivistefunktiolla lohkokoon ollessa $n = 3$.

Taulukko 2.2: Paloittain määritelty tiiviste merkkijonolle, jossa merkkejä on vaihdettu.

h	e	i		m	a	a	i	l	m	a	?
38			7f			e5			09		

Taulukossa 2.2 on kuvattu muutoksia, jotka aiheutuvat, kun syötteen alkioita vaihdetaan. Koska syötteen pituus säilyy muuttumattomana, muutokset heijastuvat ainoastaan lohkoihin, joissa muutoksia on tapahtunut; keskimmäisten lohkojen tiivisteet eivät ole muuttuneet lainkaan. Sen sijaan taulukossa 2.3 huomataan algoritmin toiminnan häiriintyvän pahasti, kun syötteen pituus muuttuu. Alkion lisäys tai poisto muuttaa kyseistä alkiota seuraavien

alkoiden indeksiä. Syötteen alkuun lisätty alkio johtaa pahimmillaan kaikkien seuraavien lohkojen tiivisteen muuttumisen tunnistamattomaksi.

Taulukko 2.3: Paloittain määritelty tiiviste johon on lisätty merkki.

H	e	i	,		m	a	a	i	l	m	a	!
7b			e9			54			03			e7

Vakiokokoisilla lohkoilla esiintyvä ongelma ratkeaa, kun lohko sisältää saman alkiojonon lisäyksistä riippumatta (Kornblum, 2006). Kontekstiriippuvaisessa paloittaisessa tiivistyksessä lohkot määritetään syötteen perusteella. Algoritmin toimintaperiaatteena on jakaa syöte lohkoiksi siten, että lohkot sisältävät algoritmille määritellyn kontekstin. Kontekstiriippuva paloittain määritelty tiiviste ei ole herkkä syötteen pituuden muuttumisel le. Lisäämistä ja poistamisesta vaikuttaa ihanteellisesti yhden, ja enimmilläänkin vain lähimpien naapurilohkojen tiivisteisiin eikä heijastu muihin lohkoihin.

Paloittaisen määritettelyn lisäksi sumeaa tiivistystä on lähestytty muilla tavoilla. lohkoihin jakamisen sijaan syötteeestä voidaan etsiä tilastollisesti poikkeavia piirteitä. Poikkeavia piirteitä havainnoiva algoritmi valitsee tietyn määrän poikkeavimpia piirteitä ja tiivistää ne (Roussev, 2010). Varsinainen sumea tiiviste muodostuu valittujen piirteiden tiivisteistä. Paloittain määriteltyjen tiivisteiden tavoin tiivistämiseen käytetään kryptografista tiivistefunktiota. Huomionarvoista on, että yleensä tiivistettäväksi valitut ominaisuudet kattavat syötteestä ainoastaan osan (Martín-Pérez, Rodríguez ja Breitinger, 2021). Syötteestä ei siis tiivistetä kuin osa, toisin kuin paloittain määritteltäessä.

Poikkeavuudet säilyttävän tiivisteen lähestymistavassa tulkitaan, että samalla tavalla merkittävästi poikkeavat syötteet liittyvät toisiinsa. Ei siis ole todennäköistä, että satunnaisissa syötteissä esiintyisi suuri määrä samoja, epätyypillisiä piirteitä. Kuvatussa tilanteessa on pääteltävissä syötteiden olevan mitä luultavimmin peräisin samasta alkuperästä, ja piirteet ovat muokkausten jälkeen jääneet yhdistäväksi tekijäksi. Tällöin voidaan tehdä johtopäätös ja olettaa syötteiden olevan tietyllä todennäköisyydellä kokonaisuutenakin samankaltaisia.

Poikkeavuuksiin perustuvuvan algoritmin on etsittävä syötteestä piirteet ja valittava niistä epätyypillisimmät. Keskeinen ongelma onkin piirteen määrittely. Onglemaan ei ole yksikäsitteistä ratkaisua, ja määritelmä on tapauskohtaista. Esimerkiksi seuraavassa luvussa käsiteltävässä Sdhash-ohjelmistossa piirre on määritelty 64 tavun merkkijonoksi (Roussev, 2010).

2.3 Tavujono

Tavujonosyötteet muodostavat tiivisteen jakamalla syötteen lohkoiksi, joita verrataan samankaltaisuuden perusteella yhtäsuuruuden sijaan (Martín-Pérez, Rodríguez ja Breitinger, 2021). Lohkoista rakennetaan syötteelle uusi esitystapa, joka sisältää yksinkertaistetun kuvauksen syötteestä. Muutokset alkuperäisessä syötteessä eivät vaikuta tietyissä rajoissa yksinkertaistettuun muotoon, jonka perusteella samankaltaisuudet voidaan edelleen määrittää. Tavujonotiivisteet eivät etsi syötteestä tietyntyyppisiä ominaisuuksia kuten ominaisuustiivisteet.

Tavujonotiivisteille on määriteltävä logiikka, jolla lohkonta suoritetaan. Lohkot voivat olla myös vakiokokoisia, jos niiden tiivistykseen ei käytetä kryptografista tiivistettä; esimerkiksi Mvhash-b-ohjelmistossa lohkot ovat vakiokokoisia. Lohkon arvoksi asetetaan joko 0 tai 255, riippuen alkioiden enemmistöstä. Enemmistöperiaatteen sekä arvojen pienen määrän vuoksi lohkosta toiseen siirtyvien alkioiden vaikutus häipyy.

3 Sumean tiivistämisen ohjelmistot

3.1 Ssdeep

Ssdeep on laajasti käytetty ohjelmisto kontekstiriippuvaisten paloittain määriteltyjen tiivisteiden prosessointiin (Martínez, Álvarez, Encinas ja Ávila, 2015). Syötteiden vertailu paloittain määritellyin tiivistein tapahtuu kahdessa vaiheessa. Tiivisteet on ensin generoitava, minkä jälkeen samankaltaisuuden vertaaminen tapahtuu tiivisteitä verraten. Ssdeepillä voidaan generoida kontekstitiippuvainen tiiviste tidostoille ja vertailla generoituja tiivisteitä keskenään.

Taulukko 3.1 havainnollistaa ssdeep-ohjelmiston käyttöä. Kahdelle JavaScript-lähdekooditiedostolle generoidaan tiivisteet. Tämän jälkeen saatuja tiivisteitä verrataan Ssdeepin vertailutilassa, joka ilmaisee, ovatko syötteet samankaltaisia. Ssdeep määrittää samankaltaisuuden suljetulla välillä $[0, 100]$, missä 0 tarkoittaa pienintä havaittavaa vastaavuutta, ja 100 täydellistä vastaavuutta (Kornblum, 2006). Mikäli Ssdeep ei kykene tulkitsemaan syötteitä samankaltaisiksi, komento ei tulosta mitään.

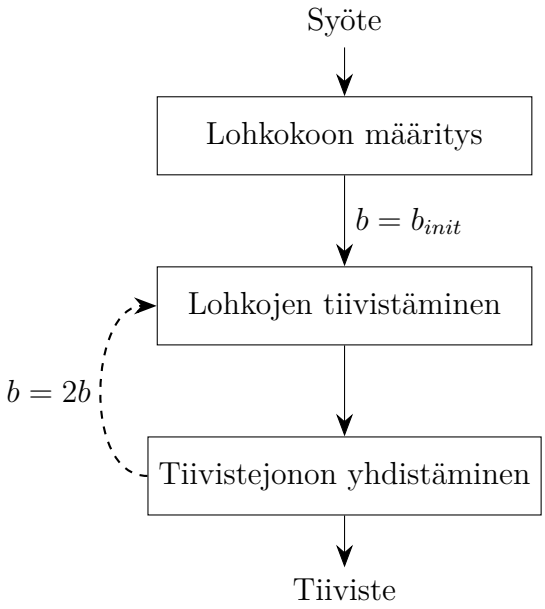
Taulukko 3.1: Alkuperäisen ja muokatun lähdekooditiedoston luonti ja vertailu ssdeep-ohjelmistolla.

```
 $ ssdeep -l functional.js functional-uncommented.js
ssdeep,1.1--blocksize:hash:hash,filename
192:eQiHcd9A+QKxYI1LIgbKvgiuqBOML2lJGnYDTo/Z:Acd9A+QKlLpbKJ01lJQYDGZ,
  "functional.js"
192:oQ+BA+QKNYI1LIPuiuqWOML2lJGnYDTo/Z:KA+QKJLp01lJQYDGZ,
  "functional-uncommented.js"

$ ssdeep -lrd functional.js functional-uncommented.js
functional-uncommented.js matches functional.js (75)
```

Ssdeepin generoima tiiviste muodostuu varsinaisen tiivisteen lisäksi lohkokoon osoittavasta kokonaisluvusta sekä syötetyn tiedoston polusta. Lohkokoko riippuu syötteenä annetun datan pituudesta (Kornblum, 2006). Itse tiiviste muodostuu kahdesta kaksoispisteellä ero-

tetusta tiivisteestä. Ssdeep iteroi syötteen uudelleen käyttäen laukaisimena kaksinkertaista lohkokokoa. Ssdeepin tiivisteiden vertailu edellyttää, että niitä luodessa käytetyt lohkokoot ovat yhtäsuuret (Kornblum, 2006).



Kuva 3.1: Kontekstiriippuvaisen paloittain määritellyn tiivisteen luonti Ssdeep-ohjelmistossa.

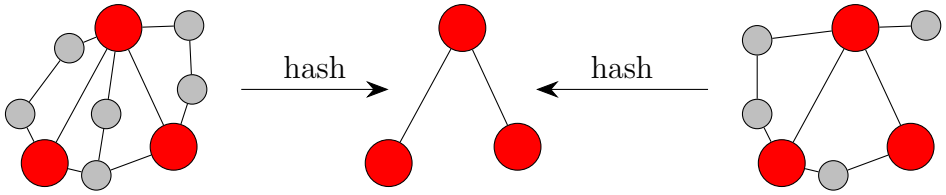
Ssdeep käyttää kontekstin tunnistamiseen monen kontekstiriippuvaisen tiivistealgoritmin tavoin rekursiivista tiivisitettä (Kornblum, 2006). Syötettä iteroidessasan Ssdeep käyttää rekursiivista tiivistettä pitääkseen yllä tilaa viimeisimmin käsiteltyjen alkioiden muodostamasta kontekstista. Rekursiivista tiivistettä käytettäessä on yleensä määritetly myös niin sanottu laukaisinarvo, johon rekursiivista tiivistettä verrataan. Laukaisinarvoa vastaava rekursiivisesti määritetlty tiiviste ilmaisee vallitsevan kontekstin vaihtumisen. Kontekstin ilmaiseva tiiviste h voidaan esittää tiivistefunktion R rekursiivisena kutsuna

$$h = R(a_i, R(a_{i-1}, R(a_{i-2}, \ldots, R(a_{i-n+1}) \ldots))),$$

missä i on haluttu syötteen indeksi ja n huomioitavien alkioiden määrä. Edellinen lauseke ei ole kuitenkaan tehokas tapa laskea tiivistettä. Käytännön sovelluksissa riittävää on tiivistää nykyinen alkio sekä edellisten alkioiden tiiviste, josta poistetaan varhaisimman alkion vaikutus algoritmista riippuen esimerkiksi bittioperaatiolla. Ssdeepissä tiivistykseen käytetään Adler32-tiivistefunktiota (Kornblum, 2006).

3.2 Sdhash

Sdhash on epätodennäköisiä ominaisuuksia tiivistävä ohjelmisto sumeiden tiivisteiden prosessointiin. Sdhash kuuluu tiivistettyihin ominaisuusjonoihin (Martín-Pérez, Rodríguez ja Breitinger, 2021). Naik, Jenkins ja Savage (2019) mainitsevat Sdhashin perustuvan väitteelle, jonka mukaan kaikilla syötteille on tieyttyjä tilastollisia ominaisuuksia. Osa syötteen ominaisuuksista on erityisen epätodennäköisesti ilmeneviä. Datan identiteetin säilymisen kannalta ominaisuuksista valitaan epätodennäköisimmin ilmenevät. Mitä useampi epätodennäköisesti ilmenevevä ominaisuus ilmenee toisessa syötteessä, sitä suuremmalla todennäköisyydeellä niiden alkuperä on sama.



Kuva 3.2: Kaksi samasta verkosta erisuuriksi muokattua verkkoa tiivistetään samaksi tiivisteeksi valitsemalla ainoastaam poikkeavat eli suuremmeat alkiot.

Sdhashin ominaisuudet ovat 64 tavun merkkijonoja (Roussev, 2010). Tiivisteen generoinnin aikana Sdhash käy läpi syötteen kaikki ominaisuudet valiten niistä epätodennäköisimmät. Jokaiselle syötteen ominaisuudelle määritetään entropia-arvo. Syötteen ominaisuuksille generoidaan presedenssiarvo perustuen entropia-arvoon. Populaariarvon laskemiminen vaatii syötteen läpikäyntiä käyttäen ikkunaa, joka rajaa sisälleen W syötteen ominaisuutta. Ikkunaa liu'utetaan ominaisuus kerrallaan eteenpäin, ja jokaisen ominaisuuden kohdalla valitaan pienimmästä indeksistä katsottuna ikkunan pienin presedenssiarvo. Valitun arvon kohdalla olevan ominaisuuden populaariarvoa kasvatetaan yhdellä.

Syötteen iteroimisen jälkeen ominaisuuksista valitaan ne, joiden populaariarvolle pätee $R_{pop} >= t$, missä t on algoritmille määritelty raja-arvo. Valituilla ominaisuuksilla katsotaan olevan tilastollisesti pieni todennäköisyys esiintyä sattumanvaraisessa syötteessä. Valitut ominaisuudet tiivistetään SHA-1-tiivistefunktiolla tallettavaksi generoitavaan tiiviisteeseen. Tiivistetyt ominaisuudet talletetaan Bloom-suodattimeen, joista muodostetaan tiiviste yhdistämällä suodattimet jonoksi. Bloom-suodattimella esitettävään joukkoon voidaan lisätä alkiota sekä tarkistaa, kuuluuko alkio joukkoon. Suodatin totetutetaan bittijonona, jonka permutaatiot kuvaavat joukkoon kuuluvia alkiota. Syötteille yhteisten ominaisuuksien määrä on selvitettävissä laskemalla suodatinten välinen Hamming-etäisyys.

Sdhashissa ilmenee Bloom-suodattimesta johtuvia aiheettomia osumia. Suorituskyyvn optimointi heikentää tarkkuutta, ja suodatin saattaa virheellisesti tulkita tuntemattoman alkion kuuluvan joukkoon. Jokainen bitti kuvaa useampaa kuin yhtä permutaatiota, joten useampi bittipermutaatio saattaa samanaikaisesti kuvata myös muita, joukkoon kuulumattomia alkioita. Aiheettomat osumat eivät käytännössä aiheuta tuloksiin merkityksellisiä muutoksia.

Breitinger, Baier ja Beckingham (2012) mainitsevat Sdhashin vahvuutena olevan katkelmien havaitseminen syötteestä tiedostojen vertaamisen sijaan.

3.3 Mvhash-b

Mvhash-b on Ssdeepistä ja Sdhashista poikkeava ohjelmisto, jonka toimintaperiaate perustuu tavujonojen esiintyvyyteen (Martín-Pérez, Rodríguez ja Breitinger, 2021). Mvhash-b ei tarkastele syötettä lohkoina.

Mvhash-b vertaa syötteen jokaisen indeksin k n-naapuruston tavujen 1-bittien lukumäärää $bitcount(N_{k,n})$ raja-arvoon t. Mikäli $bitcount(N_{k,n}) \leq t$, asetetaan tavu $B_k = 255$. Muutoin asetetaan $B_k = 0$. Raja-arvoksi on määritelty lukumäärä, joka kattaa n-naapurustossa bittien enemmistön. Näin syntyy eri mittaisia tavujonoja, jotka sisältävät vuorotellen joko arvon 0 tai 255. Enemmistö kuvaa kyseisten alkioiden yleistä tilaa, joka pysyy tiettyissä rajoissa ennallaan muutoksista riippumatta.

Tavujen mukauttamisen jälkeen syöte muodostuu m tavun jonoista, joissa kaikkien tavujen arvo on 0 tai 255. Tällaiset tavujonot $B = b_0, b_1, b_2, \ldots, b_n$ voidaan esittää RLE-koodattuna tavujen arvon sekä lukumäärän järjestettynä parina. RLE-koodauksen muodostama jono talletetaan Sdhashin tavoin Bloom-suodattimeen. Tiiviste muodostuu Bloom-suodattimesta, ja samankaltaisuus määritetään kahden tiivisteen välisenä Hamming-etäisyytenä. Erityisesti kaikkia suodattimia on verrattava kaikkiin, koska syötteen pituuden muutokset saattavat siirtää ominaisuuksia toiseen suodattimeen (Breitinger, Astebøl, Baier ja Busch, 2013).

Mvhash-b:n pisteytysjärjestelemä poikkeaa Ssdeepin ja Sdhashin vastaavista. Kaikkissa samankaltaisuutta kuvataan suljetulla välillä $[0, 100]$, mutta asteikko on Mvhash-b:ssä käänteinen luvun 0 kuvatessa täydellistä vastaavuutta, eli Hamming-etäisyyttä 0.

4 Soveltaminen haittaohjelmien torjunnassa

4.1 Haittaohjelmien tunnistaminen

Haittaohjelmien torjunta on laaja toimintakenttä, joka pitää sisällääñ toisistaan poikkeavia osa-alueita. Haittaohjelmien tunnistaminen on torjuntatyön keskeinen osa-alue, joka sisältää myös laajan skaalan erilaisia lähestymistapoja. Aslan ja Samet (2020) esittävät haittaohjelma-analyysin jaon staattiseeen sekä dynaamiseen analyysiin. Staattisessa analyysissa tutkitaan haittaohjelmaa suorittamatta sen ohjelmakoodia lainkaan. Dynaamisessa analyysissa keskitytään sen sijaan haittaohjelman käyttäytymisen tutkimiseen ohjelmakoodia suorittamalla.

Tiivistämisellä on merkitystä lähinnä staattisessa analyysissa. Haittaohjelmien tunnistuksessa on tyypillisesti käytetty menetelmänä haittaohjelmista luotujen tiivisteiden vertailua (Sarantinos, Benzaid, Arabiat ja Al-Nemrat, 2016). Tunnettujen haittaohjelmien tiivisteistä pidetään yllä tietokantaa. Mikäli epäily haittaohjelmasta ilmenee, epäilty ohjelma tiivistetään ja sitä verrataan tietokannassa oleviin tiivisteisiin. Yhtäläinen tiiviste viittaa epäíllyn ohjelman olevan lähes varmuudella tunnettu haittaohjelma. Haittaohjelmaksi todettu ohjelma voidaan nyt eliminoida ilman tarvetta ohjelman suorituksenaikaisen toiminnan tarkastellulle dynaamisen analyysin keinoin.

Perinteisesti haittaohjelmien tiivistämiseen on käytetty kryptografisia tiivisteitä, esimerkkeinä MD5 ja SHA-1 (Sarantinos, Benzaid, Arabiat ja Al-Nemrat, 2016). Kryptografiset tiivisteet ovat suorituskykyisiä, mutta niillä ei voida tunnistaa muunneltuja haittaohjelmia. Roussev (2011) mainitsemia kryptografisten tiivisteiden rajoitteita ovat

- Datan havaitseminen toisen datan sisältä
- Lähdekoodiversioiden havaitseminen
- Samaa alkuperää olevien dokumenttien tunnistaminen

On havaittu, että haittaohjelmia laativien tahojen toiminnassa esiintyy toistuvia kaavamaisuuksia. Ilmiö heijastuu levitettävissä haittaohjelmissa, joista löytyy yhteneviä piirtei-

tä. Haittaohjelman valmistaja saattaa hyödyntää samaa tai samankaltaista lähdekoodia useassa haittaohjelmassa, ja tällaisten yhtäläisyyksien havaitseminen rajaa potentiaalisten haittaohjelmien joukkoa pienemmäksi.

4.2 Edellytykset

Ohjelmistojen toteutukset poikkeavat toisistaan, mikä aiheuttaa eroavaisuuksia tehtävistä suoriutumisessa. Ohjelmistoja voidaan luokitella sen perusteella, kuinka ne määrittelevät samankaltaisuuden. Erilaisilla tavoilla tunnistaa samankaltaisuuksia on vahvuutensa sekä heikkoutensa. Tiettyyn tarkoitukseen optimaalisen ohjelmiston valinta on tilannesidonnainen, ja toiset ohjelmistot suoriutuvat paremmin tietyissä tilanteissa kuin toiset (Pagani, Dell'Amico ja Balzarotti, 2018).

Tyypillisesti ohjelmistojen eroja on havainnoitu lähinnä kokeellisesti (Martín-Pérez, Rodríguez ja Breitinger, 2021). Ohjelmistojen lähestymistapojen suuri määrä vaikeuttaa sumeiden tiivisteiden vertailua. Sumeisiin tiivisteisiin liittyvän terminologian suhteen ei ole vallitsevaa konsensusta, eikä universaalia vertailua mahdollistavaa luokittelua ole ollut. Martín-Pérez, Rodríguez ja Breitinger (2021) esittävät luokittelun, joka mahdollistaa muun muassa suorituskykyä, tulosten luotettavuutta sekä hyökkäysalttiutta. tarkastelun.

Oliver (2021) esittää tekijöitä, jotka on syytä ottaa huomioon sumean tiivistämisen ohjelmistoa valitessa.

- Samanakaltaisuuden havaitsemiskyky
- Kyky vastustaa hyökkäyksiä
- Tiivisteen koko ja sen luomiseen kuluva aika
- Skaalautuus

Tarkkuus, jolla samankaltaisuuksia havaitaaan, on tärkeä sumean tiivisteen käyttökelpoisuutta kuvaava mittari. Tarkkuuteen vaikuttaaa toteutusperiaatteen lisäksi syötteenä annetun datan tyyppi. Esimerkiksi konteksiriippuvaisia paloittain määriteltyjä tiivisteitä, kuten ssdeep, ei pidetä soveltuvina lähdekoodianalyysiin (Pagani, Dell'Amico ja Balzarotti, 2018). Epätodennäköisiin ominaisuuksiin perustuvat tiivisteet, kuten Sdhash, sopivat tällaiseen tarkoitukseen paremmin. Haittaohjelma-analyysissa keskeisessä roolissa on konekielisen ohjelmankoodin analysointi, joten valitun ohjelmiston vaikutus on suuri. Myös

Roussev (2011) käsittelee Sdhashin tarkkuttaa, minkä perusteella Sdhashin kyky tunnistaa haittaohjelmia on ylivoimainen Ssdeepiin nähden. Ssdeepin, Sdhasin ja Mvhash-b:n keskinäistä tarkkuuseroa tutkivat Naik, Jenkins ja Savage (2019) käyttäen WannaCry- ja WannaCryptor-haittaohjelmia. Kaikkien kolmen todetaan suoriutuvan tehokkaasti tämäntyyppisten haittaohjelmien tunnistamisesta.

Martín-Pérez, Rodríguez ja Breitinger (2021) esittää tiivisteiden toimintaan vaikuttaviksi tekijöiksi syötteen kattavuuden sekä tiivisteen koon. Ssdeepin ja Mvhash-b:n tiivisteet perustuvat koko syötteeseen, eivätkä muutokset syötteessä jää huomaamatta. Sdhashissa vain pieni osa syötteestä tiivistetään. Osittainen syötekattavuus mahdollistaa hyökkäyksiä, jotka eivät täydellä syötekattavuudella onnistu (Martín-Pérez, Rodríguez ja Breitinger, 2021).

4.3 Haittaohjelmien tunnistamisen välttäminen

Pagani, Dell'Amico ja Balzarotti (2018) tutkivat Ssdeep-, Sdhash-, Tlsh- ja mrsh-v2-ohjelmistojen suoriutumista konekielisen ohjelmakoodin yhtäläisyyksien löytämisessä. Kolmessa tilanteessa analysoitiin kirjastojen tunnistamista, erilaisten käännöskonfiguraatoiden vaikutusta sekä ohjelmien samankaltaisuutta.

Kirjastojen tunnistamisessa on kyse sisällytetyn datan havaitsemisesta suuremmasta datamäärästä. Havaintona esitetään, ettei yksikään ohjelmisto kyennyt yhdistämään kaikkia konekielisten objektitiedostojen sisältämiä kirjastoja kokonaisiin konekielisiin ohjelmiin. Eniten kirjastoja tunnisti Sdhash. Erityisesti heikoiten suorituneen Ssdeepin mainitaan tuottaneen kaikissa tapauksissa pienimmän samankaltaisuusarvon.

Käännöskonfiguraatioiden kohdalla tutkittiin C-kääntäjän (gcc) optimointitasojen sekä eri kääntäjien käytön vaikutusta. Molemmissa tapauksissa Ssdeep suoriutui heikoimmin Sdhashin suoriutuessa paremmin. Ohjelmien samankaltaisuuden tunnistusta vertaillessa ohjelmakoodiin lisättiin sattumanvaraisesti tyhjiä konekäskyjä sekä vaihdettiin sattumanvaraisia konekäksyjä. Havaittiin, että jo muutamat muutokset tuottivat ajoittain käännettyyn ohjelmaan, jota ohjelmistot eivät tunnistaneet. Ilmiöstä todetaan, että pienet muutokset voivat saada aikaan muutoksia myös muualla ohjelmakoodissa johtaen tunnistamattomiin muutoksiin.

Haittaohjelman tunnistamista voidaan vaikeuttaa monin tavoin. Aslan ja Samet (2020) mainistevat useita menetelmiä, joilla haittaohjelman tunnistamista voidaan vaikeuttaa. Me-

netelmiä, jotka aiheuttavat haasteita sumeiden tiivistefunktioiden käytölle tunnistamisessa, ovat muun muassa lähdekoodin salaus, metamorfoosi, naamiointi ja ohjelmiston pakkaus eri tavoin. Kaikissa menetelmissä pyrkimysksenä on minimoida tiivistämisen hyöty staattisessa haittaohjelma-analyysissa. Metamorfisena toteutettu haittaohjelma muokkaa omia konekäskyjään pyrkien todellisuudessa käyttäytymiseen, joka ei suoraan ilmene lähdekoodista staattisen analyysin keinoin (Aslan ja Samet, 2020).

Dataa voidaan naamioida semanttisen merkityksen muuttumatta siten, ettei se kuitenkaan vaikuta samanlaiselta (Oliver, Forman ja Cheng, 2014). Sumeiden tiivisteiden tuloksi pyritään vääristämään naamioimalla dataa eri tavoin. Oliver, Forman ja Cheng, 2014 analysoi Ssdeep-, Sdhash- ja Tlsh-ohjelmistojen herkkyyttä muutoksille. Ohjelmakoodin muutosten analysoinnissa pyrittiin muokkaamaan lähdekooditiedostoja siten, että suoritettava ohjelma on ekvivalentti ja tuottaa tiivisteen, jonka perusteella se ei ole tulkittavissa samankaltaiseksi alkuperäisen kanssa. Tutkimuksessa tutkittiin keinoja, joilla konekielisestä ohjelmakoodista luotua tiivistettä voidaan muuttaa mahdollisimman paljon. Esitettyjä keinoja haittaohjelman naamioimiseksi ovat muun muassa

- Loogisten JA- ja TAI-operaation järjestäminen uudelleen
- Uusien kokonaisluku- ja merkkijonomuuttujien lisääminen
- Funktioiden järjestyksen muuttaminen
- Tyhjien käskyjen lisääminen
- Satunnaisen binäärimuotoisen datan lisääminen
- Merkkijonojen katkaiseminen

Tutkituista ohjelmstoista Ssdeep ja Sdhash eivät tunnistaneet kaikkia muunneltuja ohjelmia, mutta Tlsh kykeni tunnistamaan kaikki. Erityisesti muuttujien lisääminen laski Ssdeep-ohjelmiston tuottamaan vähäisintä yhtäläisyyttä kuvaavan arvon. Muutosten aiheuttamat erot tiivisteissä vaihtelevat menetelmittäin, ja eri ohjelmiston herkkyys muutoksille vaihtelee. Tulosten perusteella Tlsh-ohjelmiston esitetään olevaan sekä Ssdeep- ja Sdhash-ohjelmistoja luotettavampi ja turvallisempi vaihtoehto haittaohjelma-analyysiin.

5 Sumeiden tiivisteiden haasteet

Sumeat tiivisteet tarjoavat ratkaisun samankaltaisuutta tarkasteleviin ongelmiin. Sumeilla tiivisteillä on kuitenkin myös ongelmakohtansa, joita tässä luvussa käsitellään. Ongelmia ilmenee sekä yleisellä tasolla että ohjelmistotasolla. Monet ongelmat vaikuttavat kaikkiin ohjelmistoihin, mutta ohjelmistoissa ilmenee paikallisiakin puutteita. Usein nämä ovat virheitä suunnittelussa tai ajan myötä merkittäviksi kasvaneita puutteita suorituskyvyssä ja luotettavuudessa. Sumeita tiivisteitä käytetään haittaohjelmien torjunnassa, joten niitä vastaan pyritään luonnollisesti hyökkäämään mitä luovimmin keinoin.

5.1 Rajoitteet

Sumeat tiivisteet kärsivät suurehkosta määrästä haasteita. Puutteet eivät kuitenkaan ole helposti kuvattavissa, vaan monet niistä ilmenevät tilanteesta ja ohjelmistosta riippuen. Yleisellä tasolla sumeilla tiivisteillä on rajoitteita, jotka estävät tai hankaloittavat ohjelmiston optioinnin. Ohjelmistoissa taas esiintyy ohjelmistokohtaisia puutteita, jotka ovat korjattavissa.

Keskeinen käytännön rajoite sumeilla tiivisteillä on kyvyttömyys tulkita syötteiden semanteista merkitystä Li, Sundaramurthy, Bardas, Ou, Caragea, Hu ja Jang (2015). Nykyiset sumeat tiivisteohjelmistot tulkitsevat syötteen syntaksitasolla semanttisten rakenteiden tunnistamisen sijaan. Tämä mahdollistaa hyökkäykset tiivisteitä vastaan.

Sumeiden tiiviseteiden käyttöä rajoittaa tulosten subjektiivisuus (Naik, Jenkins ja Savage, 2019). Kehittyneimmänkään ohjelmiston käsitys samankaltaisuudesta ei aina vastaa käyttäjän tulkintaa. Samankaltaisuudelle ei voida määrittää universaalia raja-arvoa ja ohjelmistot ainoastaan arvioivat samankaltaisuutta niille annettujen sääntöjen mukaisesti. Eri ohjelmistot voivat tuottaa samalla syötteellä radikaalistikin poikkeavia tuloksia, joten ohjelmiston valinta on olennaisessa roolissa. Toisaalta eri tulkitsijat voivat myös tulkita saman tuloksen eri tavoin. Samanakaltaisuuden määrittämisessä tiivistysohjelmisto suorittaa ainoastaan kehityksessä määritetyn havainnoinnin ja jättää tulosten tulkistemisen ohjelman käyttäjälle.

Kuva 5.1: Naamitoitava kuva

Kuva 5.2: Ensimmäisessä ja toisessa vaiheessa käsitelty kuva.

5.2 Puutteet ja haavoittuvuudet

Sumeita tiivisteitä voidaan yrittää johtaa harhaan eri tavoin. Edellisessä luvussa käsitellyn haittaohjelman naamioinnin lisäksi menetelmällä, jossa harmitonta dataa muokataan tarkoituksellisesti aiheuttamaan osuma jonkin haitalliseksi todetun sisällön kanssa, voidaan hidastaa haittaohjelmien torjuntaa. Haittaohjelmien leviämistä voidaan tehostaa myös kasvattamalla torjuntatyön kuormittavuutta. Kasvanut työmäärä hidastaa haittaohjelmien tunnistamista. Kuormittavuutta voidaan lisätä pyrkimällä aiheuttamaan perättömiä haittaohjelmaepäilyjä. Tällainen harhautusmahdollisuus on olemassa sumeita tiivisteitä vastaan. Ermerin ja Steenberg (ei julkaisupäivää) esittää tällaisen uhan erityisesti Ssdeepiä vastaan ja tutkii mahdollisuuksia tulosten vääristelyyn.

Ermerin ja Steenberg (ei julkaisupäivää) pyrkivät laatimaan menetelmän, jolla olisi mahdollista luoda proseduraalisesti tunnistamattomia kuvia, joiden samankaltaisuus lähdekuvan kanssa olisi suuri. Esitetty menetelmä muokkaa kuvia kahdessa vaiheessa, joista ensimmäisessä luodaan pikseleitä yksitellen muokkaamalla uusi kuva. Tämän kuvan on tarkoitus olla semanttisesti tunnistamaton, eli ihmisen ei tulisi voida yhdistää niitä toisiinsa. Edellytyksenä kuitennkin on, että Ssdeep merkitsee osumaksi generoidun kuvan samankaltaiseksi alkuperäisen kanssa. Generoidusta kuvasta luodaan toisessa vaiheessa nopeasti suuri määrä hieman toisistaan poikkeavia versioita muuttamalla sattumanvaraisesti valittuja pikseleitä. Kuvassa 5.1 on tutkimuksessa käytetty alkuperäinen kuva; kuvassa 5.2 on alkuperäisen kuvan käsitelty versio.

Kokeessa havaittiin ssdeep-ohjelmiston kyenneen yhdistämään kaikki generoidut kuvatiedostot alkuperäiseen. Lisäksi generoitujen kuvien huomautetaan olevan ilmeisen tunnistettavan näköisiä alkkuperäisisen kuvan kanssa. Siten tavoitetta generoida tunnistamattomia kuvatiedostoja ei saavutettu. Lisäksi huomautetaan kehitetyn menetelmän olleen hidas ja sopimaton käytännön sovelluksiin. Tutkimuksessa on esitetty potentiaalisia optimointikeinoja menetelmän suorituskyvyn kasvattamiseksi.

Datan naamiointia sumeita tiivisteitä vastaan tutkitvat Oliver, Forman ja Cheng (2014). Aiemmin käsitellyn ohjelmien tunnistamisen lisäksi naamiointi suoritettiin kuva- ja tekstitiedostoille sekä HTML-websivuille käyttäen Ssdeep-, Sdhash- ja Tlsh-ohjelmistoja. Kuvien naamioinnissa roskapostituksessa käytettyjä kuvia käännettiin sekä venytettiin, minkä lisäksi kirjasinkokoa ja -lajia muutettiin. Tekstitiedostoja muutettiin lisäämällä, poistamalla ja vaihtamalla sanoja, vaihtamalla rivien paikkaa ja lisäämällä satunnaista dataa. HTML-sivuihin lisättiin tyhjiä merkkejä sallittuihin kohtiin.

Ssdeepin todetaan suoriutuneen heikoimmin kuvien yhdistämisessä; Sdhash ja Tlsh suoriutuivat hyvin. Sdhash ei suoriutunut käännettyjen ja venytettyjen kuvien tunnistamisessa, ja Tlsh tunnisti eniten kuvia. Tekstitiedostojen muuttaminen aiheutti ongelmia Ssdeepille ja Sdhasille, Tlsh suoritui paremmin. 500 muokatusta Html-sivuista Ssdeep ja Sdhash tunnistivat 11 ja 16, ja Tlsh 291. Vertailu osoittaa, että Tlsh suoritui kaikissa tilanteissa merkittävästi paremmin. Havaitaan, että eri ohjelmistojen suoriutumisen välillä on merkittäviä eroja, jotka riippuvat syötteen formaatista. Toiset ohjelmistot reagoivat herkemmin datatyypin muutokseen tai tarkoitukselliseen harhauttamiseen. Lisäksi huomautetaan, että erityisesti ohjelmakoodin muuntelu aiheuttaa em. datatyyppejä suuremman haasteen staattiselle haittaohjelma-analyysille.

5.3 Ohjelmistojen erot

On selvää, että haittaohjelma-analyysissa tutkielmassa käsitelyllyillä tiivisteohjelmistoilla on mwrkittäviä eroja. Kontekstiriippuvaisten paloittain määriteltyjen tiivisteiden tunnistuskyky on heikko, ja Sdhash sekä Mvhash-b vaikuttavat soveltuvan haittaohjelmien tunnistamiseen paremmin. Erityiseti Ssdeepin ja Sdhashin välillä havaitaan merkittävä ero tarkkuudessa sekä ohjelmakoodin että muuntyyppisen datan suhteen.

Martín-Pérez, Rodríguez ja Breitinger (2021) esittävät neljä hyökkäystyyppiä sumeita tiivisteitä vastaan; samankaltaisuuden vähentäminen ja simulointi sekä tiivisteiden generoinnin ja vertailun hankaloittaminen. Hyökkäyksille altistavat tekijät vaihtelevat, ja suurimmassa osassa ohjelmistoja esiintyy haavoittuvuuksia. Tutkielmassa kuvatut ohjelmistot ovat alttiita kaikille hyökkäystyypeille.

Tietyn ohjelmiston suoriutuminen riippuu suurelta osin tilanteen edellyttämistä tarpeista ja käsitellyn datan tyypeistä. Sdhashin vahvuuksiin ei kuulu kokonaisten tiedostojen vertailu, vaan pienempien rakenteiden havaitseminen (Breitinger, Baier ja Beckingham, 2012). Ohjelmistot kärsivät erilaisista puutteista toteutuksessa.

6 Yhteenveto

Samankaltaisuuden tunnistamiseen on ollut tarvetta monessa paikassa, ja aiemmat ratkaisut eivät ole tähän kyenneet. Erityisesti tarvetta on haittaohjelmien tunnistamisessa, jossa on perinteisesti käytetty kryptografisia tiivisteitä. Ongelmaa lähestyvät sumeat tiivisteet mahdollistavat samankaltaisten haittaohjelmien tunnistamisen.

Sumea tiivistys on uudehko tekniikka, mutta sumean tiivistyksen ohjelmistoja on kehitetty laajasti. Eri ohjelmistot lähestyvät samankaltaisuuden tunnistamista poikkeavin tavoin, mikä on tehnyt vertailun ja luokittelun haastavaksi. Ohjelmistoista käsiteltiin tunnetuimpiin kuuluvat ominaisuusjonoja tiivistäväþ Ssdeep ja Sdhash sekä hieman uudempi Mvhash-b, joka keskittyy tavujonoihin.

Kaikki kolme käsiteltyä sumean tiivistyksen ohjelmistoa poikkeavat toteutustavaltaan radikaalisti toisistaan. Poikkeavat esitystavat samankaltaisuudelle vaikeuttavat ohjelmiston valintaa. Toiset ovat päällekkäisiä, toiset taas aivan erilaisia. Ohjelmistojen soveltuvuutta haittaohjelma-analyysin tarpeisiin pohdittiin perinteisten haittaohjelmien tunnistamiskeinojen ja torjuntatyön edellytysten pohjalta; tästä edettiin analysoimaan menetelmiä, joilla haittaohjelmien tekijät voivat harhauttaa sumein tiivistein tehtävää torjuntatyötä.

Lopuksi käsiteltiin rajoitteita, joita sumeat tiivisteet tuovat mukanaan. Yleiseten rajoitteiden lisäksi käesiteltiin Ssdeepin, Sdhashin ja Mvhash-b:n toteutusten puutteita ja hyökkäysalttiutta, sekä verrattiin toteutusten eroavaisuuksien ilmenemistä käytännössä. Läpi käytiin myös sumeiden tiivisteiden haavoittuvuuksia sekä näitä hyödyntäviä potentiaalisia hyökkäyksiä.

Lähteet

Aslan, Ö. A. ja Samet, R. (2020). ”A Comprehensive Review on Malware Detection Approaches”. *IEEE Access* 8, s. 6249–6271. DOI: 10.1109/ACCESS.2019.2963724.

Breitinger, F., Astebøl, K. P., Baier, H. ja Busch, C. (2013). ”mvHash-B - A New Approach for Similarity Preserving Hashing”. Teoksessa: *2013 Seventh International Conference on IT Security Incident Management and IT Forensics*, s. 33–44. DOI: 10.1109/IMF.2013.18.

Breitinger, F., Baier, H. ja Beckingham, J. (elokuu 2012). ”Security and implementation analysis of the similarity digest sdhash”. Teoksessa.

Ermerin, J. ja Steenberg, W. van (ei julkaisupäivää). ”Anti-forencics in ssdeep similarity hashing” ().

Kornblum, J. (2006). ”Identifying almost identical files using context triggered piecewise hashing”. *Digital Investigation* 3. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS ’06), s. 91–97. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2006.06.015. URL: https://www.sciencedirect.com/science/article/pii/S1742287606000764.

Li, Y., Sundaramurthy, S. C., Bardas, A., Ou, X., Caragea, D., Hu, X. ja Jang, J. (elokuu 2015). ”Experimental Study of Fuzzy Hashing in Malware Clustering Analysis”. Teoksessa.

Martín-Pérez, M., Rodríguez, R. J. ja Breitinger, F. (2021). ”Bringing order to approximate matching: Classification and attacks on similarity digest algorithms”. *Forensic Science International: Digital Investigation* 36. DFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference, s. 301120. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301120. URL: https://www.sciencedirect.com/science/article/pii/S2666281721000172.

Martínez, V. G., Álvarez, F. H., Encinas, L. H. ja Ávila, C. S. (2015). ”A New Edit Distance for Fuzzy Hashing Applications”.

Naik, N., Jenkins, P. ja Savage, N. (2019). ”A Ransomware Detection Method Using Fuzzy Hashing for Mitigating the Risk of Occlusion of Information Systems”. Teoksessa: *2019 International Symposium on Systems Engineering (ISSE)*, s. 1–6. DOI: 10.1109/ISSE46696.2019.8984540.

Oliver, J. (2021). *TLSH - A Locality Sensitive Hash*. URL: %7Bhttps://tlsh.org/index.html%7D.

Oliver, J., Forman, S. ja Cheng, C. (marraskuu 2014). ”Using Randomization to Attack Similarity Digests”. Teoksessa: vol. 490, s. 199–210. ISBN: 978-3-662-45669-9. DOI: 10.1007/978-3-662-45670-5_19.

Pagani, F., Dell’Amico, M. ja Balzarotti, D. (maaliskuu 2018). ”Beyond Precision and Recall: Understanding Uses (and Misuses) of Similarity Hashes in Binary Analysis”. Teoksessa: s. 354–365. DOI: 10.1145/3176258.3176306.

Roussev, V. (2010). ”Data Fingerprinting with Similarity Digests”. Teoksessa: *Advances in Digital Forensics VI*. Toim. K.-P. Chow ja S. Shenoi. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 207–226. ISBN: 978-3-642-15506-2.

– (2011). ”An evaluation of forensic similarity hashes”. *Digital Investigation* 8. The Proceedings of the Eleventh Annual DFRWS Conference, S34–S41. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2011.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1742287611000296.

Sarantinos, N., Benzaid, C., Arabiat, O. ja Al-Nemrat, A. (elokuu 2016). ”Forensic Malware Analysis: The Value of Fuzzy Hashing Algorithms in Identifying Similarities”. Teoksessa: s. 1782–1787. DOI: 10.1109/TrustCom.2016.0274.