Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Kontekstiriippuvaiset tiivisteet haittaohjelmien torjuntamenetelmänä

Valtteri Varvikko

4.11.2022

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
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen kandiohjelma

Tekijä — Författare — Author

Valtteri Varvikko

Työn nimi — Arbetets titel — Title

Kontekstiriippuvaiset tiivisteet haittaohjelmien torjuntamenetelmänä

Ohjaajat — Handledare — Supervisors

prof. Nikolaj Tatti

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kanditutkielma	4.11.2022	10 sivua

Tiivistelmä — Referat — Abstract

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

algorithms, hash functions

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Sisällys

1 Johdanto

Tietojenkäsittelyssä käsitellään valtavia määriä dataa. Laskentakyvyn kasvaessa myös tiedonsiirto- ja tallennuskapasiteetti kehittyvät, ja tietojenkäsittelyn kehitys luo myös rikollisia intressejä edistäviä mahdollisuuksia. Sekä tekninen rikostutkinta että kyberrikollisuus ovat riippuvaisia tietotekniikan kehityksestä. Innovaatiot luovat mahdollisuuksia kehittää rikollisia menetelmiä sekä toisaalta ratkaisuja haitallista toimintaa vastaan.

Kasvava datavirta helpottaa haittaohjelmien huomaamatonta leviämistä. Haittaohjelma saattaa jäädä kokonaan huomaamatta, ja haittaohjelmia voidaan tietoisesti naamioida aiheuttamaan harhaanjohtavia tuloksia torjunnassa. Haitallisen datan tunnistaminen kasvavasta datavirrasta edellyttää kohdennettuja resursseja ja suorituskykyisiä ratkaisuja. Tiedon esittämislle kompaktimmassa muodossa onkin tarvetta sen tehokkaan tunnistamiseen vuoksi.

Dataa voidaan tiivistää muun muassa pakkausalgoritmeilla ja tiivistefunktioilla. Pakkauksesta poiketen tiivistefunktiot tiivistävät datan palauttamattomaan muotoon. Toisaalta tiiviste sisältää lähes uniikin tunnisteen, jolla alkuperäinen data on käytännössä tunnistettavissa. Tiivistystä voidaankin hyödyntää keinona haittaohjelmien havaitsemisessa suuresta datamäärästä. Tyypillisesti tunnistetuista haittaohjelmista on luotu tiiviste, jolloin saman tiivisteen tuottavaa ohjelmaa voidaan epäillä haittaohjelmaksi. Mikä tahansa muutos tuottaa kuitenkin tunnistamattoman tiivisteen kryptografiset tiivistefunktioilla, joita tarkoitukseen on perinteisesti käytetty. Rikolliset tahot pyrkivät kiertämään haittaohjelmien torjuntamenetelmiä esimerkiksi tekemällä ohjelmistoon muutoksia, jolloin niitä ei voida enää tunnistaa tiivisteitä vertailemalla.

Seuraavassa luvussa käsitellään yleisesti tiivistefunktioita sekä taustoitetaan aiheeseen. Luvussa 3 käydään läpi kontekstiriippuvaisten tiivisteiden periaate sekä verrataan keskisiä toteutuksia. Luvussa 4 käsitellään konteksiriippuvaisten tiivisteiden soveltamista haittaohjelmien torjuntaan sekä esiin nousevia haasteita. Luvussa 6 esitetään konteksiriippuvaisten tiivistefunktioiden heikkouksia sekä analysoidaan haavoittuvuuksien hyväksikäytössä sovellettuja menetelmiä.

2 Tiivisteet

2.1 Tiivisteet haittaohjelmien torjunnassa

Kryptografisia tiivisteitä on käytetty teknisen rikostutkinnan menetelmänä (Kornblum, 2006). Kryptografiset tiivistefunktiot tuottavat samalla syötteellä identtisen tiivisteen. Syötettä prosessoidessa jokainen alkio muuttaa generoitavan tiivisteen tunnistamattomaksi. Identtiset syötteet ovat tunnistettaviksi identtisiksi yhtäläisen tiivisteen perusteella, mutta yhdenkin bitin muutos tuottaa uuden, tunnistamattoman tiivisteen.

Haittaohjelmien torjunnassa on perinteisesti käytetty yleisiä kryptgrafisia tiivistefunktioita, kuten MD5 ja SHA-1. Torjunnassa kerätään tunnettujen haittaohjelmien tiivisteitä, jolloin nämä voidaan suurella varmuudella tunnistaa haittaohjelmiksi (Kornblum, 2006).

2.2 Paloittain määritelty tiiviste

Tyypillisesti yksittäisen hyökkääjätahon menettelyssä esiintyy kaavamaisia piirteitä, ja hyökkääjä saattaa esimerkiksi käyttää paikoitellen samaa koodia eri ohjelmissa (Naik, Jenkins ja Savage, 2019). Samankaltaisuuksia havaitsevia tiivisteitä voidaan luoda keskittymällä koko syötteen sijaan sen pienempiin osioihin. Olennaista ei ole tunnistaa täsmälleen tietynlainen syöte, vaan löytää syötteestä haluttuja rakenteita.

Yksinkertainen sumea tiiviste voidaan laatia jakamalla syöte halutun kokoisiin lohkoihin. Lohkokoko n on vakio kaikille syötteen lohkoille. Lohkot iteroidaan generoiden samalla jokaiselle lohkolle tiiviste. Lohkon tiiviste riippuu tällöin ainostaan lohkon sisältämien alkioiden arvoista. Varsinainen tiiviste on lohkoille generoitujen tiivisteiden muodostama jono. Formaalisti jokainen alkio a_i, missä $n \mid i$, sekä tätä edeltävät alkiot $a_{i-1}, a_{i-2}, a_{i-3}, \ldots, a_{i-n}$, tiivistetään ja saatu tiiviste sijotetaan tiivistejonoon. Syntynyttä tiivistejonoa voidaan vertailla muihin kuvatusti generoituihin tiivisteisiin, ja homologiat ilmenevät tiivistejonoissa identtisinä tiivisteinä; poikkeavat tiivisteet taas viittaavat eroihin syötteiden välillä.

Taulukossa 2.1 on kuvattu yksinkertaista paloittain määritellyn tiivisteen generoiva algoritmia, jossa syöte jaetaan jaetaan kolmen alkion lohkoihin; lohkoille generoidaan tiiviste

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

Taulukko 2.1: Paloittain määritelty tiiviste merkkijonolle.

CRC-8-tiivistefunktiolla ja tiivisteistä muodostetaan jono. Taulukossa 2.2 on kuvattu muutoksia, jotka tiivisteisiin ja tiivistejonoon aiheutuvat kun syötteeseen tehdään muutoksia. Tässä tapauksessa alkioita on ainoastaan korvattu toisella, ja syötteen pituus säilyy muuttumattomana. Muutokset heijastuvat ainostaan lohkoihin, jotka sisältävät muutettuja alkioita; muiden lohkojen tiivisteet säilyvät muuttumattomina, ja tiivistejonoja vertaamalla voidaan löytää syötteissä esiintyvät homologiat.

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

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

Menetelmä säilyttää tunnisteet homologioista, kun syöteen pituus ei muutu. Kuitenkin alkioita lisättäessä ja poistettaessa muutokset ilmenevät myös muissa lohkoissa tuloksen vääristyessä pahasti. Alkion lisäys tai poisto muuttaa seuraavien alkioiden indeksiä, jolloin osa alkioista siirtyy toiseen lohkoon. Lohkon alkiojonon muuttuessa lohkon tiivistekin muuttuu, vaikka alkioiden arvot ovat pysyneet muuttumattomina. Tätä ilmiötä on havainnollistettu taulukossa 2.3; kolmannen alkioen jälkeen sijoiitettu uusi alkio muuttaa kaikkien seuraavien lohkojen tiivisteet muuttumattomaksi.

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

Taulukko 2.3: Palottain määritelty tiiviste merkkijonolle, johon on lisätty uusi merkki.

Algoritmin palauttama tiivistejono saattaa merkittävästi poiketa alkuperäiselle syötteelle generoidusta tiivistejonosta, ja sattumanvaraisen syötteen alusta poistettu alkio voi tuottaa kryptografisen tiivistefunktion tavoin aiemmasta täysin tunnistamattoman tiivisteen. Tiiviste muuttuu sitä enemmän, mitä pienempään syöþteen indeksiin alkion lisäys tai poisto kohdistuu. sLohkokoon on syytä riippua alkioiden muodostamasta kontekstista; alkiojonon $p_i, p_{i+1}, p_{i+2}, \ldots, p_j$ semanttinen merkitys pysyy samana indekseistä riippumatta siitä, mihin indeksiin alkion lisäys tai poisto kohdistuu.

3 Kontekstiriippuvaiset tiivistefunktiot

Kontekstiriippuvaiset tiivistefunktiot on eräs paikallisherkkien tiivistefunktioiden ryhmä, jonka keskeinen periaate on prosessoida syöte keskittyen alkioiden kontekstiin. Useat kontekstiriippuvaiset tiivistefunktiot jakavat syötteen vakiokokoisten lohkojen sijaan vaihtelevan kokoisiin lohkoihin, jotka määräytyvät dynaamisesti syötteen sisällön perusteella. Tällöin alkion lisäys tai poisto vaikuttaa ihanteellisesti yhden, ja enimmilläänkin vain lähimpien lohkojen tiivisteeseen eikä heijastu muihin lohkoihin.

3.1 Rekursiivinen tiiviste

Kontekstin tunnistaminen onnistuu rekursiivisella tiivistefunktiolla. Rekursiivinen tiiviste lasketaan syötettä iteroidessa jokaiselle alkiolle rekursiivisesti huomioiden muut tietyn ehdon täyttävät alkiot (Kornblum, 2006). Usein tämä ehto tarkoittaa syötteen viimeisimpiä alkiota. Rekursiivisen tiivisteen h generointia syötteen kohdassa i huomioiden n edeltävää alkiota voidaan havainnollistaa tiivistefunktion R rekursiivisena kutsuna

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

Tiivistettä laskiessa ei käytännössä ole kuitenkaan tarpeen laskea jokaiselle edelliselle alkiolle tiivistettä, koska viimeisin tiiviste on jo laskettu edeltävistä alkioista. Uusi tiivisteon laskettavissa hyödyntämällä viimeksi generoitua tiivistettä, josta poistetaan tiivistefunktiosta riippuen vanhimman alkion vaikutus esimerkiksi loogisilla operaatioilla.

Kontekstin määrityksessä tarkkaillaan rekursiivisen tiivistefunktion generoimaa tiivistettä. Rekursiivista tiivistettä verrataan ennalta määritellyn laukaisinarvoon, ja yhtäsuuruus indikoi kontekstimuutosta. Tällöin lohkolle generoidaan tiiviste yleensä staattisella tiivistefunktiolla (MD5, SHA-1, Fowler-Noll-Vo ym.).

3.2 Toteutuksista

3.2.1 ssdeep

ssdeep on laajasti käytetty ohjelmisto kontekstiriippuvaisten tiivisteiden generointiin sekä käsittelyyn (ssdeep Project, 2018). ssdeep-ohjelmistoa voidaan käyttää konteksiriippuvaisten tiivisteiden generointiin sekä olemassaolevien tiivisteiden vertailuun. ssdeep generoi syötteestä konteksiriippuvaisen tiivisteen joka koostuu peräkkäisistä lohkoille generoiduista tiivisteistä sekä käytetystä rekursiivisen tiivistefunktion laukaisinarvon käytetystä lohkokoosta. Kuvassa 3.1 on esimerkki ssdeep-ohjelmiston käytösta kahden samankaltaisella JavaScript-lähdekooditiedostolla; molemmille generoidaan tiivisteet, minkä jälkeen tiedostoja samankaltaisuus määritetään.

```
$ ssdeep -l functional.js functional-uncommented.js
ssdeep,1.1--blocksize:hash:hash,filename
192:eQiHcd9A+QKxYI1LIgbKvgiuqBOML2lJGnYDTo/Z:Acd9A+QKlLpbKJO1lJQYDGZ,
  "functional.js"
192:oQ+BA+QKNYI1LIPuiuqWOML2lJGnYDTo/Z:KA+QKJLpO1lJQYDGZ,
   "functional-uncommented.js"

$ ssdeep -l -r -d functional.js functional-uncommented.js
functional-uncommented.js matches functional.js (75)
```

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

ssdeepiä käytetään laajasti monenlaisiin käyttötarkoituksiin ollen alansa de facto -standardi (ssdeep Project, 2018). Eräs keskeinen alue on tietoturva, jossa ssdeepiä sekä vastaavia ohjelmistoja käytetään tunnistamaan haittaohjelmia. ssdeep voi tunnistaa muita, saman tahon laatimia haittaohjelmia yhdestä, haittaohjelmaksi todetun ohjelman lähdekoodista generoidusta tiivisteestä vertaamalla sitä em. epäiltyjen haittaohjelmien tiivisteisiin.

4 Kontekstiriippuvaisten tiivisteiden soveltaminen haittaohjelmien torjunnassa

4.1 Edellytykset

4.2 Tulosten hyödyntäminen

4.3 Skaalautuvuus

5 Haavoittuvuudet ja hyökkäykset

5.1 Harhaanjohtaminen dataa manipuloimalla

Haittaohjelmien torjuntatyötä pyritään vaikeuttamaan erilaisin menetelmin. Haittaohjelmia pyritään toiminnallisuus säilyttäen naamioimaan tunnistamisen minimoimiseksi. Aslan ja Samet, 2020 mainitsee useita dynaamista analyysia vaikeuttavia menetelmiä, mutta esimerkiksi ohjelmiston pakkaus ja salaus heikentävät myös sumeiden tiivisteiden tehokkuutta.

5.1.1 Aiheettomat osumat

Haittaohjelmien leviämistä voidaan tehostaa kasvattamalla torjuntatyön kuormittavuutta. Kasvanut työmäärä hidastaa haittaohjelmien tunnistamista. Kuormittavuutta voidaan lisätä pyrkimällä aiheuttamaan perättömiä haittaohjelmaepäilyjä. Ermerins, Steenbergen ja Ginkel, ei julkaisupäivää esittää tällaisen uhan ja tutkii mahdollisuuksia ssdeep-ohjelmiston tulosten vääristelyyn.

Tutkimuksessa tavoitteena oli laatia menetelmä, jolla olisi mahdollista generoida tunnistamattomia kuvia, joiden samankaltaisuus lähdekuvan kanssa olisi suuri. Esitetty menetelmä muokkaa kuvia kahdessa vaiheessa, joista ensimmäisessä luodaan alkuperäisestä semanttisesti tunnistamaton kuva, jonka ssdeep kuitenkin merkitsee osumaksi. Toisessa vaiheessa tästä kuvasta luodaan tehokkaasti useita uusia, toisistaan poikkeavia kuvia muuttamalla sattumanvaraisesti valittuja pikseleitä.

Kokeessa havaittiin ssdeep-ohjelmiston kyenneen yhdistämään kaikki generoidut kuvatiedostot alkuperäiseen. Tavoitetta generoida tunnistamattomia kuvatiedostoja ei kuitenkaan saavutettu, ja generoitujen kuvatiedostojen huomautetaan olevan ilmeisen tunnistettavan näköisiä.

5.1.2 Naamiointi osumien välttämiseksi

Dataa voidaan naamioida semanttisen merkityksen muutumatta siten, ettei se kuitenkaan vaikuta samankaltaiselta. (Aslan ja Samet, 2020). Haittaohjelmia naamioimalla pyritään

vääristämään sumeiden tiivistefunktioiden tuloksia. Eri ohjelmistojen sietokyky muutoksille vaihtelee (Oliver, Forman ja Cheng, 2014).

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 kansas. Kokeessa tutkittiin useita menetelmiä, joilla konekielisestä ohjelmakoodista luotua tiivistettä voidaan muuttaa mahdollisimman paljon. Taulukossa 5.1 on listattu erilaisia menetelmiä lähdekoodin naamioimiseksi.

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

Taulukko 5.1: Ohjelmakoodin naamioinnissa käytettyjä menetelmiä (Oliver, Forman ja Cheng, 2014)

Ohjelmstoista ssdeep ja SDHASH eivät tunnistaneet kaikkia modifitoituja lähdekooditiedostoja, mutta TLSH tunnisti kaikki. Erityisesti muuttujien lisäämine laski ssdeep-ohjelmiston tuottamaan matalimman arvon. Muutosten aiheuttamat erot tiivisteissä vaihtelevat menetelmittäin, ja eri ohjelmiston herkkyys muutoksille vaihtelee. Tulosten perusteella TLSH-ohjelmiston esitetään olevaan ssdeep- ja SDHASH-ohjelmistoja vakaampi ja turvallisempi

6 Yhteenveto

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.

Ermerins, J., Steenbergen, W. van ja Ginkel, J. van (ei julkaisupäivää). ”Anti-forensics 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.

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., Forman, S. ja Cheng, C. (2014). ”Using Randomization to Attack Similarity Digests”. Teoksessa: *Applications and Techniques in Information Security*. Toim. L. Batten, G. Li, W. Niu ja M. Warren. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 199–210. ISBN: 978-3-662-45670-5.

ssdeep Project (2018). *ssdeep*. URL: https://ssdeep-project.github.io/ssdeep/index.html (viitattu 10. 10. 2020).