Pätkäistyt hajautusarvot

Valtteri Varvikko

13. lokakuuta 2022

Tietojenkäsittelyssä käsitellään valtavia määriä dataa. Tiedon esittämiselle kompaktimmassa muodossa on laajalti tarvetta, milloin tallennuskapasiteetin säästämisen, milloin vastaavan datan tehokkaan tunnistamiseen vuoksi. Tiivistää voidaan muun muassa pakkausalgoritmeilla ja hajautusfunktioilla. Pakkauksesta poiketen hajautusfunktiot tiivistävät datan palauttamattomaan muotoon. Toisaalta hajautusarvo sisältää uniikin tunnisteen, jolla alkuperäinen data on tunnistettavissa. Merkityksettöminkin muutos tuottaa täysin erilaisen tunnisteen, eivätkä syötteet ole enää linkitettävissä toisiinsa. Semanttisesta näkökulmasta muutos voi säilyttää datan samankaltaisena, ja ihmissilmä löytää suurestikin poikkevista tiedostoista yhtäläisyyksiä. Samankaltaisuuksien tunnistamiseen kryptografisista tiiviistefunktioista ei ole, mutta niitä hyödyntävät *fuzzy hash* -algoritmit mahdollistavat yhtenäisten rakenteiden havaitsemisen.

Piecewise hashing

Kryptografiset hajautusfunktiot tuottavat samalla syötteellä saman tuloksen. Syötettä prosessoidessa jokainen alkio muuttaa generoitavan hajatusarvon edellisestä tunnistamattomaksi — tämä ilmiö on olennainen kryptografisissa sovelluksissa ja siten tarkoituksellinen. Identtiset syötteet ovat todettavissa identtisiksi yhtäläisen hajautusarvon perusteella, mutta yhdenkin bitin muutos tuottaa edellisestä tunnnistamattoman hajautusarvon. Näin ollen samankaltaiset syötteet eivät ole tunnistettavissa samankaltaisiksi kryptografisilla hajautusfunktiolla.

Tiedostoissa esiintyviä samankaltaisuuksia kutsutaan *homologioiksi* (Kornblum, 2006). Homologioiden tunnnistaminen onnistuu hajautusarvoja vertaamalla, mutta tässä tapauksessa hajautusarvo tulee olla tunniste syötteessä esiintyvistä homologioista koko syötteen sijaan. Tätä tavoitetta tukemaan on laadittu kryptografisista hajautusfunktioista poikkeava algoritmityyppi. *Fuzzy hash* -

algoritmit tulkitsevat syötettä tarkastellen useampaa alkiota kerralla, joten alkion hajautusarvoon vaikuttaa myös naapurialkioiden arvo. Näin eroavaisuudet syötteissä aiheuttavat muutoksia hajautusarvoon ainoastaan paikallisesti, ja hajautusarvo säilyy muilta osin lähestulkoon muuttumattomana.

Pätkäisty hajautusarvo voidaan toteuttaa esimerkiksi algoritmilla, joka jakaa syötteen n alkion osajonoiksi. Alkio a_i, missä $i \equiv 0 \pmod{n}$, sekä sitä edeltävät alkiot $a_{i-1}, a_{i-2}, \ldots, a_{i-n}$ redusoidaan esimerkiksi yhteenlaskulla skalaariksi, josta generoidaan tälle osajonolle hajautusarvo. Hajautusarvo lasketaan myös syötteen loppuun mahdollisesti jäävälle n alkiolle. Generoidut hajautusarvot palautetaan jonona, jota voidaan vertailla muihin kuvatusti generoituihin hajautusarvoihin. Homologiat ilmenevät hajautusarvojonossa identtisinä hajautusarvoina; poikkeavat hajautusarvot taas viittaavat eroihin syötteiden välillä.

Context triggered piecewise hashing

Edellä kuvattu ratkaisu toimii kohtalaisesti samanmittaisille syötteille, joissa alkioita on ainoastaan korvattu toisella kajoamatta syötteen pituuteen. Tulos kuitenkin häiriintyy pahasti, jos syötteen koko muuttuu alkuperäiseen verrattuna alkioita lisättäessä tai poistettaessa. Muiden alkioiden sijainti syötteessä vaihtuu, ja seuraavat osajonot sisältävät useimmiten alkuperäisestä poikkeavan alkiojonon. Muutokset ulottuvat syötteen loppuun asti, jolloin jokainen muuttunut osajono tuottaa alkuperäisestä poikkeavan hajautusarvon. Algoritmin palauttama hajautusarvojono saattaa merkittävästi poiketa alkuperäiselle syötteelle generoidusta hajautusarvostajonosta, ja sattumanvaraisen syötteen alusta poistettu alkio tuottanee kryptografisen hajautusfunktion tavoin aiemmasta täysin tunnistamattoman hajautusarvon. Hajautusarvo muuttuu sitä enemmän, mitä pienempään syötteen indeksiin alkion lisäys tai poisto kohdistuu.

Context triggered piecewise hashing (CTPH) on tekniikka *fuzzy hash* -algoritmien toteuttamiseen. Välttääkseen edellä kuvatun ongelman CTPH-algoritmit jakavat syötteen vakiokokoisten osajonojen sijaan vaihtelevan kokoisiin osajonoihin, jotka määräytyvät dynaamisesti syötteen sisällön perusteella. Tällöin alkion lisäys tai poisto vaikuttaa ihanteellisesti yhden, ja enimmilläänkin vain lähimpien osajonojen hajautusarvoon eikä heijastu muihin osajonoihin.

Monissa CTPH-algoritmeissa osiointi perustuu *laukaisimiin (trigger)*, joilla määritetään osajonon päätepiste alkion kontekstin perusteella (Kornblum, 2006). Laukaisin aktivoituu tietyllä syötteestä generoidulla arvolla ja ilmaisee määritellyn kokonaisuuden päättymistä. Käytännössä konteksti tarkoittaa alkion lähimmissä

indekseissä olevia alkioita, joten kontekstille generoitu hajautusarvo on riippuvinen useamman alkion yhdessä määrittämästä kokonaisuudesta. Etu *piecewise hasheihin* on syötteen osiointi kontekstin perusteella vakiokokoisten osajonojen sijaan.

Kontekstin tunnistaminen onnistuu *rolling hash* -hajautusfunktiolla. Hajautusarvo lasketaan syötettä iteroidessa jokaiselle alkiolle rekursiivisesti huomioiden muut tietyn ehdon täyttävät alkiot. Usein tämä ehto tarkoittaa syötteen viimeisimpiä alkiota. Hajautusarvon h generointia alkiolle a_i sekä viimeisimmille n alkiolle voidaan havainnollistaa hajautusfunktion R rekursiivisena kutsuna

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

Arvoa laskiessa ei käytännössä ole tarpeen laskea jokaiselle edelliselle alkiolle hajautusarvoa, koska viimeisin hajautusarvo on jo laskettu edeltävistä alkioista. Uusi hajautusarvo on laskettavissa hyödyntämällä viimeksi generoitua hajautusarvoa, josta poistetaan hajautuksesta riippuen vanhimman alkion vaikutus esimerkiksi loogisilla operaatioilla.

Mikäli *rolling hash* aktivoi laukaisimeen, hajautusarvojonoon lisätään vallitsevan osajonon alkioille generoitu hajautusarvo. Hajautusarvon generointiin käytetään tyypillisesti perinteistä hajautusfunktiota, kryptografisia (MD5, SHA-1 tms.) tai ei-kryptografisia (FNV tms.). Kuvattua menetelmää toistaen syötteestä muodostuu hajautusarvojono, joka sisältää syötteen osajonoista generoidut, toisistaan riippumattomat hajautusarvot.

Toteutuksista

CTPH-toteutuksia on markkinoilla useita, kuten ssdeep, SDHASH ja mvHASH-B. Eri algoritmien toteutukset voivat poikkeavat toisistaan, ja osa lähestyy yhteistä ongelmaa erilaisesta näkökulmasta. Tästä huolimatta algoritmit jakavat yhteisen perusajatuksen, eli syntaksisten homologioiden löytämisen syötteestä.

spamsum Eräs varhainen CTPH-hajautusfunktio, *spamsum*, tunnistaa syötteissä esiintyviä samankaltaisuuksia. spamsum-algoritmin totetus perustuu CPTH-tekniikan ajatukseen syötteen käsittelystä konteksiriippuvaisesti ja seuraa pääpiirteissään edellä esitettyä kuvausta CTPH-algoritmin toiminnasta. Suorituksen alussa spamsum määrittää laukaisimen arvoksi *block sizen*, joka lasketaan syötteen perusteella. Syöttettä iteroitaessa *rolling hashia* päivitetään alkiolle FNV-tiivistefunktion generoimalla tiivistellä (Kornblum, 2006).

Hajautusarvojen vertailu vaatii näiden generoinnissa käytetyn *block sizen*. Koska *block size* on riippuvainen syötteestä, se palautetaan yhdessä hajautusarvon kanssa. spamsum iteroi syötteen kahdesti, sekä *block sizella* b että $2b$. Näin saadaan parannettua hajautusarvojen vertailumahdollisuutta, ja hajautusarvoja i ja j vertaillessa on täytyttävä joko $b_i = b_j$, $2b_i = b_j$ tai $b_i = 2b_j$. Hajautusarvoa i voi siis verrata toiseen hajautusarvoon, jonka *block size* on joko b_j tai kahden potenssi.

ssdeep spamsum-algoritmin pohjalta on kehitetty ssdeep-ohjelmisto konteksiriippuvaisten hajautusarvojen käsittelyyn (ssdeep Project, 2018). ssdeepin hajautusarvojen generointi toimii jokseenkin samoin kuin spamsumin. ssdeep generoi syötteestä konteksiriippuvaisen hajautusarvon. spamsumin tavoin hajautusarvo koostuu peräkkäisistä osajonoille generoiduista hajautusarvoista sekä käytetystä *block sizesta*.

ssdeepiä käytetään laajasti monenlaisiin käyttötarkoituksiin ollen alansa de facto -standardi (ssdeep Project, 2018). Eräs keskeinen alue on tietoturva, jossa ssdeepiä sekä vastaavia ohjelmistoja voidan käyttää tunnistamaan haittaohjelmia. Tyypillisesti yksittäisen hyökkääjätahon menettelyssä esiintyy kaavamaisia piirteitä, ja hyökkääjä saattaa esimerkiksi käyttää paikoitellen samaa koodia eri ohjelmissa (Naik ym., 2019). ssdeep voi tunnistaa muita, saman tahon laatimia haittaohjelmia yhdestä, haittaohjelmaksi todetun ohjelman lähdekoodista generoidusta CTPH:sta vertaamalla sitä em. epäiltyjen haittaohjelmien hajautusarvoihin.

Heikkoudet *Fuzzy hash* -algoritmien havaitaan suoriutuvan kiitettävästi (Naik ym., 2019). Erityisesti ssdeep-funktion mainitaan kykenevän yhdistämään laajastikin muuttettuja sekä puuttellisia tiedostoja alkuperäisiin. *Fuzzy hashing* ei kuitenkaan ole aukoton tekniikka. Nämä algoritmit ovat vaikeasti ennustettavia, eikä syötteiden samankaltaisuudelle ole johdettavissa yleistä raja-arvoa, jonka alittaessaan syötteet eivät ole enää samankaltaisia. Samankaltaisuus tietyssä tilanteessa ei ole laskettavissa oleva ongelma, vaan ohjelman käyttäjän subjektiivinen näkemys. Tulkintaa vaikeuttaa myös algoritmien syntaksiin perustuvaan analyysiin — hajautusarvojen vertailu ei kerro syötteen semanttisesta kontekstista.

Datan pakkaus aiheuttaa merkittäviä haasteita *fuzzy hash* -hajautusfunktioille (Naik ym., 2019). Ne eivät juurikaan kykene käsittelemään samankaltaisesta datasta generoituja hajautusarvoja. Eräs keskeinen *fuzzy hashien* käyttökohde on

tunnistaa haittaohjelmia, joiden levitys keskittyy merkittävissä määrin juuri tietoliikenteeseen jonka pakatun datan analysoinnin automatisointi ei onnistune.

Fuzzy hashing ei ole innovaationa tuoreimmasta päästä, mutta sittemmin sille on luotu monenlaisia käyttökohteita. Enenevissä määrin tarvitaan eksaktin yhtäläisyyden lisäksi samankaltaisuuksien tunnistamista ja luokittelua. CTPH-algoritmeilla voidaan vähentää toisteisuutta, tunnistaa haittaohjelmia tai löytää puutteellisen tiedoston alkuperäinen kappale rikostutkinnassa. Huomioitavaa on, että tulokset ovat numeerisia parametrejä, joista ei voi tilannesidonnaisuuden vuoksi johtaa yleispätevää mallia. Algoritmi tunnistaa ainoastaan homologiat, ihmisen tulee edelleen päättää, mitä saadusta tuloksesta kussakin tilanteesssa seuraa.

Viitteet

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing [The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06)]. *Digital Investigation*, *3*, 91–97. https://doi.org/https://doi.org/10.1016/j.diin.2006.06.015

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

ssdeep Project. (2018). *ssdeep*. Haettu lokakuuta 10, 2010, osoitteesta https://ssdeep-project.github.io/ssdeep/index.html