Paavo Pörhö

Datan tarkistussummia tukevat tiedostojärjestelmät ja niiden edut datan oikeellisuuden tarkistamisessa

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Paavo Pörhö

Yhteystiedot: paavo.porho@iki.fi

Ohjaaja: Tuomo Rossi

Työn nimi: Datan tarkistussummia tukevat tiedostojärjestelmät ja niiden edut datan oikeellisuuden tarkistamisessa

Title in English: File systems supporting data checksums and their advantages in verifying data integrity

Työ: Kandidaatintutkielma

Sivumäärä: 20+0

Tiivistelmä: Tutkielmassa selvitetään datan eheystarkistuksia tukevien tiedostojärjestelmien etuja datan oikeellisuuden tarkistamisessa. Tutkielmassa käydään myös läpi vaihtoehtoisia toteutuksia ja niiden etuja kyseisiin tiedostojärjestelmiin nähden.

Avainsanat: tiedostojärjestelmä, zfs, btrfs, resilient file system, tarksitussumma, bittivirheiden määrä, virheenkorjauskoodi, tiedostojärjestelmän yhtenäisyys, tiedostojärjestelmän tarkistus, eheyden tarkistus

Abstract: This thesis explores the benefits of file systems that support data integrity checks in verifying the integrity of data. It also discusses alternative implementations and their advantages over these file systems.

Keywords: file system, zfs, btrfs, resilient file system, checksum, bit error rate, error correction code, file system consistency, file system checking, integrity checking

Jyväskylässä 30. huhtikuuta 2023

Sisällys

1 Johdanto

Tutkielmassa selvitetään datan eheystarkistuksia tukevien tiedostojärjestelmien etuja datan oikeellisuuden tarkistamisessa. Lisäksi käsitellään tiedostojärjestelmien ulkopuolella toteutettavan datan oikeellisuuden tarkistamisen etuja tiedostojärjestelmiin nähden.

Datan eheystarkistuksia tukevat tiedostojärjestelmät ovat harvoin käytössä suosituimmissa käyttöjärjestelmissä. Suurin osa rajautuu yhteensopivuudeltaan Unix-kaltaisiin käyttöjärjestelmiin, joista suosituimpien joukossa ovat Btrfs, ZFS ja ReFS. Btrfs on kehitetty Linux-järjestelmiin (Wani ja Bhat 2018) ja ZFS on portattu Solaris-käyttöjärjestelmästä yleisimpiin Unix-järjestelmiin (Beebe, Stacy ja Stuckey 2009). ReFS on kehitetty Windows-järjestelmille, muttei ole vielä korvannut NTFS-tiedostojärjestelmää (Lee ym. 2021). Täten suuri osa datan oikeellisuuden tarkistamisesta rajoittuu tiedostojärjestelmien ulkopuolelle.

Tutkimusta rajataan niin, että datan palautusta ja RAID-konfiguraatioita ei käsitellä. Nämä rajautukset huomioon ottaen tutkimuskysymyksenä on: mitä etuja datan oikeellisuuden tarkistamiseen saadaan datan eheystarkistuksia tukevalla tiedostojärjestelmällä, jos sama voidaan saavuttaa muilla toteutuksilla?

Tutkielmassa käsitellään ensin yleisimpiä datan oikeellisuutta tarkistavia tiedostojärjestelmiä, jonka jälkeen käsitellään kyseisten tiedostojärjestelmien tukemia tarkistussummia. Tämän jälkeen tutkitaan tiedostojärjestelmien ulkopuolella toimivia datan tarkistamisen toteutuksia.

2 Tiedostojärjestelmät

Tiedostojärjestelmän tehtävänä on esittää tallennusvälineelle tallennettu data tiedostoina ja hoitaa tiedostojen jakautuminen ja metadata tallennusvälineen sektoreilla. Linux-jakeluissa yleisesti käytettävät tiedostojärjestelmät Ext2 ja Ext3 toteuttavat tämän 1024, 2048 ja 4096 tavun kokoisilla lohkoilla, joista metadatan ja datan sisältävät lohkoryhmät koostuvat. Metadataan tallennetaan muun muassa tiedoston inode, joka sisältää tiedostoon liittyvän metatiedon, kuten aikaleiman, käyttöoikeudet ja kohdistimet tiedoston datalohkoihin (Fairbanks 2012, luku 2.1).

2.1 Ext4

Ext4 eroaa äskeisistä ExtX-tiedostojärjestelmistä tarkistussummien käytössä. Toisin kuin ExtX, Ext4 hyödyntää CRC32-tarkistussummia journaloinnissa (Fairbanks 2012, luku 4.5) ja CRC32C-tarkistussummia metadatan yhteydessä (“2. High Level Design — The Linux Kernel documentation”, n.d.). ExtX:n tapaan Ext4 ei luo datasta tarkistussummia.

2.2 ZFS

ZFS on alun perin Solaris-käyttöjärjestelmälle kehitetty datan eheyttä tukeva tiedostojärjestelmä, mutta on saapunut muun muassa FreeBSD-, NetBSD-, OS X-, ja Linux-käyttöjärjestelmiin kernel- ja käyttäjätason portteina (Beebe, Stacy ja Stuckey 2009, luku 1). Toisin kuin Ext4, ZFS luo sekä datasta että sen metadatasta tarkistussummat. Lohkojen muutokset käsitellään myös turvallisemmin copy-on-write transaktionaalisella objektimallilla, jolloin sähkökatkokset ja muut järjestelmäviat eivät korruptoi dataa (Beebe, Stacy ja Stuckey 2009, luku 2.1).

ZFS:n lohkot koostavat puurakenteen, joka sisältää tietoa lehtisolmuissa sijaitsevan datan tarkistussummista. Tarkistussummat tallennetaan lohko-osoittimien yhteyteen, jolloin puurakenteessa kulkeminen mahdollistaa myös samanaikaisen datan verifikaation (Rodeh, Bacik ja Mason 2013, luku 2.1). Kirjoitettava data sijoitetaan ZFS:ssä ExtX:ää turvallisemmin.

ExtX sijoittaa sekä tiedoston metadatan että datan samaan lohkoryhmään suorituskyvyn nimissä. ZFS voi sijoittaa kyseisen datan minne tahansa tallennuslaitetta (Beebe, Stacy ja Stuckey 2009, luku 2.2).

2.3 Btrfs

ZFS-tiedostojärjestelmään verrattuna Btrfs on osoittautunut yleisemmäksi. Se on otettu käyttöön monissa Linux-jakeluissa, kuten Fedora- (“Disk Configuration”, n.d.) ja SUSE Linux Enterprise-jakeluissa (“Overview of File Systems in Linux | SLES 12 SP4”, n.d.). Kuten ZFS, Btrfs tukee metadatan ja datan tarkistussummia.

Oletuksena tiedostojärjestelmä käyttää nopeaa CRC32C-tarkistussummaa, joka ei ole törmäyksenkestävä. Linux-kerneli version 5.5 jälkeen Btrfs tukee myös törmäysten kannalta turvallisempia XXHASH-, SHA-256- ja Blake2b-tarkistussummia (*Checksumming — BTRFS documentation,* n.d.). Btrfs käyttää B-puita levyn rakenteessa ja ZFS:n tapaan copy-on-write -metodia (Rodeh, Bacik ja Mason 2013, luku 1). ZFS:n lohko-osoittimien sijaan Btrfs tallentaa tarkistussummat erilliseen puurakenteeseen (Rodeh, Bacik ja Mason 2013, luku 2.1).

2.4 ReFS

ReFS on Windows-käyttöjärjestelmälle kehitetty Microsoftin omisteinen tiedostojärjestelmä, joka tukee datan ja metadatan tarkistussummia. Siinä missä Btrfs- ja ZFS-tiedostojärjestelmissä datan tarkistussummaus on oletusarvolta päällä, Microsoftin dokumentaation mukaan ReFS tarkistaa oletusarvoltaan pelkästään metadataa (Microsoft 2022). Btrfs ja ZFS luovat tarkistussummia koko osion datalle, mutta ReFS:ssä oikeellisuuden tarkistus voidaan rajoittaa yksittäisiin tiedostoihin tai polkuihin. Microsoftin dokumentaatiossa mainitaan, että datan oikeellisuuden tarkistaminen on hyvä pitää pois päältä suorituskykyherkissä järjestelmissä.

ReFS:n lähdekoodi on suljettua, joten virallista dokumentaatiota ei olla julkaistu siinä käytettävästä tarkistussumma-algoritmista. Tiedostojärjestelmän takaisinmallinnus on osoittanut, että levyllä little endian-muodossa luetut tarkistussummat ovat identtisiä ntfs-3g ajurin

käyttämään CRC64-algoritmiin (Nordvik ym. 2019). ReFS käsittelee tiedostojen muutokset allocate-on-write -tekniikalla, jossa muutokset kirjoitetaan uuteen paikkaan levyä. Kun tiedostoa kirjoitetaan, tarkistussumma lasketaan sen yhteydessä. Tällöin turvaudutaan mahdollisilta sähkökatkoksilta, jossa kirjoitettava data korruptoituu eikä alkuperäisestä datasta ole enää tarkistettavaa versiota (Sinofsky 2012).

3 Tarkistussummat

Tarkistussumman tehtävänä on luoda uniikki ja vakiopituinen tuloste annetusta datasta, riippumatta sen pituudesta. Tulosteen avulla nähdään, onko data muuttunut aiempaan tulokseen verrattuna. Luotu tarkistussumma ei voi olla täysin uniikki vakiopituuden takia, joka on otettava huomioon vanhemmissa algoritmeissa, kuten CRC:ssa. Tutkielman kannalta olennaisia tarkistussumma-algoritmeja ovat SHA-256, SHA-512 ja CRC. Tarkastellaan kyseisten algoritmien lisäksi ZFS:n ja Btrfs:n tukeman SHA-256-algoritmin eroja yleiseen MD5-algoritmiin.

3.1 SHA-512 ja SHA-256

SHA eli Secure Hash Algorithm on ryhmä kryptografisia tiivistefunktioita. Funktioiden toiminta kuvataan NIST:in julkaisemassa raportissa *Secure Hash Standard* (2015, luku 1). Niiden tehtävänä on luoda annetusta viestistä tiiviste yhdensuuntaisen ja iteratiivisen tiivistefunktion avulla. Algoritmin toiminta koostuu esikäsittelystä ja tiivisteen laskennasta. Esikäsittelyssä tiivisteen laskentaa varten asetetaan aloitusarvot ja viesti täydennetään ja jäsennetään m-bittisiin lohkoihin. Tämän jälkeen tiivistefunktio luo viestiaikataulun täydennetystä viestistä, jota käytetään funktioiden, vakioarvojen ja sanaoperaatioiden kanssa tiivistearvojen iteratiiviseen luomiseen. Viimeistä iteratiivisesti luotua tiivistearvoa käytetään tämän jälkeen tarkistussumman määrittämiseen.

SHA-1 on altis hyökkäyksille, joten SHA-algoritmien julkaisija NIST on kehottanut siirtymistä SHA-1:stä SHA-256:een, vaikka se on 2.2 kertaa hitaampi (Gueron, Johnson ja Walker 2011, luku 1). ZFS ja Btrfs tukevat SHA-2 -ryhmän algoritmeista molemmat SHA-256:tta. ZFS tukee tämän lisäksi myös SHA-512:ta (*Checksumming — BTRFS documentation,* n.d.) (“Checksums and Their Use in ZFS — OpenZFS documentation”, n.d.).

NIST:in mukaan SHA-algoritmit eroavat toisistaan merkittävimmin datalle tarjottavien tietoturvavahvuuksien osalta. Eroja nähdään myös algoritmeissa käytettävien lohkojen ja sanojen pituuksissa. Esimerkiksi SHA-256:n lohkon ja sanan koko on puolet SHA-512:n koosta (*Secure Hash Standard* 2015, luku 1).

3.2 CRC

CRC:n toiminnasta selittää Williams (1993), jonka mukaan algoritmin toiminta perustuu jakolaskuihin. Annettua dataa käsitellään yhtenä binäärisenä numerona, joka jaetaan toisella määrätyllä numerolla. Kyseisen laskutoimituksen jakojäännöstä käytetään algoritmin tarkistussummana (Williams 1993, luku 3).

Laskutoimituksen jakajana käytetään polynomia G pituudella W. Annetun datan loppuun lisätään W:n verran nollabittejä, jonka jälkeen dataa jaetaan polynomilla G niin kauan, kunnes saadaan tarkistussumma (Williams 1993, luku 6). Sopivan polynomin valinta on vaikeaa ja määrittää algoritmin vahvuuden. Tarkistussummalla huomataan esimerkiksi yhden bitin korruptio valitsemalla polynomi, jossa on vähintään kaksi ykkösbittiä. CRC16 käyttää esimerkiksi polynomia $(16, 15, 2, 0)$ (Williams 1993, luku 7). Mahdollisia polynomeja on keksitty useisiin eri tilainteisiin, joista muun muassa CRC32:ta käytetään PNG-tiedostoformaatissa (Laphroaig 2017) ja Ethernetissä (Williams 1993, luku 7), CRC32C:tä Ext4- ja Btrfs-tiedostojärjestelmissä ja CRC64:ää ReFS-tiedostojärjestelmässä.

Tiedetään, että polynomin valinta vaikuttaa siihen, kuinka suureen dataan CRC-algoritmia on mielekästä käyttää ja minkälaista korruptiota sillä voidaan huomata. Kyseistä rajoitusta voidaan täten käsitellä paremmin tiedostojärjestelmissä kuin tarkistussummatiedostoissa. Monet datan tarkistussummia tukevat tiedostojärjestelmät kuten Btrfs tallentavat tarkistussumman lohkosta koko tiedoston sijaan (“2. High Level Design — The Linux Kernel documentation”, n.d.). Tällöin CRC-algoritmille annettava data pysyy vakiopituisena tiedoston pituudesta riippumatta, mikä voi soveltua algoritmille tarkistussummatiedostoja paremmin.

3.3 MD5:n ja SHA-256:n vertailu

Tutkielmassa käsiteltävät tiedostojärjestelmät eivät tue MD5-tarkistussummaa. Manuaalisessa tiedoston tarkistamisessa MD5 on osoittautunut yleiseksi tarkistussumma-algoritmiksi. Liang ja Lai (2007) käsittelevät tutkielmassaan algoritmia, joka nopeuttaa MD5:lle toteutettavan törmäyshyökkäyksen laskemista. Tutkielmassa todetaan, että MD5:lle voidaan toteuttaa törmäyshyökkäys viidessä tunnissa 1,70 GHz Pentium 4-prosessorilla. Voidaan todeta, että MD5 ei täten sovellu tahallisen korruption tarkistamiseen.

Satunnaisen korruption tarkistamiseen MD5 soveltuu SHA-256:tta paremmin. Vaikka törmäyshyökkäykset ovat kyseisellä algoritmilla mahdollisia, MD5 on SHA-256:tta nopeampi. Rachmawatin et al. (2018) mukaan MD5 ja SHA-256 ovat asymptoottiselta suoritusajaltaan molemmat $\Theta(N)$ (Rachmawati, Tarigan ja Ginting 2018, luku 3.4), mutta käyntiajan suhteen MD5 osoittautuu nopeammaksi (Rachmawati, Tarigan ja Ginting 2018, luku 3.3).

CRC toimii kohtuullisen hyvin satunnaisen korruption tarkistamiseen, mutta ei ole MD5:n tapaan turvallinen tahallisen korruption sattuessa. Jos tarkistussumma-algoritmilta vaaditaan CRC:tä luotettavampaa virheen havaitsemista ilman tarvetta SHA-256:n kryptografiselle vahvuudelle, voidaan todeta MD5:n toimivan SHA-256:tta paremmin sen nopeamman suoritusajan perusteella.

4 Toteutukset eheyden tarkistamisessa

Tiedostojärjestelmät voivat tarkistaa dataa automaattisesti tiedoston luku- ja kirjoitusoperaatioiden yhteydessä, mutta kuten tutkielman alussa todettiin, suurin osa käytössä olevista tiedostojärjestelmistä eivät tätä toiminnallisuutta tue. Datan tarkistaminen on laajalti jäänyt tiedostojärjestelmien ulkopuolelle, joten tarkastellaan kyseisiä toteutuksia ja verrataan niiden toimintaa datan tarkistussummia tukeviin tiedostojärjestelmiin.

4.1 Virheenkorjauskoodit

On hyvä ottaa huomioon, että tallennusvälineet pitävät yllä virheenkorjauskoodeja jokaisella sektorilla, jolloin luettu tieto voidaan tarkistaa ja korjata. Tämä tapahtuu automaattisesti, ilman että käyttäjää kerrotaan kyseisistä operaatioista. Korjauksen jälkeen data saatetaan kirjoittaa uudelleen tai siirtää eri paikkaan (Shah ja Elerath 2005, luku 3.1).

Keskusmuistissa voidaan myös käyttää virheenkorjauskoodeja (Krishnan, Panigrahy ja Parthasarathy 2009). Ilman kyseistä virheenkorjauskoodeja tukevaa keskusmuistia, datalla on mahdollisuus korruptoitua jo ennen kuin se tallennetaan levylle.

4.2 Tarkistussummatiedostot

Tarkastellaan, miten tarkistussummatiedostoja voidaan luoda manuaalisesti ja millä tavoin tiedostojen tarkistaminen voidaan automatisoida. Unix-käyttöjärjestelmissä voidaan luoda ja tarkistaa tarkistussummatiedostoja GNU Core Utilities-paketissa olevien tarkistussummasovellusten avulla. Seuraava komento luo rekursiivisesti SHA-256-tarkistussummatiedoston `polku`-muuttujassa olevan kansion tiedostoista ja nimeää sen polun mukaan.

```
find "$polku" -type f -exec sha256sum {} ';' > "$polku".sha256
```

Tiedostot voidaan tarkistaa suorittamalla sovellus `-c` argumentilla ja tarkistussummatiedostolla.

```
sha256sum -c "$polku".sha256
```

Tämä toimii muuttumattoman varmuuskopion tarkistamiseen, muttei ole käytännöllinen järjestelmässä, jossa tiedostot voivat muuttua. Kyseinen ratkaisu toimii tällöin optisilla levyillä, joiden ISO 9660-tiedostojärjestelmä ei tue tarkistussummia. Toteutus on myös muihin tutkielmassa käsiteltäviin tarkistuksiin verrattuna parempi yksittäisten tiedostojen osalta, etenkin Internetin välityksellä jaettujen tiedostojen yhteydessä.

4.3 Laajennetut tiedostoattribuutit

Monet tiedostojärjestelmät, mukaan lukien ExtX, Ext4 ja NTFS, tukevat vapaamuotoisen metadatan tallentamista tiedostojen yhteyteen. Kyseistä tilaa voidaan hyödyntää datan oikeellisuuden tarkistamiseen.

Linuxille on saatavilla avoimen lähdekoodin sovellus `cshatag`, jolla voi tarkistaa tiedostojen oikeellisuuden laajennettuja tiedostoattribuutteja ja muokkausaikaleimaa hyödyntäen (rfjakob 2023). Sovellus on uudelleentoteutus Maxime Augierin Python-sovelluksesta `shatag`. Sovellus tallentaa dokumentaation mukaan tiedostojen muokkausaikaleiman ja tarkistussumman laajennettuihin tiedostoattribuutteihin, jos niitä ei löydetä. Tällöin korruptio voidaan huomata, jos muokkausaikaleima on pysynyt samana ja tarkistussumma on muuttunut edellisestä ajosta.

Sovellus voidaan automatisoida `cron`-ajastuspalvelulla suorittamalla se tietyin väliajoin tai aina käynnistyksen yhteydessä. `cshatag` tulostaa oletusarvoisesti kaikkien tiedostojen analyysit standardivirtaan ja korruptoituneet tiedostot standardivirheeseen. Tällöin standardivirheeseen tulostetut korruptioviestit voidaan liittää haluttuun tiedostoon, jonka käyttäjä voi väliajoin tarkistaa. Kyseinen tuloste voidaan myös lähettää käyttäjälle `notify-send`-sovelluksella, jolloin korruptioviesti ilmestyy näytölle ilmoituksena. Samankaltainen automatisaatio voidaan toteuttaa myös Task Scheduler-ohjelmistolla Windows-käyttöjärjestelmässä.

4.4 Laitekartoittajat

Linux-kernelin tarjoamaa laitekartoittajakehystä voidaan käyttää myös tarkistussummien varastointiin. Kehyksen avulla fyysiset lohkolaitteet voidaan kartoittaa korkeamman tason vir-

tuaalisiksi lohkolaitteiksi. Dm-integrity on kyseistä kehystä hyödyntävä komponentti, jonka avulla lohkolaitteen voidaan emuloida sisältävän sektoreiden yhteydessä ylimääräisiä eheyteen liittyviä tunnisteita (“dm-integrity — The Linux Kernel documentation”, n.d.). Journalointia tai bittikarttaa voidaan käyttää dm-integrity-komponentin kanssa, joista journalointi on hitaampaa mutta sähkökatkosten sattuessa varmempaa.

4.5 Tarkistussummatietokannat

Samoin kuin tarkistussummatiedoston luonnissa, tarkistussummat voidaan myös tallentaa yhteiseen tietokantaan. Laajennettuihin tiedostoattribuutteihin verrattuna toteutus eroaa pelkästään muokkausaikaleiman ja tarkistussumman varastoinnissa. Eräs tietokantaa hyödyntävä sovellus on Łukasz Langan kirjoittama `bitrot`.

Samoin kuin `cshatag`, `bitrot` huomaa korruption vertaamalla nykyistä ja tallennettua tarkistussummaa, jos muokkausaikaleimat täsmäävät. Tiedostoattribuuttien sijaan tarvittavat tiedot tallennetaan `.bitrot.db`-nimiseen SQLite 3-tietokantaan. Kuten dokumentaatiossa mainitaan, sovelluksen luoma tietokanta voidaan siirtää eri levylle (Langa 2023).

Rajoitteiden takia sovellus ei täysin korvaa `cshatag`-sovellusta. Koska tarkistussummia ei tallenneta tiedostojärjestelmän attribuutteihin, on tietokannassa pidettävä yllä tiedostonimiä. Jos tiedosto nimetään uudestaan tai siirretään eri kansioon, `cshatag` huomaa muokkausaikaleiman muutoksen ja tallentaa sen. Samasta tilanteesta `bitrot` ei kykene palautumaan, koska tietokanta osoittaa hävinneeseen tiedostoon. Manuaalisesti muutos voidaan korjata SQL-komennoilla, mutta automatisoituna sovellus ei kykene tarkistamaan, onko uudelleennimetty tiedosto vielä virheetön. Tällöin sovellus olettaa tiedoston olevan uusi.

4.6 Ohjelmistot

Tarkistussummien tallentaminen voi nostaa uuden ongelman. Tiedoston tarkistamiseen voi luottaa vain, jos itse tarkistussumma on luotettava, eikä ole korruptoitunut tai tahallisen muutoksen myötä vaarantunut. Tarkistussumman väärentäminen on mahdollista sekä tarkistussummatietokannoissa että tiedostojärjestelmissä.

Tämän suhteen voidaan hyödyntää IntegrityCatalogia, jossa itse eheysmetadataa pidetään ensiluokkaisena objektina, jonka eheys tarkistetaan säännöllisesti. Chondros ja Roussopoulos (2018) käsittelevät tutkielmassaan IntegrityCatalogin toimintaa ja nostavat esille yksinkertaisiin tarkistussummatiedostoihin liittyviä turvallisuusriskejä.

Tarkistussummatiedostot ovat luotettavia vain, jos niihin on kirjoitusoikeus pelkästään luotetuilla käyttäjillä. Muutoin tarkistussummatiedostojen poistaminen tai muokkaus on hyökkääjälle yksinkertaista. Chondros ja Roussopoulos ottavat myös huomioon tarkistussummien pysyvyyden. Jos tarkistussummat on tallennettu kolmannen osapuolen palveluihin kuten Usenettiin, datan tarkistaminen ei tule olemaan tulevaisuudenkestävää.

Chondros ja Roussopoulos kuvaavat tutkielmassa IntegrityCatalogin toimintaa tarkemmin. Se pitää tietovarastossa olevista objekteista yllä eheystietoja ja tarkistaa objektit säännöllisesti. Tämän lisäksi järjestelmä pitää yllä itse eheystietoihin liittyvää muutoshistoriaa. Jos IntegrityCatalogin tietoihin tulee odottamattomia muutoksia tai korruptiota, voidaan järjestelmä palauttaa aiempaan tilaan. IntegrityCatalogia ei myöskään tarvitse jättää yhden ainoan luottamuspisteen varaan, vaan järjestelmän tiedot voidaan jakaa useisiin solmuihin säännöllisesti (Chondros ja Roussopoulos 2018, luku 3).

Tiedostojen tarkistamisen puitteissa tarkistussummatietokannat ja tiedostojärjestelmien toteuttama datan tarkistaminen toimivat useassa tilanteessa tarpeeksi hyvin, mutta monen käyttäjän järjestelmässä voi olla mielekästä käyttää IntegrityCatalogin tapaista ohjelmistoa. Kyseisen ohjelmiston tapaisia toimintoja ei ole saatavilla tutkielmassa käsiteltävissä tiedostojärjestelmissä. Voidaan todeta, että tiedostojärjestelmä eroaa perimmäisessä tehtävässään liikaa, jotta siihen voitaisiin mielekkäästi sisällyttää IntegrityCatalogin tapaista toiminnallisuutta. Jotkin tarkistamisen toteutukset on täten hyvä pitää erillisinä ohjelmistoinaan.

4.7 Erot tiedostojärjestelmiin verrattuna

Tallennusvälineet eivät ole täydellisiä, jonka takia vähänkään tärkeästä datasta tulisi aina olla tarkistussummia muodossa tai toisessa. Yleisellä tasolla tiedostojärjestelmien tarkistussummaus toimii käyttötapauksesta riippumatta turvaverkkona. Niiden tuomat hyödyt ovat mahdollisia suorituskykyyn liittyviä haittoja huomattavasti suuremmat, erityisesti loppukäyttäjän

näkökulmasta.

Suurin datan eheyttä tarkistavan tiedostojärjestelmän etu on sen tuoma automatisaatio. Käyttäjän ei itse tarvitse konfiguroida datan oikeellisuuden tarkistamista muuten kuin valitsemalla sitä tukeva tiedostojärjestelmä. Tarkistaminen on tämän kannalta luonnollisinta abstrahoida tiedostojärjestelmän kutsujen alle. Muiden toteutuksien tukemiin tarkistussummaalgoritmeihin verrattuna erot ovat minimaalisia lopputulos huomioon ottaen.

Tiedostojärjestelmä ei luonnostaan ole ainoa ratkaisu. Sen luomat tarkistussummat jäävät siihen osioon tai tallennuslaitteeseen, jossa sitä käytetään. Jos dataa on siirrettävä paikasta toiseen, on tarkistussummat laskettava uudestaan tai kalastettava tiedostojärjestelmästä. Tällöin erilliset tarkistussummatiedostot ja -tietokannat toimivat tehtävässään paremmin.

Tiedostojen oikeellisuuden tarkistamisessa voidaan käyttää yksittäisiä tarkistussummatiedostoja, mutta tiedostoa avatessa tämä johtaisi yhteen ylimääräiseen erillisen tiedoston lukuoperaatioon (Sivathanu, Wright ja Zadok, n.d., luku 4.2). Tiedostojärjestelmä voi tallentaa tarkistussummat tiedoston metadatan yhteyteen, jolloin ylimääräisiä tiedostoja ei tarvitse tarkistaessa avata.

5 Yhteenveto

Tutkielmassa käsiteltiin datan tarkistussummia tukevia tiedostojärjestelmiä ja vaihtoehtoisia toteutuksia datan tarkistamiseen. Tiedostojärjestelmien suurimmaksi eduksi todettiin niiden kyky datan tarkistamisen automatisoinnille. Toiseksi eduksi todettiin CRC-tarkistussummien suhteen tiedostojärjestelmien kyky tarkistaa tiedostot lohkoittain.

Tutkielmassa otettiin myös huomioon vaihtoehtoisien toteutuksien etuja, joista tärkeimmäksi osoittautui niiden siirrettävyys tiedostojärjestelmiin nähden. Käytiin myös läpi `cshatag`-sovellus, jolla voidaan toteuttaa datan tarkistussummat tiedostojärjestelmissä, jotka tukevat tarkistussummien sijaan laajennettuja tiedostoattribuutteja. Vaihtoehtoiset toteutukset tukevat myös useampia tarkistussumma-algoritmeja, jolloin ne voivat toimia järjestelmän vaatimuksista riippuen tiedostojärjestelmiä paremmin.

Lähteet

"2. High Level Design — The Linux Kernel documentation". n.d. Viitattu 11. huhtikuuta 2023. https://www.kernel.org/doc/html/latest/filesystems/ext4/overview.html#checksums.

Beebe, Nicole Lang, Sonia D. Stacy ja Dane Stuckey. 2009. "Digital forensic implications of ZFS". *Digital Investigation,* The Proceedings of the Ninth Annual DFRWS Conference, 6 (1. syyskuuta 2009): S99–S107. ISSN: 1742-2876, viitattu 26. tammikuuta 2023. https://doi.org/10.1016/j.diin.2009.06.006. https://www.sciencedirect.com/science/article/pii/S1742287609000449.

Checksumming — BTRFS documentation. n.d. Viitattu 10. huhtikuuta 2023. https://btrfs.readthedocs.io/en/latest/Checksumming.html.

"Checksums and Their Use in ZFS — OpenZFS documentation". n.d. Viitattu 11. huhtikuuta 2023. https://openzfs.github.io/openzfs-docs/Basic%20Concepts/Checksums.html.

Chondros, Nikos ja Mema Roussopoulos. 2018. "Developing IntegrityCatalog, a software system for managing integrity-related metadata in digital repositories". *Software: Practice and Experience* 48 (1): 45–64. ISSN: 1097-024X, viitattu 26. tammikuuta 2023. https://doi.org/10.1002/spe.2515. https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2515.

"Disk Configuration". Fedora Docs. n.d. Viitattu 10. huhtikuuta 2023. https://docs.fedoraproject.org/en-US/workstation-docs/disk-config/.

"dm-integrity — The Linux Kernel documentation". n.d. Viitattu 10. huhtikuuta 2023. https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-integrity.html.

Fairbanks, Kevin D. 2012. "An analysis of Ext4 for digital forensics". *Digital Investigation,* The Proceedings of the Twelfth Annual DFRWS Conference, 9 (1. elokuuta 2012): S118–S130. ISSN: 1742-2876, viitattu 10. huhtikuuta 2023. https://doi.org/10.1016/j.diin.2012.05.010. https://www.sciencedirect.com/science/article/pii/S1742287612000357.

Gueron, Shay, Simon Johnson ja Jesse Walker. 2011. "SHA-512/256". *Information Technology: New Generations,* 354–358.

Krishnan, Sriram C., Rina Panigrahy ja Sunil Parthasarathy. 2009. “Error-Correcting Codes for Ternary Content Addressable Memories”. Conference Name: IEEE Transactions on Computers, *IEEE Transactions on Computers* 58, numero 2 (helmikuu): 275–279. ISSN: 1557-9956. https://doi.org/10.1109/TC.2008.179.

Langa, Łukasz. 2023. *bitrot.* Original-date: 2013-01-16T10:25:46Z. 31. maaliskuuta 2023. Viitattu 4. huhtikuuta 2023. https://github.com/ambv/bitrot.

Laphroaig, Manul. 2017. *PoC or GTFO.* Google-Books-ID: lvMxDwAAQBAJ. No Starch Press, 31. lokakuuta 2017. ISBN: 978-1-59327-898-4.

Lee, Seonho, Jungheum Park, Hyunuk Hwang, Seungyoung Lee, Sangjin Lee ja Doowon Jeong. 2021. “Forensic analysis of ReFS journaling”. *Forensic Science International: Digital Investigation* 38 (1. lokakuuta 2021): 301136. ISSN: 2666-2817, viitattu 9. helmikuuta 2023. https://doi.org/10.1016/j.fsidi.2021.301136. https://www.sciencedirect.com/science/article/pii/S2666281721000342.

Liang, Jie ja Xue-Jia Lai. 2007. “Improved Collision Attack on Hash Function MD5”. *Journal of Computer Science and Technology* 22, numero 1 (1. tammikuuta 2007): 79–87. ISSN: 1860-4749, viitattu 14. maaliskuuta 2023. https://doi.org/10.1007/s11390-007-9010-1. https://doi.org/10.1007/s11390-007-9010-1.

Microsoft. 2022. “ReFS integrity streams”, 29. maaliskuuta 2022. Viitattu 10. huhtikuuta 2023. https://learn.microsoft.com/en-us/windows-server/storage/refs/integrity-streams.

Nordvik, Rune, Henry Georges, Fergus Toolan ja Stefan Axelsson. 2019. “Reverse engineering of ReFS”. *Digital Investigation* 30 (1. syyskuuta 2019): 127–147. ISSN: 1742-2876, viitattu 23. helmikuuta 2023. https://doi.org/10.1016/j.diin.2019.07.004. https://www.sciencedirect.com/science/article/pii/S1742287619301252.

“Overview of File Systems in Linux | SLES 12 SP4”. n.d. Viitattu 10. huhtikuuta 2023. https://documentation.suse.com/sles/12-SP4/html/SLES-all/cha-filesystems.html.

Rachmawati, D., J. T. Tarigan ja A. B. C. Ginting. 2018. “A comparative study of Message Digest 5(MD5) and SHA256 algorithm”. Place: Bristol, United Kingdom Publisher: IOP Publishing, *Journal of Physics: Conference Series* 978, numero 1 (maaliskuu). ISSN: 17426588, viitattu 9. helmikuuta 2023. https://doi.org/10.1088/1742-6596/978/1/012116. https://www.proquest.com/docview/2572080803/abstract/5F754DBD78B54A3FPQ/1.

rfjakob. 2023. *rfjakob/cshatag*. Original-date: 2012-05-12T14:01:40Z. 2. huhtikuuta 2023. Viitattu 4. huhtikuuta 2023. https://github.com/rfjakob/cshatag.

Rodeh, Ohad, Josef Bacik ja Chris Mason. 2013. “BTRFS: The Linux B-Tree Filesystem”. *ACM Transactions on Storage* 9, numero 3 (1. elokuuta 2013): 9:1–9:32. ISSN: 1553-3077, viitattu 26. tammikuuta 2023. https://doi.org/10.1145/2501620.2501623. https://dl-acm-org.ezproxy.jyu.fi/doi/pdf/10.1145/2501620.2501623.

Secure Hash Standard. 2015. NIST FIPS 180-4. National Institute of Standards ja Technology, heinäkuu. Viitattu 30. huhtikuuta 2023. https://doi.org/10.6028/NIST.FIPS.180-4. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Shah, S. ja J.G. Elerath. 2005. “Reliability analysis of disk drive failure mechanisms”. Teoksessa *Annual Reliability and Maintainability Symposium, 2005. Proceedings.* 226–231. ISSN: 0149-144X, Annual Reliability and Maintainability Symposium, 2005. Proceedings. Tammikuu. https://doi.org/10.1109/RAMS.2005.1408366.

Sinofsky, Steven. 2012. “Building the next generation file system for Windows: ReFS”, 16. tammikuuta 2012. Viitattu 10. huhtikuuta 2023. https://learn.microsoft.com/en-us/archive/blogs/b8/building-the-next-generation-file-system-for-windows-refs.

Sivathanu, Gopalan, Charles P Wright ja Erez Zadok. n.d. “Enhancing File System Integrity Through Checksums”.

Wani, Mohamad Ahtisham ja Wasim Ahmad Bhat. 2018. “Dataset for forensic analysis of B-tree file system”. *Data in Brief* 18 (1. kesäkuuta 2018): 2013–2018. ISSN: 2352-3409, viitattu 26. tammikuuta 2023. https://doi.org/10.1016/j.dib.2018.04.100. https://www.sciencedirect.com/science/article/pii/S2352340918304530.

Williams, Ross N. 1993. *A Painless Guide to CRC Error Detection Algorithms,* 19. elokuuta 1993. Viitattu 30. huhtikuuta 2023. http://www.ross.net/crc/download/crc_v3.txt.