

Antti Hakkarainen

**LÄNSIMAISIA VALTIOTASON
SUOSITUKSIA OHJELMISTOKEHITYKSEN
MUISTITURVALLISUUDESTA**

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Joonas Multanen
Toukokuu 2024

TIIVISTELMÄ

Antti Hakkarainen: Länsimaisia valtiotason suosituksia ohjelmistokehityksen muistiturvallisuudesta

Kandidaatintyö

Tampereen yliopisto

Luonnontieteiden kandidaatti, tietojenkäsittelytiede

Toukokuu 2024

C- ja C++ -ohjelmointikielien tarjoavat ohjelmoijalle mahdollisuuden manuaaliseen muistinhallintaan ja monissa tavanomaisissa tilanteissa myös edellyttävät sitä. Työssä käydään läpi C- ja C++-kielien muistinkäsittelyä yleistasolla, muistinkäsittelyvirheistä aiheutuvat neljä yleisintä tietoturva-vaavoittuvuustyyppiä ja kolme tuoretta länsimaista raporttia ohjelmistokehityksen muistiturvallisuuteen liittyen. Työn tavoitteena on selvittää, minkälaisia suosituksia muistiturvalliseen ohjelmistokehitykseen liittyen on viime aikoina julkaistu.

Mitre Corporationin ylläpitämän haavoittuvuustietokannan mukaan suurin osa ohjelmistojen tietoturva-vaavoittuvuuksista aiheutuu C- ja C++ -ohjelmointikielien muistihaavoittuvuuksista. Yleisimmät haavoittuvuustyypit löydettiin selaamalla Mitren Internet-sivujen tietokantaa, ja valitsemalla vuosien 2019–2023 CWE Top25-listalta C:lle ja C++:lle spesifit muistinkäsittelyvirheisiin liittyvät haavoittuvuustyypit. Raportit valittiin tarkasteluun, koska ne ovat tuoreita ja sisältö vaikutti sopivan tutkimuskysymykseen. Raportit luettiin ja muistiturvallisen ohjelmistokehityksen suosituksista kirjoitettiin lyhyt yhteenveto.

Raportit suosittavat toteuttamaan jo olemassa olevien muistiturvattomilla ohjelmointikielillä toteutettujen ohjelmistojen uudet osat muistiturvallisilla ohjelmointikielillä, ja aloittamalla uudet ohjelmistoprojektit muistiturvallisilla ohjelmointikielillä. Tutkimuskysymykseen sopivaa kirjallisuutta etsittäessä havaittiin, että kotimaisia suosituksia muistiturvalliseen ohjelmointiin ei tällä hetkellä ole olemassa, eikä niitä ole ainakaan lähiaikoina tulossa. Sivujuonteena tarkastettiin myös suomalaisten korkeakoulujen opinnäytetöiden julkaisuarkistot ja etsittiin niistä muistiturvallisuutta käsitteleviä opinnäytetöitä, joita löytyi vain muutama.

Turvallinen ohjelmointi C- ja C++ -ohjelmointikielillä vaikuttaa lukujen valossa haastavalta. Suositusten noudattaminen saattaa vähentää muistinkäsittelyvirheiden aiheuttamien tietoturvaongelmien määrää ja niistä aiheutuvia kustannuksia. Lopuksi pohditaan, mihin suuntaan C++ mahdollisesti voi kehittyä, pitäisikö muistiturvallisuutta tutkia Suomessa enemmän ja onko korkeakoulujen syytä tarkastella ohjelmointikurssien muistiturvattomien ohjelmointikielien käytön järkevyyttä.

Avainsanat: muistiturvallisuus, muistiturvallinen, ohjelmistokehitys, c, c++, ohjelmointi

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

SISÄLLYSLUETTELO

1.	Johdanto	1
2.	Ongelmat C- ja C++ -kielten muistinkäytössä	3
2.1	Ajonaikaisista muistinkäsittelyvirheistä.	3
2.2	Erilaisia ohjelmointikielten muistinkäsittelyparadigmoja	5
3.	Pääasialliset muistinkäsittelyvirheiden lähteet C/C++ -ohjelmistoissa	6
3.1	CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer	7
3.2	CWE-125 - Out-of-bounds Read	8
3.3	CWE-416 - Use After Free	8
3.4	CWE-787 - Out-of-bounds Write	9
4.	Valtiotason suositukset muistiturvallisesta ohjelmistokehityksestä	10
4.1	Secure by Design	10
4.2	Consumer Reports	11
4.3	Valkoinen Talo	11
4.4	Kotimaiset suositukset ja opinnäytteet	12
5.	Keskustelu	14
5.1	C++:n kehittäjien reagointi turvallisuuskritiikkiin.	14
5.2	Tulosten merkitys korkeakouluopetuksessa	15
6.	Yhteenveto	16
	Lähteet	17
	Liite A: Ohjelmakoodiesimerkit	20

LYHENTEET JA MERKINNÄT

ASLR	Address Space Layout Randomization
CFI	Control-Flow Integrity
CISA	Cybersecurity and Infrastructure Security Agency. Yhdysvaltain kyberturvallisuusvirasto.
CVE	Common Vulnerabilities and Exposures. Tietoturva-aukkojen raportointi- ja dokumentointiprojekti ja -järjestelmä.
CWE	Common Weakness Enumeration. Tietoturvaavaoittuvuuksien luokittelujärjestelmä.
FBI	Federal Bureau of Investigation. Yhdysvaltain liittovaltion poliisi.
HDD	Hard Disk Drive. Kiintolevy.
NSA	The National Security Agency. Yhdysvaltain kansallinen turvallisuusvirasto.
RAM	Random Access Memory. Hajasaantimuisti.
ROM	Read Only Memory. Lukumuisti.
SSD	Solid-State Drive, puolijohdeasema.

1. JOHDANTO

Edeltävän vuoden aikana useat eri ulkomaiset viranomaislähteet [4, 5, 12, 31] ovat tehneet julkaisuja, joissa käsitellään tietoturvaongelmia ja niistä aiheutuvia alati kasvavia kustannuksia. Kyberrikollisuuden kustannuksista on hyvin vaikeaa löytää luotettavalta vaikuttavia arvioita. Erään lähteen mukaan vuotuiset kulut liikkuvat miljardeissa euroissa [15]. IBM:n mukaan [16] yksi tietoturvaloukkaus maksaa nykyään keskimäärin 4,45 miljollista dollaria kappaleelta, nousun ollessa 15,3 % vuoden 2020 tasosta. Viranomaisjulkaisut esittävät lukuisia syitä, joista kyberrikollisten hyödyntämät tietoturvahaavoittuvuudet tavanomaisesti johtuvat. Yhtenä toistuvana teemana esiintyy muistiturvattomien ohjelmointikielien käyttö.

Vuonna 2012 Victor van der Veen ym. osoittivat [33], että yli 60 % suosituista haavoittuvuustyökalukokoelmissa hyödynnetyistä tietoturvahaavoittuvuuksista olivat muistihaa-voittuvuuksien syytä. Vastaaviin lukuihin on päässyt Microsoft, joka vuonna 2019 sanoi [3], että n. 70 % löydettyistä haavoittuvuuksista heidän ohjelmissaan ovat muistihaa-voittuvuuksia. Bhurtel ja Rawat vuonna 2023 julkaistussa työssään [2] raapivat (engl. scraping) Mitre Corporationin haavoittuvuustietokantaa ja päätyivät vastaaviin tuloksiin.

Muistihaa-voittuvuudet ovat siis merkittävä tietoturvahaavoittuvuuksien lähde, ja muistihaa-voittuvuuksien suhteellinen osuus kaikista löydettyistä haavoittuvuuksista on pysynyt suhteellisen vakiona pitkiä aikoja. Tietokoneen muistin virheellisestä käytöstä aiheutuvat tietoturvaongelmat yksilöityvät pitkälti C/C++ -ohjelmointikieliin yleisyyden ja kielten perustavanlaatuisen ominaisuuksien vuoksi [33]. Työssä käsitellään molempia kieliä yhtä aikaa, sillä alun perin C++ oli Bjarne Stroustrupin C:n päälle luoma olio-ohjelmointilaajennos. Nykyään kielet ovat jonkin verran erkaantuneet toisistaan, eikä kaikki C-syntaksi ole enää validia C++:aa. C++:lla on silti mahdollista tehdä samoja ohjelmointivirheitä kuin C:llä, joten työssä käsitellään molempia kieliä yhdessä.

Kotimaisia, suomeksi tai englanniksi kirjoitettuja, muistiturvallisuutta yleisellä tasolla käsitteleviä opinnäytetöitä tai julkaisuja ei ole juurikaan tehty. Koska aiheesta on viime aikoina käyty maailmalla kiivasta keskustelua, katsaus muistiturvallisen ohjelmistokehityksen nykysuosituksen tilaan vaikuttaa olevan paikallaan. Työ ei siis varsinaisesti liity jo olemassa olevaan suomalaisen tieteellisen yhteisön käymään keskusteluun, vaan sen tarkoituksena on pikemminkin kysyä, olisiko aiheesta tarpeellista keskustella enemmän. Työssä perehdy-

tään erityisesti muistiturvallisuuksiä käsitteleviin länsimaisten viranomaisten julkaisemiin englanninkielisiin valtiotason suosituksiin.

Ulkomaisten viranomaissuosituksen mukaan [4, 5, 12, 31] ohjelmistoalan toimijoiden riippuvuutta C/C++ -ohjelmointikielistä on hyvä vähentää mahdollisuuksien mukaan. Kielessä on tuotettu vuosikymmenten aikana valtava määrä ohjelmistoja, joten mahdollinen muutos nykytilanteeseen tapahtunee hyvin hitaasti. Virheellisen muistinkäytön aiheuttamista tietoturvaongelmista saanee lukea ja kokea kukin vielä siis pitkään, sillä lukemattomien syiden, kuten inertian, muutosvastarinnan, osaamisen puutteen ja keskeneräisten vaihtoehtoisten ohjelmointityökalujen vuoksi muistiongelmat eivät ole todennäköisesti katoamassa maan päältä pelkkien viranomaissuosituksen vuoksi.

Johdannon jälkeen 2. luvussa käydään lyhyesti läpi suosituksen mukaisia turvallisen ohjelmistotuotannon perusteita, ja esitetään mistä muistinhallinnassa ja muistiturvallisudessa on kyse tietokoneohjelmia tehdessä. 3. luvussa esitetään teoriapohjan rakentamiseksi Mitre Corporationin ylläpitämän tietoturvaavaoittuvuustyyppitietovaraston C/C++ -kieliin yksilöityvät neljä yleisintä haavoittuvuustyyppiä. 4. luvussa katsastetaan mm. NSA:n, Consumer Reportsin, CISA:n ja FBI:n julkaisuja, Euroopan Unionin tulevaa kyberturvallisuuteen liittyvää säädösehdotusta, ja kotimaisten tahojen kuten kyberturvallisuuskeskuksen ja Valtiovarainministeriön julkaisuja. Lukujen 2 ja 3 aihepiirien käsittely työssä auttaa ymmärtämään oleellisia taustalla olevia syitä, miksi luvussa 4 käsitellyt suositukset on laadittu. Tämä auttaa kokonaiskuvan hahmottamisessa. Lopuksi keskustellaan tuloksista, haastetaan niitä vasta-argumentein, ja koostetaan löydöksiä yhteen.

Työtä lukiessa kannattaa tiedostaa, että viranomaisjulkaisut eivät tyypillisesti ole tieteellisen metodin avulla tuotettuja vertaisarvioituja töitä, ja globaalin tarkastelun poisjättäminen mm. Afrikan, Etelä-Amerikan ja Aasian maiden osalta saattaa antaa yksipuolisen kuvan vallitsevasta tilanteesta yleisellä tasolla. Laajemman mittakaavan globaalin työn tekeminen vaatisi todennäköisesti selvästi nykyistä enemmän tutkimusta.

2. ONGELMAT C- JA C++ -KIELIEN MUISTINKÄYTÖSSÄ

On olemassa ainakin kaksi eri todellisuutta: reaali maailma ja ideaali maailma. Ideaali maailmassa esimerkiksi ohjelmistokehitys tehtäisiin aina parhaiden saatavilla olevien työkalujen ja tiedon avulla, tinkimättä mistään, rajattomalla budjetilla, työvoimalla ja aikataululla. Reaali maailmassa ohjelmistokehityksessä joudutaan tinkimään useista eri muuttujista, syistä, jotka voivat liittyä esimerkiksi yrityksen kilpailukykyyn markkinoilla, käytettävissä olevien tiimien osaamiseen tai yliopistotutkijan käytettävissä oleviin resursseihin.

Kompromissien tekemisen tavoitteena voi esimerkiksi olla, että lopputulos on riittävän hyvä käyttötarkoitukseensa, tuote kestää käytössä suunnitellun elinkaaren ajan, ja ylläpito-kustannukset pysyvät siedettävällä tasolla [1]. Toisin sanoen turvallisuus ohjelmistoalalla maksaa rahaa ja aikaa. Esimerkiksi C:n ja C++:n puutteet tiedostetaan kentällä C++:n kehittäjä myöten [30], mutta reaali maailman syiden vuoksi kieliä käytetään ohjelmistokehityksessä laajasti.

Ohjelmointikielien voidaan jakaa sekä matalan että korkean tason ohjelmointikieliin. Matalan tason Assembly koostuu pääasiassa yksittäisiä konekäskyjä 1:1 suhteessa vastaavista symbolisista käskyistä. Assemblyn symbolisen notaation avulla ihmisen on huomattavasti helpompaa luoda ohjelmia, kuin suoraan binäärikoodilla tietokoneen kanssa kommunikoida. Assemblyn kirjoittaminen vaatii silti ihmistä ajattelemaan kuin tietokone. Helpotukseksi on luotu korkean tason ohjelmointikieliä kuten C ja C++, jotka abstrahoiivat tietokoneen käskykannan ohjelmakoodin kääntäjän huoleksi [28].

C-kieliin on jätetty ohjelmoijalle mahdollisuus hallita tietokoneen muistia matalalla tasolla. Muistinkäytön virheet ohjelman ajon aikana altistavat ohjelman tietoturvaongelmille. Seuraavaksi esitetään hieman C-kielten muistinkäyttöä, joka valaisee, miksi kielillä on mahdollista tehdä muistinkäsittelyvirheitä.

2.1 Ajonaikaisista muistinkäsittelyvirheistä

Automaattisessa tietojenkäsittelyssä tarvitaan muistia joka vaiheessa. Muistia on käytössä useita eri tyyppisiä, kuten välimuistit (Cache), hajasaantimuisti (RAM), lukumuisti (ROM), massamuistit kuten kiintolevyt (HDD), puolijohdeasemat (SSD), nauhat ym. Erilaisiin

muisteihin tallennetaan sekä ohjelmien ajonaikaisia tietoja, että pitkäaikaisessa säilytyksessä olevia tietoja [28].

Tietokoneohjelman suorituksen aikana ohjelma käyttää pääasiassa hajasaantimuistia ohjelman tilan tallentamiseen. Käyttöjärjestelmä ja rauta abstrahoiivat ohjelmoijalta sen, min-kälaisessa muistissa - välimuistissa, hajasaantimuistissa, kiintolevyllä - ohjelmiston ajonaikaiset osat ovat. Tavanomaisesti ohjelmalle riittää, että se pyytää käyttöönsä N määrän muistia käyttöjärjestelmältä. Koska ohjelmat usein tarvitsevat ennalta määrittelemättömän määrän muistia käyttöönsä ajon aikana, ohjelmat pyytävät käyttöjärjestelmältä muistia käyttöönsä *dynaamisesti*. Vastapainona tälle on *staattinen* muistinkäyttö, jolloin ohjelman tarvitseman muistin määrä tiedetään tarkasti jo ohjelmakoodin kääntämisen yhteydessä [28].

C-kielillä dynaamista muistia pyydetään käyttöön *malloc()*-komennolla[8], ja vapautetaan *free()*-komennolla[7]. *malloc()*-kutsulle annetaan parametriksi määrä, kuinka paljon muistia halutaan käyttöön. Kutsu palauttaa osoittimen muistialueeseen, joka pitää muistaa vapauttaa muiden ohjelmien käyttöön *free()*-kutsulla, kun ohjelma ei tarvitse enää itse kyseistä muistimäärää. Parametrina annetaan tällöin osoitin, joka kertoo, mikä muistialue halutaan vapauttaa. Osoittimen avulla[6] ohjelmoija voi käskyttää ohjelmaa tallentamaan ajonaikaisia tietoja, ja ohjelma siten tietää, minne sen kuuluu tiedot tallentaa, ja mistä ne myöhemmin löytyvät. Tämänkaltainen ohjelmoijan osaamiseen ja huolellisuuteen tukeutuva manuaalinen muistinhallinta altistaa ohjelmistot monenlaisille muistinkäsittelyvirheille.

Aikanaan tietoturvan huomiointi ei ollut merkittävä tekijä, sillä ensimmäiset tietoturvahyökkäykset ja haittaohjelmat luotiin vasta 80-luvulla. Ehkä siksi ohjelmoijille annettiin C:ssä mahdollisuus käsitellä tietokoneen muistia matalalla tasolla mm. muistiosoittimien avulla. Koska virheetön tietokoneen muistinkäsittely on huomattavan vaikeaa toteuttaa käytännössä, tämän C:n arkkitehtuurisen päätöksen seurauksena suurin osa C:llä kirjoitettujen tietokoneohjelmien tietoturvaavoittuvuuksista ovatkin muistinkäytön virhetilanteiden hyödyntämiä.

Virhetilanteet ovat vakava ongelma siksi, että ohjelman ajonaikainen virheellinen muistinkäyttö tarjoaa tyypillisesti hyökkääjälle mahdollisuuden tehdä laitteistolla haluamiaan asioita. 2.11.1988 aktivoitu Morriksen mato[17] on yksi ensimmäisistä matotyypisistä haittaohjelmista, minkä seurauksena tuhannet Internetiin kytketyt tietokoneet saastuivat - noin 10 % silloisesta Internetiin kytketystä konekannasta! - jotka muuttuivat madon toimintalogiikan vuoksi tilapäisesti käyttökelvottomiksi. Mato hyödynsi mm. puskurin ylivuotoa *finger*-verkkopalvelussa[14], joka oli toteutettu C-ohjelmointikielellä.

2.2 Erilaisia ohjelmointikielien muistinkäsittelyparadigmoja

Tietokoneen muistia voi varata käyttöön monella eri tapaa. C ja C++ -kielissä muistia voi varata dynaamisesti keosta itse. Jotkut ohjelmointikielet kuten Java, C#, ja Python käyttävät roskienkeräystä (Garbage collection). Roskienkeräysparadigmassa ohjelma automaattisesti vapauttaa muistia, kun se päättelee, että mikään ohjelman suorituksen aikainen osa ei enää viittaa johonkin ohjelman aikaisemmin varaamaan muistialueeseen. Tämä hidastaa C:hen verrattuna ohjelmien suorituskykyä, ja voi aiheuttaa hetkellisiä ylimääräisiä hidastumisia roskienkeräysprosessin aktivoituessa. Rust-ohjelmointikieli on taas suunniteltu niin, että sen kääntäjä pystyy ohjelmakoodista päästelemään, kuinka paljon muistia tarvitaan ajon aikana. Käytetystä ohjelmointikielestä riippuen ohjelmoijan ei siis välttämättä tarvitse hallita ohjelman muistinkäyttöä itse.

Rustia on usein ehdotettu C:n korvaajaksi. Toisaalta sen kanssa on tyypillinen muna-kana-ongelma: Koska monien mielestä ohjelmointikieli ja sitä ympäröivät ohjelmistotyökalut eivät ole riittävän kypsiä korvaamaan C-kieltä, ja C:llä on toteutettu niin suuret määrät ohjelmistoja, ei katsota järkeväksi siirtyä Rustiin. On mahdollista, että Rustin työkalut kehittyisivät kaupallisten toimijoiden edellyttämälle taholle, jos sitä käytettäisiin runsaammin. Koska urauurtavan pioneerin toimintaan sisältyy hankalasti ennustettavia liiketaloudellisia riskejä, ei “tuttuja ja turvallisia” C:tä ja C++:aa käyttämällä joudu niitä välttämättä kohtaamaan.

C++ -kielen kehittäjä Bjarne Stroustrup viime syksyn CPPConin puheessaan esittää, [30]. että on hyvinkin mahdollista, että nykyaikaisesta C++:sta voidaan jalostaa kieli, joka pystyy oleellisesti ratkaisemaan nykyisiä muistinhallintaan liittyviä ongelmia. Vastakohtana hänelle suurista ohjelmistoalan toimijoista mm. Google on myöntänyt, että C++:n kehittäminen muistiturvalliseksi kieleksi ei ole realistinen tavoite [29]. Samalla kuitenkin Google toteaa, että yksittäisten yritysten satojen miljoonien ja globaalisti miljardien koodirivien uudelleenkirjoittaminen ei myöskään ole realistinen tavoite.

Ohjelmointikielen valinta ei ole yksinkertainen tehtävä. Se on kompromissi ainakin käytävissä olevien ohjelmistotyökalujen, rautatuen, suorituskykyvaatimusten, saatavilla olevien ohjelmoijien osaamisen, standardien ja turvallisuuden välillä. Myös ohjelmointikielien kehittäjien lupaukset kielen kehityksen jatkamisesta saattavat osaltaan saada toimijoita pitäytymään perinteisissä kielissä.

3. PÄÄASIAALLISET MUISTINKÄSITTELYVIRHEIDEN LÄHTEET C/C++ -OHJELMISTOISSA

Mitre Corporation on voittoa tavoittelematon yritys, jonka toimintaa mm. Yhdysvaltain kotimaan turvallisuusvirasto rahoittaa. Mitre ylläpitää maailmanlaajuisesti tunnettua CVE-järjestelmää (Common Vulnerabilities and Exposures), ja CWE-luokittelua (Common Weakness Enumeration). Nämä ovat yleisesti tunnistettu yhdeksi merkittävimmistä lähteistä uusien tietoturva-vaivoituvuuksien julkaisussa ja luokittelussa.

Akateeminen tiedeyhteisö viittaa rutiininomaisesti CVE-haavoituvuuksiin ja CWE-luokitteluun julkaisuissaan. Kyberturvallisuusyhteisö ylläpitää ja käyttää myös muita lähteitä, kuten CERT/CC:n julkaisemia haavoituvuuksia, yritysten kuten Microsoftin ja Oraclen omia haavoituvuustietokantoja ym. Työssä keskitytään yleisimmin käytettyyn CWE-luokitteluun.

Mitre julkaisee vuosittain CWE Top25-listaa[21]. CWE:t jakaantuvat yli 600 eri luokkaan, joista jokainen kuvaa yhtä haavoituvuuden tyyppiä. Lista nostaa esiin yleisimmät haavoituvuustyyppit, joita hyökkääjät ovat kuluneen vuoden aikana käyttäneet hyväkseen. Listan kokoamisessa on käytetty hyväksi useita eri lähteitä, kuten NVD:tä (National Vulnerability Database), CERT/CC:tä, ja yritysten omia haavoituvuustietokantoja. CWE:t jakaantuvat yli 600 eri luokkaan, joista jokainen kuvaa yhtä haavoituvuuden tyyppiä.

Taulukossa 3.1 on viiden vuoden ajalta neljä yleisintä erityisesti C- ja C++ -kielten muistinkäsittelyvirheisiin liittyvää haavoituvuustyyppiä, jotka esiintyvät Mitren Top25-listalla. Kielille on yhteistä, että samankaltaisia muistinkäsittelyvirheitä ei tyypillisesti pysty tekemään muilla yleisesti käytössä olevilla korkean tason ohjelmointikielillä. CWE-476: NULL Pointer Dereference on mahdollista myös monilla muilla kielillä, kuten Go:lla, Javalla ja C#lla, joten sitä ei valittu tarkasteluun mukaan.

Taulukko 3.1. *CWE-tyypit ja niiden sijoitus eri vuosina CWE Top25-listalla.*

Tyyppi	2019	2020	2021	2022	2023
CWE-119	1	5	17	19	17
CWE-125	5	4	3	5	7
CWE-416	7	8	7	7	4
CWE-787	12	2	1	1	1

Haavoittuvuustyyppien koko nimet ovat:

- CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer
- CWE-125 - Out-of-bounds Read
- CWE-416 - Use After Free
- CWE-787 - Out-of-bounds Write

Em. haavoittuvuuksia on käytännössä hyödynnetty esim. OpenSSL-kirjaston Heartbleed-haavoittuvuudessa, joka ohjelmoitiin kirjastoon 1.2.2012, ja havaittiin aprillipäivänä 1.4.2014. Tyypiltään haavoittuvuus oli CWE-125, eli indeksin ohilukuvirhe [26]. OpenSSL-kirjaston Git-diffistä huomataan [27], että ohjelma ei tarkistanut käyttäjältä saatua syötettä, vaan paljasti virheellisen *malloc()*-komennon myötä käyttäjälle palvelimen muistin sisältöjä.

Lukuisia muistinkäyttöhaavoittuvuuksia löydetään jatkuvasti mm. käyttöjärjestelmistä ja Internet-selaimista. Usein haavoittuvuudet löydetään ennen kuin hyökkääjät ehtivät hyödyntää niitä laajamittaisesti, joten niistä ei aiheudu yhtä merkittäviä tietoturvaongelmia kuin Heartbleedistä. Yksittäisten tietoturva- haavoittuvuuksien hintaa on haastavaa arvioida, mutta erään lähteen mukaan kustannukset Heartbleed-haavoittuvuudesta olivat alkaen 500 miljoonaa dollaria [18].

3.1 CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer

Puhekielessä virhetyyppiä kutsutaan puskurin ylivuodoksi [24]. Se on selvästi menettänyt merkitystään Mitre:n Top25-listalla viime vuosien aikana. Yksi mahdollinen syy tälle on, että virhetyyppi on liian geneerinen, ja sen sijaan tyyppiluokittelussa on opittu käyttämään spesifimpiä virhetyyppejä. Mm. indeksin ohiluku- (CWE-125) ja kirjoitusvirheet (CWE-787) ovat puskurin ylivuotovirheen alaluokkia, ja Mitre suosittelee vahvasti käyttämään spesifimpiä haavoittuvuustyyppieitä. Lisäksi puskurin ylivuotovirhe terminä tarkoittaa eri käyttäjille eri asioita, joten tarkempi termi on otsikon mukainen *Improper Restriction of*

Operations within the Bounds of a Memory Buffer. Siksi työssä ei ole tästä virhetyypistä esimerkkiä, vaan esimerkin voi katsoa kohdista CWE-125 tai CWE-787.

3.2 CWE-125 - Out-of-bounds Read

Vapaasti suomennettuna indeksin ohiluku [22] on pysynyt suhteellisen samalla tasolla CWE-virhelistauksissa monen vuoden ajan. Virhe syntyy, kun esimerkiksi listasta yritetään lukea alkia indeksistä, joka ei ole listalla, kuten ohjelmakoodiesimerkissä liitteessä A.1. Tällöin ohjelma lukee tietoa sille varatun muistialueen ulkopuolelta, eikä ohjelman toiminta ole määriteltyä [9].

Ohjelma voi virhetilanteessa kaatua, tai tarjota hyökkääjälle mahdollisuuden lukea tietokoneen muistiin tallennettua tietoa, jota ohjelman ei ole tarkoitus lukea. Sopivalla syötteellä hyökkääjä voi saada tietoonsa esimerkiksi muistiin tallennettuja salasanoja, kryptografisia avaimia, tai lisää tietoa, joka avustaa hyökkääjää saamaan syvemmän pääsyn järjestelmiin.

Virhettä voi ohjelmointiteknisesti ehkäistä esimerkiksi olettamalla, että kaikki syöte on vaarallista. Ennen operaatiota voidaan tarkistaa kaikki relevantit muuttujat, kuten syöteen pituus, tyyppi, kaikki sallitut arvot, puuttuvat tai ylimääräiset syötteet, syntaksi ja johdonmukaisuus samankaltaisten muuttujien suhteen. Vähintään syötteet, joita käytetään indeksoinnissa tai osoittimien kanssa, pitää validoida, ja epäkelvot syötteet tulee hylätä tai muuntaa muotoon, jossa ne eivät aiheuta vaaraa.

3.3 CWE-416 - Use After Free

Vapaasti suomennettuna 'vapautuksen jälkeinen käyttö' eli VJK [23], on yksi C- ja C++-kielille tyypillisistä tietoturvaavaoittuvuuksista. Tilanne syntyy, kun osoittimen viittaamaan muistialuetta käytetään uudelleen muistin vapauttamisen jälkeen, kuten ohjelmakoodiesimerkissä liitteessä A.2. Kun vapautettua muistia yritetään käyttää saman tai toisen osoittimen kautta, tämä luo tilanteen, jossa osoitin viittaa muistialueeseen, joka ei enää sisällä alkuperäisiä tietoja. Koska C:n ja C++:n standardit eivät määrittele ohjelman käyttäytymistä tällaisessa tilanteessa, seuraukset voivat olla arvaamattomia [9] ja aiheuttaa vakavia tietoturvaongelmia.

Jotta hyökkääjä voi hyödyntää VJK-virhettä, ohjelman suorituksen aikana täytyy täyttyä kolme ehtoa [13]:

- Ohjelma luo riippuvia osoittimia vapauttamalla osoittimen osoittaman muistialueen.
- Hyökkääjä varaa muistia samalle muistialueelle, johon riippuva osoitin osoittaa.
- Riippuvaa osoitinta käytetään myöhemmin hakemaan tietoa jo aikaisemmin vapautetulta muistialueella luku- ja/tai kirjoitusoperaatioiden avulla.

Ehtojen täyttyminen ei ole triviaalia, vaan vaatii hyökkäjältä yleensä syvällistä perehtymistä ohjelman toimintaan, ja ohjelman ajonaikaisen suorituksen heikkouksien etsimistä erilaisin menetelmin. Haavoittuvuuksia voi yrittää vähentää estämällä yhden tai useamman edellä mainitun ehdon täyttymistä. Guin ym. [13] mukaan menetelmät voidaan jakaa viiteen eri luokkaan. Suuri osa menetelmistä hidastaa ohjelman ajonaikaista suorituskykyä, vaihdellen 0,6 – 540 % välillä.

3.4 CWE-787 - Out-of-bounds Write

Vapaasti suomennettuna indeksin ohikirjoitus [25] on vaarallisin tiedossa oleva tietoturva-vaavoittuvuus suhteutettuna haavoittuvuuksien yleisyyteen ja vakavuuteen. Virhe syntyy, kun esimerkiksi kolme alkioita pitkän listan 4. alkioon yritetään kirjoittaa tietoa, kuten ohjelmakoodiesimerkissä liitteessä A.3. Tällöin ohjelma kirjoittaa tiedot sille varatun muistialueen ulkopuolelle, eikä ohjelman toiminta ole määriteltyä [9].

Ohjelma voi virhetilanteessa esimerkiksi korruptoida validia tietoa, kaatua, tai antaa hyökkäjälle mahdollisuuden ajaa haluamaansa ohjelmakoodia. Sopivalla syötteellä hyökkääjä voi ylikirjoittaa ohjelman käytössä olevaa muistia, kuten paluuosoitteita tai funktiopointtereita. Tämä tarjoaa hyökkäjille mahdollisuuden muokata ohjelman suorituslogiikkaa, ja voi antaa hyökkäjälle pääsyn syvemmälle järjestelmään.

Virhettä voi ohjelmointiteknisesti ehkäistä esimerkiksi validoimalla kaikki ohjelman käytämät syötteet, tai vähintään syötteet, jotka vaikuttavat muistinhallintaan. Ennen muistio-paraatioita voidaan tarkistaa, onko indeksi validi vai ei. Jos indeksi on liian pieni tai liian suuri, sitä ei kannata käyttää.

4. VALTIOTASON SUOSITUKSET MUISTITURVALLISESTA OHJELMISTOKEHITYKSESTÄ

Noin vuoden sisään on julkaistu lukuisia eri kansainvälisiä suosituksia, jotka käsittelevät ohjelmistokehityksen turvallisuutta. Tarkasteluun valittiin kolme melko tuoretta raporttia, jotka sopivat tutkimuskysymyksen aihepiiriin. Raporttien sisällöstä tiivistetään muistiturvallisuuteen liittyvät osuudet ja suositukset. Luvuissa 2 ja 3 esiteltiin teoriapohjaa, jonka sisäistäminen auttaa ymmärtämään syitä, miksi tässä työssä tarkasteltavia suosituksia on laadittu, ja miksi useat eri tahot ovat lähteneet suosittamaan muistiturvallisten ohjelmointikielien käyttöä nykyistä laajemmin.

4.1 Secure by Design

CISA ym. on julkaissut lokakuussa 2023 turvallisuuslähtöisen ohjelmistokehityksen konseptin nimeltään “Secure by Design”. Ohjeen tarkoituksena on tarjota suosituksia yrityksille, joiden avulla he voivat viedä ohjelmistokehityksen prosessejaan turvallisuuslähtöiseen suuntaan [5]. Länsimaiset kyberturvallisuusvirastot ovat yhdessä huomanneet tarpeen suosituksille. Eräs mahdollinen syy voi olla alati kasvavat kyberrikollisuuden taloudelliset kustannukset ja muut tietoturvaloukkausten aiheuttamat haitat. Julkaisu on tarkoitettu etenkin niitä toimijoita varten, joiden käytännöt voisivat olla nykyistä turvallisempia.

He ehdottavat, että kaikkien ohjelmistoalan toimijoiden pitäisi suunnitella tuotteet ja ohjelmistojen oletusasetukset jo lähtökohtaisesti turvallisiksi. Yksi monista tavoista on korvata mahdollinen muistiturvattomien ohjelmointikielien käyttö mahdollisimman suurilta osin muistiturvallisilla ohjelmointikielillä. Lyhyellä tähtämellä ASLR, CFI, rautapohjainen mitigaatio ja fuzzaus voivat auttaa vähentämään muistivirheistä aiheutuvia tietoturvaongelmia. Toimijoita suositellaan julkaisemaan tiekartta, josta käy ilmi, miten he aikovat siirtyä turvattomista kielistä turvallisiin ohjelmointikieliin [5]. Konkreettisia muistiturvallisuuden tiekartan laadintaan liittyviä seikkoja käydään läpi erillisessä CISA ym:n julkaisussa joulukuulta 2023 [4].

4.2 Consumer Reports

Consumer Reports julkaisi muistiturvallisuutta yhteenvetävän raportin tammikuussa 2023. Aluksi toistetaan n. 60–70 %:n lukuja muistihaavoittuvuuksien esiintymisestä. Raportti myöntää C:n ja C++:n käytöstä luopumisen monitahoiset ongelmat, mutta ehdottaa, että pitkän aikavälin kulut voivat olla silti halvemmat, mitä aikaisemmin haastava siirtymä pois C:n ja C++:n käytöstä tehdään. Raportti sivuaa em. ohjelmointikielten käytön ongelmallisuutta opinahjojen koulutusohjelmissa, ja toisaalta yritysten odotuksia valmistuvien tekniikan alan osaajien kyvyistä [12].

Vakiintuneemmilla aloilla turvallisuus on nykyään itsestään selvää, mitä se ei aikanaan ole ollut. Turvallisuuden suhteen haetaan vertauksia mm. 1960-luvun autoteollisuuden turvallisuusvastaisuudesta, lääketieteestä ja ilmailusta. Monien eri alojen on pitänyt työskennellä pitkäjänteisesti turvallisuuden kehittämiseksi, ja Consumer Reportsin mukaan sama työmaa on edessä myös ohjelmistoalalla. Raportissa pohditaan monenlaisia tapoja vaiheittaisen muutoksen ja tietoisuuden lisäämisen tiimoilta. Lopussa suositellaan lyhyesti lukuisia erilaisia keinoja, joita alan toimijat voivat harkita, kuten muistiturvallisten ohjelmointikielten käyttöä uusia tuotteita tehdessä, ja vanhojen tuotteiden uusia osia tehdessä [12].

Kriittisimmät kirjastot ja ohjelmistopakettit olisi hyvä vaihtaa muistiturvallisilla kielillä toteutetuiksi ensin, aloittaen pehmeimmistä ja haavoittuvimmista hyökkäysvektoreista. Yrityksiltä toivotaan muistiturvalliisiin ohjelmointikieliin siirtymisen strategiaa ja avoimuutta muistiturvallisten ohjelmointikielten käytöstä ja ohjelmointivirheiden syistä. Raportissa toivotaan rahallisia tai lainsäädännöllisiä motivaatiotekijöitä muistiturvalliisiin ohjelmointikieliin siirtymiseen, ja julkisten PR-kampanjoiden organisointiin. Kampanjoiden kohteena tulisi olla mm. yritysten hallinto, journalistit ja kansalaiset [12].

4.3 Valkoinen Talo

Kirjoitushetkellä tuorein julkaisu on Valkoisen Talon raportti helmikuulta 2024, joka osana presidentti Joseph Bidenin kansallista kyberturvallisuusstrategiaa ehdottaa kahta muutosta: tietoturvaohjelmalta puolustautumisen vastuunkantoa ja pitkäjänteiseen kyberturvallisuustyöhön investointia. Raportin mukaan ohjelmistoprojektien ohjelmointikielten valintaan pitää kiinnittää enemmän huomiota etenkin muistiturvallisuuden osalta. Rautatason suunnittelun ja formaalien metodien tarjoamia mahdollisuuksia muistihaavoittuvuuksien vähentämisen osalta on syytä tutkia nykyistä monipuolisemmin. Lopuksi raportti ehdottaa kehittämään ohjelmistojen mitattavuutta [31].

Raportti myöntää, että tarjolla ei ole riittäviä motivaatiotekijöitä yrityksiä kohtaan, jotta ne suhtautuisivat tietoturvaongelmiin proaktiivisemmin jo alusta alkaen. Toiminta on liian usein reaktiivista, ja se ylipäätään hankaloittaa puolustajien kykyä ennakoita ja valmistautua tuleviin kyberhyökkäyksiin. Pelkästään olemassa olevien haavoittuvuuksien korjaus

ei riitä, sillä suuren riskin aiheuttavat ohjelmistoissa olevat vielä löytymättömät haavoittuvuudet. Muistinkäsittelyvirheet ovat raportin mukaan yksi vakavimmista ohjelmistojen tietoturva-avoittuvuuden lajeista [31].

Erityisesti C ja C++ ovat tunnettuja tästä, ja raportti esittää, kuten monet muut toimijat ovat jo aiemmin tehneet, muistiturvallisten ohjelmointikielien käyttöä. Raportti suosittelee tekemään ohjelmistojen suunnitteluvaiheessa päätöksiä, jotka eivät johda turvattomien ohjelmointikielien käyttöön. Jo olemassa olevien ohjelmistojen koodivarastojen osalta myönnetään, että täydellinen uudelleenkirjoitus on usein liian suuri haaste. Sen sijaan raportti suosittelee ainakin kriittisimpien osien uudelleenkirjoitusta muistiturvallisilla ohjelmointikielillä. Näin toimien haavoittuvuuden hyväksikäytön riskiä voidaan vähentää oleellisesti myös vanhoissa ohjelmistoissa [31].

4.4 Kotimaiset suositukset ja opinnäytteet

Suomen tasavallan viranomaistahojen tuottamassa kyberturvallisuusmateriaalissa [10, 19, 20, 32] ei mainita muistiturvallista ohjelmointia. Yksi vaihtoehto on, että muistiturvallisuudesta kyllä puhutaan, mutta tiedon löytäminen on haastavaa. Tämän vuoksi, ja ylipääntään sen varmistamiseksi, että onko kyse tosiaan siitä, että tietoa ei ole, vai onko se vain hyvin piilossa, Kyberturvallisuuskeskukselta kysyttiin sähköpostitse tilannekuvaa ja mielipidettä muistiturvallisen ohjelmoinnin nykytilan suositusten ja ohjeistusten suhteen.

20.3.2024 päivytyssä vastauksessaan Kyberturvallisuuskeskus myönsi, että varsinaista määräystä asiasta ei ole. He ovat esittämässä nelivuotista ohjelmistoturvallisuutta käsittelevää hanketta, jossa tulnaisiin tuottamaan lisää materiaaleja. Hankkeessa on kehittämisen lisäksi muitakin painopisteitä, kuten järjestelmän hankinta, sen elinkaaren hallinta ja ohjelmistoturvallisuuden koulutus. Samassa viestissä viitattiin, että olemassa olevana materiaalina ovat edellä mainittu kyberturvallisuusmateriaali, ja Valkoisen Talon myös edellä käsitelty tuore raportti.

Puhutaanko muistiturvallisesta ohjelmoinnista yleistasolla sitten suomalaisissa opinnäytteissä? Aihetta lienee soveliasta sivuta pintapuolisesti joka tapauksessa, koska valtiotason suosituksia aiheesta on haastavaa löytää. Onko muistiturvallinen ohjelmointi aihe, kenties, joita korkeakouluissa tutkitaan, vai onko kyse vain muutamien itseoppineiden harrastajien omasta mielenkiinnosta? Suomalaiset korkeakoulujen opinnäytteet ovat jaoteltu varsin kirjavasti erilaisiin tietoarkistoihin. Ammattikorkeakouluilla on yhteinen Theseus, kun taas jokaisella yliopistolla on omansa. Tarkastelun ulkopuolelle jätettiin MPKK, Taideyliopisto ja ruotsinkieliset yliopistot julkaistujen töiden kielen tai todennäköisenä pidetyn aihepiirin vuoksi.

Hakusanoina toimivat suomen kielellä **muistiturvallinen OR muistiturvallisuus** ja englanniksi **"memory safety"OR "memory-safe"OR "memory vulnerability"AND**

Taulukko 4.1. Tarkastelun kohteena olleet korkeakoulut ja tietovarastot

Korkeakoulu	Sähköinen opinnäytetyöarkisto
Ammattikorkeakoulut	https://www.theseus.fi
Aalto-yliopisto	https://aaltodoc.aalto.fi
Helsingin yliopisto	https://helda.helsinki.fi
Itä-Suomen yliopisto	https://erepo.uef.fi
Jyväskylän yliopisto	https://jyx.jyu.fi
Lapin yliopisto	https://lauda.ulapland.fi
LUT	https://lutpub.lut.fi
Oulun yliopisto	https://oulurepo.oulu.fi
Tampereen yliopisto	https://trepo.tuni.fi
Turun yliopisto	https://www.utupub.fi
Vaasan yliopisto	https://osuva.uwasa.fi

program* OR software. Kandidaatintyöt kirjoitetaan pääasiassa suomeksi, ja lukuisat korkeamman tason opinnäytteet tehdään englanniksi. Suomenkielisissä hakutuloksissa esiintyi muutama Rust-ohjelmointikieltä käsittelevä opinnäytetyö. Englanninkielisen hakutermin tarjosi myös hyvin niukalti tuloksia, joista kolme vaikuttivat edes jossain määrin relevanteilta:

- Nyman, T. Toward Hardware-assisted Run-time Protection. Tietotekniikan laitos, Aalto-yliopisto. Väitöskirja. Marraskuu 2020.
- Liljestrand, H. Hardware-assisted memory safety. Aalto-yliopisto. Väitöskirja. Tammi-kuu 2020.
- Valkonen, Ville. Proactive security measures in coding. Informaatitieteiden laitos, Tampereen yliopisto. Pro gradu -tutkielma. Joulukuu 2024.
- Shajarian, S. The C++ programming language in modern computer science. Informaatio-tekniikan ja viestinnän tiedekunta, Tampereen yliopisto. Pro gradu -tutkielma. Huhtikuu 2020.
- Sidoroff, I. Evaluating Approaches for Detecting and Eliminating Memory Safety Errors in Linux Kernel Programming. Jyväskylän ammattikorkeakoulu. YAMK-opinnäytetyö. Kesäkuu 2019.

5. KESKUSTELU

Suosituksen perusteella on selvää, että muistiturvattomien ohjelmointikielien käyttöä on syytä vähentää mahdollisuuksien mukaan. Suosituksia voisi jalkauttaa käytäntöön lakien ja säädösten avulla. Varsinaista muistiturvallista ohjelmointia määräävää lainsäädäntöä ei kirjoitushetken näkymien perusteella ole länsimaissa tulossa, ellei rivien välistä halua Euroopan Unionin Kyberresilienssisäädöstä [11] tulkita sellaiseksi. Euroopan Unioni on perinteisesti katsonut lainsäädännöllisen tason toimenpiteet tarpeellisiksi. Mikä on EU:n kyberturvallisuussäädösten lopullinen muoto, ja miten sen vaatimukset implementoidaan kansallisessa lainsäädännössä, jää nähtäväksi. Päivitetäänkö kotimaisia suosituksia suhtautumaan kriittisemmin muistiturvallisuudesta aiheutuviin ongelmiin?

Yksinkertaisen haun perusteella kotimaisissa yliopistojen ja ammattikoulujen opinnäytetöissä ei käsitellä muistiturvallisuutta yleisellä tasolla. Verrokkina samalla luvussa 4.4 käytetyllä englanninkielisellä hakutermillä Google Scholar -hakukone tarjosi noin 11200 hakutulosta. Mahdollisena rajoitteena tässä työssä käytetty yksinkertainen haku ei löytänyt kaikkia relevantteja opinnäytetöitä. Laajempi katsaus myös konferenssijulkaisuihin ja lehtiartikkeleihin voi antaa lisätietoa vallitsevasta tilanteesta.

Kenties muistiturvallisen ohjelmoinnin vaikutus kyberturvallisuuteen ei ole kovinkaan yleinen tutkimuksen aihe suomalaisissa korkeakouluissa? Kun tätä peilaa viimeaikaisiin kansainvälisiin julkaisuihin, kansainvälisellä tasolla muistiturvallinen ohjelmointi kiinnostaa selvästi enemmän monia eri tahoja, kuten korkeakouluja, kuluttajia edustavia tahoja, ylintä valtiojohtoa ja eri kyberturvallisuuselimiä. Jos muistiturvallinen ohjelmistokehitys on niin tärkeä aihe kuin ulkomaiset suositukset antavat ymmärtää, olisiko aihetta syytä tutkia enemmän myös Suomessa?

5.1 C++:n kehittäjien reagointi turvallisuuskritiikkiin

Muistivirheitä voi ehkäistä monella tapaa, mutta edes ne yhdessä eivät takaa täydellistä turvaa. C++ -kielen luoja, Bjarne Stroustrup, piti CppCon 2023:ssa avajaispuheen "Delivering Safe C++" viime syksynä [30]. Yli tunnin kestäneessä puheessaan Stroustrup myönsi, että staattinen analyysi ei takaa täydellistä turvallisuutta. Ohjeita tai sääntöjä ei noudateta ilman pakotusta. Peruskirjastot eivät anna täyttä pääsyä laitteistoon. Kielen osajoukon luominen ei toimi, koska vaarallisimmat kielen ominaisuudet ovat elintärkeitä. Puheessaan

hän ehdotti kokonaisvaltaisemmaksi ratkaisuksi Ada-ohjelmointikielen kaltaisia profileja myös C++:aan. On vaikeaa ennustaa, ottaako ehdotus tuulta alleen, ja millä aikataululla mahdollisia implementaatioita nähdään. Vaihtoehtoiset ohjelmointikieliet ovat jo nyt olemassa, joten ne ja valtiotason suositukset ja jopa mahdollinen lainsäädäntö painostavat perinteisesti hitaasti muuttuvaa C++:aa ja sitä käyttäviä tahoja uudistumaan. Mihin suuntaan C- ja C++ -kieliä kehitetään suositusten perusteella?

Bjarne Stroustrup esitti puheessaan laskelman, jossa keskikokoisen n. 10 miljoonaa ohjelmakoodiriviä sisältävän tuotteen uudelleenkirjoitus maksaa karkeasti noin miljardi dollaria yhdysvaltalaisella palkkatasolla [30]. Vaikka olemassa olevien tuotteiden uudelleenkirjoituksesta koituu suuret kulut, voivat pitkällä tähtäimellä vertailtuna kyberrikollisuudesta aiheutuvat kulut nousta silti suuremmiksi. Ehkä tätäkin aihetta on syytä tutkia tarkemmin? Yksi mahdollinen tulevaisuuden suuntaus on, että vaihtoehtoisten, muistiturvallisten ohjelmointikielten käyttö lisääntyy, jos C++ ei uudistu riittävän nopeasti. Suuntausta motivoivat mainitut alati kasvavat kustannukset, ja se, että edes kaikki nyt käytettävissä olevat virheitä ehkäisevät työkalut ja menetelmät eivät ole onnistuneet vähentämään muistivirheiden aiheuttamien tietoturvaavaoittuvuuksien suhteellista määrää.

5.2 Tulosten merkitys korkeakouluopetuksessa

Merkittävä osa ohjelmistoalan osaajista saa formaalia ohjelmistokehityksen opetusta korkeakouluissa, joten muistiturvallisuuden ongelmia ei siksi kannata tarkastella suppeasti pelkästään tuotantopuolen haasteena. On syytä pohtia, pitäisikö myös opetusta sopeuttaa tuoreiden suositusten mukaiseksi. Yliopisto-opetuksen ideaali on tarjota opiskelijoille vankka perusta tieteellisen tiedon tuottamiseen ja tieteellisen metodin soveltamiseen.

Perinteisesti yliopistojen keskeisenä tehtävänä on ollut valmistaa opiskelijat kohtaamaan monimutkaisia haasteita, innovoimaan ja edistämään oman tieteenalansa kehitystä. Opetuksen on siksi hyvä perustua ajantasaiseen ja kriittisesti arvioituun tietoon, ja sen on tuettava opiskelijoiden kykyä itsenäiseen ajatteluun ja ongelmanratkaisuun. Vaikka C ja C++ ovat yritysmaailmassa yleisesti käytettyjä kieliä, jotka mahdollistavat suorituskykyisen järjestelmätason ohjelmoinnin, niiden käyttöön liittyy merkittäviä ongelmia. Nykysuositusten valossa em. kielten opettaminen edellyttää huomattavaa panostusta muistiturvallisuuden opettamiseen, mikä voi viedä aikaa muilta tärkeiltä opetuksen osa-alueilta.

Sopisiko C- ja C++ -kielten huonojen tapojen opettaminen, jos se ylipäätään katsotaan tieteellisesti tarpeelliseksi, paremmin esimerkiksi vapaaehtoiseksi osaksi maisteriopintoja. Maisterivaiheessa opiskelijoilla olisi vankempi perusta ohjelmoinnista, ja he ovat valmiimpia käsittelemään C- ja C++ -kielten monimutkaisuutta ja vaaroja. Kandidaattitasolla voitaisiin keskittyä mieluummin ohjelmoinnin perusteisiin, ohjelmistosuunnitteluun ja turvallisuuslähtöisen kehityksen periaatteisiin, käyttäen kieliä, jotka automatisoivat muistinhallinnan ja siten vähentävät turvallisuusriskien määrää.

6. YHTEENVETO

Läpikäytyjen suositusten perusteella vaikuttaa, että muistiturvallinen ohjelmointi ja siihen siirtyminen näyttelee tärkeää roolia ulkomaisissa valtiotason ohjelmistokehityksen turvallisuussuosituksissa. Suomessa näin ei toistaiseksi vaikuta olevan. Voi olla, että Suomen viranomaisten toimesta muistiturvallisuus on katsottu muuttujaksi, johon ei ole ollut tarpeellista kiinnittää erityistä huomiota ohjelmistokehityksen suosituksissa.

Muistiturvallinen ohjelmistokehitys C- ja C++ -kielillä on erittäin haastavaa, eikä turvalliselle tasolle ole päästy vuosikymmeniä jatkuneesta tutkimuksesta ja kehityksestä huolimatta. Ensisijaisesti suositukset ehdottavat pitkällä tähtäimellä turvattomista kielistä luopumista, sillä nykytiedon valossa vaikuttaa siltä, että turvallisuutta on hyvin vaikea rakentaa kieliin mukaan jälkikäteen. Toissijaisesti suositukset ehdottavat, että jo olemassa olevien ohjelmistojen tietoturvakriittisimmät osat toteutettaisiin mahdollisuuksien mukaan uudestaan muistiturvallisilla ohjelmistokielistä. Osittaiset ongelmanratkaisumenetelmät keskittyvät tavanomaisesti erilaisiin mitigaatiomenetelmiin, ja ohjelmakoodin ja käännettyjen ohjelmien analysointiin. Vaikka näitä menetelmiä ei työssä sen tarkemmin esitelty, on hyvä tiedostaa, että niitä käytetään laajasti. On hyvin mahdollista, että ilman em. menetelmiä tilanne olisi vielä nykyistä selvästi pahempi.

C- ja C++ -kielet ovat muistiturvallisuuuteen liittyvistä ongelmista huolimatta perinteisesti olleet riittävän hyviä moniin eri käyttötarkoituksiin. Koska turvattomilla kielillä on ohjelmoitu valtavia määriä ohjelmistoja, on niistä luopuminen haastava, kallis ja pitkä prosessi. Nykyisten C:n ja C++:n käyttäjien osaaminen ja työkalut ovat orientoituneita kielten käyttöön, ja korvaajiksi ehdotettuihin muistiturvalliisiin ohjelmointikieliin siirtymisessä on omat haasteensa, mm. työntekijöiden osaamisen, puutteellisen laitteistotuen, kehitystyökalujen ja uudenlaisten prosessien vuoksi. Siksi onkin todennäköistä, että C- ja C++ -kielten käyttöä jatketaan ohjelmistokehityksessä vielä pitkiä aikoja niiden ongelmista huolimatta.

LÄHTEET

- [1] Hala Assal ja Sonia Chiasson. ”Security in the Software Development Lifecycle”. *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. Baltimore, MD: USENIX Association, elokuu 2018, ss. 281–296. ISBN: 978-1-939133-10-6. URL: <https://www.usenix.org/conference/soups2018/presentation/assal>.
- [2] Manish Bhurtel ja Danda B. Rawat. ”Unveiling the Landscape of Operating System Vulnerabilities”. *Future Internet* 15.7 (2023). ISSN: 1999-5903. DOI: 10.3390/fi15070248. URL: <https://www.mdpi.com/1999-5903/15/7/248>.
- [3] Catalin Cimpanu. *Microsoft: 70 % of all security bugs are memory safety issues*. Helmikuu 2019. URL: <https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/> (viitattu 17. 03. 2024).
- [4] CISA, NSA, FBI, ACSC, CCCS, NCSC-UK, NCSC-NZ ja CERT NZ. *The Case for Memory Safe Roadmaps - Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously*. Joulukuu 2023. URL: <https://www.cisa.gov/sites/default/files/2023-12/The-Case-for-Memory-Safe-Roadmaps-508c.pdf>.
- [5] CISA, NSA, FBI, ACSC, CCCS, CERT NZ, NCSC-NZ, NCSC-UK, BSI, NCSC-NL, NCSC-NO, NÚKIB, INCD, KISA, NISC-JP, JPCERT CC, CSA ja CSIRTA-MERICAS. *Secure by Design. Shifting the balance of cybersecurity risk: Principles and approaches for secure by design software*. Lokakuu 2023. URL: https://www.cisa.gov/sites/default/files/2023-10/SecureByDesign_1025_508c.pdf.
- [6] cppreference.com. *Pointers*. Joulukuu 2013. URL: <https://en.cppreference.com/book/pointers> (viitattu 17. 03. 2024).
- [7] cppreference.com. *free*. Lokakuu 2022. URL: <https://en.cppreference.com/w/c/memory/free> (viitattu 17. 03. 2024).
- [8] cppreference.com. *malloc*. Lokakuu 2022. URL: <https://en.cppreference.com/w/c/memory/malloc> (viitattu 17. 03. 2024).
- [9] cppreference.com. *Undefined behavior*. Marraskuu 2023. URL: <https://en.cppreference.com/w/cpp/language/ub> (viitattu 26. 01. 2024).
- [10] Digi- ja väestötietovirasto. *Turvallisen sovelluskehityksen käsikirja*. URL: <https://wiki.dvv.fi/display/SOVOP> (viitattu 20. 03. 2024).
- [11] Euroopan komissio. *Euroopan parlamentin ja neuvoston asetukset digitaalisista elementeistä sisältävien tuotteiden horisontaalisista kyberturvavaatimuksista ja asetusten (EU) 2019/1020 muuttamisesta*. Asetusehdotus. 2022.

- [12] Y. Grauer. *Future of Memory Safety – Challenges and Recommendations*. Tammi-kuu 2023. URL: <https://advocacy.consumerreports.org/wp-content/uploads/2023/01/Memory-Safety-Convening-Report-1-1.pdf>.
- [13] Binfa Gui, Wei Song, Hailong Xiong ja Jeff Huang. ”Automated Use-After-Free Detection and Exploit Mitigation: How Far Have We Gone?” *IEEE Transactions on Software Engineering* 48.11 (2022), ss. 4569–4589. DOI: 10.1109/TSE.2021.3121994.
- [14] *How security flaws work: The buffer overflow*. URL: <https://arstechnica.com/information-technology/2015/08/how-security-flaws-work-the-buffer-overflow/> (viitattu 17. 03. 2024).
- [15] Nai Fovino I, Barry G, Chaudron S, Coisel I, Dewar M, Junklewitz H, Kampourakis G, Kounelis I, Mortara B, Nordvik JP, Sanchez Martin JI, Baldini G, Barrero J, Chaudron S, Coisel I, Draper Gil G, Duch Brown N, Eulaerts O, Geneiatakis D, Hernandez Ramos JL, Joanny G, Junklewitz H, Kampourakis G, Kerckhof S, Kounelis I, Lewis A, Martin T, Nai Fovino I, Nativi S, Neisse R, Nordvik JP, Papameliou D, Reina V, Ruzzante G, Sanchez Martin JI, Sportiello L, Steri G ja Tiren-di S. *Cybersecurity, our digital anchor: A European perspective*. Tekninen raportti KJ-NA-30276-EN-N (online), KJ-NA-30276-EN-C (print). Luxembourg, 2020.
- [16] International Business Machines Corporation. *Cost of a Data Breach Report 2023*. Tekninen raportti. 2023. URL: <https://www.ibm.com/reports/data-breach>.
- [17] Federal Bureau of Investigation. *FBI - Morris Worm*. URL: <https://www.fbi.gov/history/famous-cases/morris-worm> (viitattu 17. 03. 2024).
- [18] Sean Michael Kerner. *Heartbleed SSL Flaw’s True Cost Will Take Time to Tally*. Huhtikuu 2014. URL: <https://www.eweek.com/security/heartbleed-ssl-flaw-s-true-cost-will-take-time-to-tally/> (viitattu 20. 03. 2024).
- [19] M. Kiravuo, P. Timlin, K. Kemppainen, J. Eronen ja S. Seppänen. *Ohjelmistoturvallisuuden tila 2023*. Raportti. Lokakuu 2023. URL: <https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/publication/Ohjelmistoturvallisuuden%20tila%202023.pdf>.
- [20] Kyberturvallisuuskeskus. *Turvallinen tuotekehitys: kohti hyväksyntää*. 2018. URL: https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/publication/Turvallinen_tuotekehitys_Suomi_J003_2018.pdf.
- [21] MITRE Corporation. *CWE - CWE Top 25 Most Dangerous Software Weaknesses*. Joulukuu 2023. URL: <http://cwe.mitre.org/top25/> (viitattu 18. 01. 2024).
- [22] MITRE Corporation. *CWE - CWE-125: Out-of-bounds Read (4.14)*. Lokakuu 2023. URL: <https://cwe.mitre.org/data/definitions/125.html> (viitattu 17. 03. 2024).
- [23] MITRE Corporation. *CWE - CWE-416: Use After Free (4.13)*. Lokakuu 2023. URL: <https://cwe.mitre.org/data/definitions/416.html> (viitattu 26. 01. 2024).

- [24] MITRE Corporation. *CWE - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (4.14)*. Helmikuu 2024. URL: <https://cwe.mitre.org/data/definitions/416.html> (viitattu 17. 03. 2024).
- [25] MITRE Corporation. *CWE - CWE-787: Out-of-bounds Write (4.14)*. Helmikuu 2024. URL: <https://cwe.mitre.org/data/definitions/787.html> (viitattu 17. 03. 2024).
- [26] National Vulnerability Database (NVD), National Institute of Standards and Technology (NIST). *CVE-2014-0160: Heartbleed*. <https://nvd.nist.gov/vuln/detail/cve-2014-0160>. Heinäkuu 2014. (Viitattu 20. 03. 2024).
- [27] OpenSSL Project. *Fix for CVE-2014-0160 (Heartbleed) in OpenSSL Git repository*. 2014. URL: <https://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=96db902;ds=sidebyside> (viitattu 20. 03. 2024).
- [28] D.A. Patterson ja J.L. Hennessy. *Computer Organization and Design: The Hardware/software Interface*. The Computer Architecture and Design Series. Elsevier/Morgan Kaufmann, 2004. ISBN: 9781558606043. URL: <https://books.google.fi/books?id=4pcvjgEACAAJ>.
- [29] Alex Rebert ja Christoph Kern. *Google Online Security Blog: Secure by Design: Google's Perspective on Memory Safety*. Maaliskuu 2024. URL: <https://security.googleblog.com/2024/03/secure-by-design-googles-perspective-on.html> (viitattu 17. 03. 2024).
- [30] Bjarne Stroustrup. "Delivering safe C++". CppCon 2023. 2023. URL: <https://www.youtube.com/watch?v=I8UvQKvOSSw>.
- [31] The White House. *Back to the Building Blocks: A Path Toward Secure and Measurable Software*. Lehdistöjulkaisu. Helmikuu 2024. URL: <https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf> (viitattu 04. 03. 2024).
- [32] Valtiovarainministeriö. *Sovelluskehityksen tietoturvaohje*. VAHTI 1/2013. Valtiovarainministeriö, 2013. URL: https://finlex.fi/data/normit/41655-VAHTI_1_Sovelluskehityksen_tietoturvaohje_NETTI.pdf.
- [33] Victor van der Veen, Nitish Dutt-Sharma, Lorenzo Cavallaro ja Herbert Bos. "Memory Errors: The Past, the Present, and the Future". *Lecture Notes in Computer Science* 7462 (syyskuu 2012). DOI: 10.1007/978-3-642-33338-5_5.

LIITE A: OHJELMAKOODIESIMERKIT

Luvussa esitetyt koodiesimerkit on lainattu suoraan Mitre Corporationin CWE-tietokannasta. Esimerkit ovat havainnollistamassa, miten yksinkertaista vakavia tietoturvaongelmia aiheuttavien virheiden tekeminen on C- ja C++ -kielillä. Kriittinen rivi on kommentoitu ohjelmakoodissa suomeksi.

Ohjelma A.1. Esimerkki indeksin ohilukuvirheestä [22].

```

1 int getValueFromArray(int *array, int len, int index) {
2     int value;
3     // Tarkistetaan ainoastaan, onko indeksi liian suuri
4     if (index < len) {
5         value = array[index];
6     } else {
7         printf("Value_is:_%d\n", array[index]);
8         value = -1;
9     }
10    return value;
11 }

```

Ohjelma A.2. Esimerkki osoittimen vapautuksen jälkeisen käytöstä [23].

```

1 char* ptr = (char*)malloc (SIZE);
2 if (err) {
3     abrt = 1;
4     free(ptr);
5 }
6 if (abrt) {
7     // Käytetään osoitinta, vaikka se on vapautettu
8     logError("operation_aborted_before_commit", ptr);
9 }

```

Ohjelma A.3. Esimerkki indeksin ohikirjoitusvirheestä [25].

```

1 int id_sequence[3];
2 id_sequence[0] = 123;
3 id_sequence[1] = 234;
4 id_sequence[2] = 345;
5 id_sequence[3] = 456; // Ohi indeksin

```