Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi

Kandidaatintyö

11. toukokuuta 2011

Ferrix Hovi

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Ferrix Hovi
Työn nimi:	Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi
Päiväys:	11. toukokuuta 2011
Sivumäärä:	14
Pääaine:	Ohjelmistotuotanto ja -liiketoiminta
Koodi:	T3003
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	TkT Ari Korhonen (Tietotekniikan laitos)

Käytännön harjoittelulla on merkittävä rooli ohjelmoinnin opetuksessa ja kurssit ovat usein suuria. Automaattisella arvostelulla on tässä suuri oppimista tehostava vaikutus. Toisaalta tuntemattoman ohjelmakoodin suorittaminen on aina riski. Tämä työ tutkii kahta Teknisellä korkeakoululla kehitettyä kotitehtävätarkistinta turvallisuusnäkökulmasta ja selvittää, voiko staattista analyysiä lisäämällä parantaa näiden turvallisuutta. Lisäksi tutkitaan opetettavan ohjelmointikielen vaikutusta tarkistimeen liittyviin riskeihin.

Teknisellä korkeakoululla kehitetyt tarkistimet EXPACA ja Scheme-robo ovat lähestymistavoiltaan hyvin erilaisia. Edellinen suorittaa käännettyä ohjelmaa virtuaalikonsolissa ja jälkimmäinen suorittaa Scheme-proseduuria metasirkulaarisessa tulkissa. Scheme-robo on onnistuneesti rajannut kielen riskialttiit ominaisuudet pois eikä siinä siksi ole ilmeisiä turvallisuusriskejä. EXPACA suorittaa täysimittaisia ohjelmia, jotka on toteutettu virhealttiilla kielillä. Tämän takia staattisen analyysin lisäämisestä tarkistusprosessiin voisi olla hyötyä.

Staattiset analysaattorit antavat yleisesti ottaen niin paljon vääriä hälytyksiä, ettei niiden käyttö automaattisessa arvostelussa ole mielekästä. Pscan ja UNO ovat analysaattoreita, jotka havaitsevat yhteensä neljän tyyppisiä ongelmia. Ne tuottavat keskimääräistä vähemmän vääriä hälytyksiä ja soveltuvat mahdollisesti EXPACA:n laajentamiseen.

Työssä havaitaan myös, että opetettavalla ohjelmointikielellä ja staattisen analyysin opetuksella voi olla suuri merkitys opiskelijoiden tietämykselle turvallisesta ohjelmointitavasta.

Avainsanat:	staattinen analyysi, automaattinen arvostelu
Kieli:	Suomi

Sisältö

1 Johdanto

Ohjelmoinnilla on olennainen osa tietotekniikan opetuksessa ja käytännön harjoittelulla on ohjelmoinnin opiskelussa merkittävä rooli. Ripeästi saatavalla palautteella on oppimista tehostava vaikutus. Kurssit ovat monesti, esimerkiksi TKK:lla, osallistujamääriltään suuria. Tehtävien arvosteleminen tehokkaasti käsin kokonaisen vuosikurssin kokoisilla kursseilla on käytännössä mahdotonta. Automaattisen arvostelun edut ovat myös merkittävimpiä opintojen alussa, jolloin soveltavien harjoitusten osuus on pienempi (Carter et al., 2003). Näistä syistä monet opettajat, yliopistot ja yhtiöt ovat kehittäneet työkaluja, joilla opiskelijat voivat tarkistuttaa ohjelmointiharjoituksensa automaattisesti, saada palautetta ja kurssin suorittamiseksi tarvittavia pisteitä.

Tuntemattoman ohjelmakoodin suorittaminen on aina turvallisuusriski ja se pitää ottaa huomioon ohjelmointiharjoitusten tarkistimissa. Parhaassa tapauksessa riittävän tarkasti havaittu turvallisuusriski tarjoaa opiskelijalle mahdollisuuden ymmärtää ja oppia tekemästään virheestä turvallisessa ympäristössä ennen työelämään siirtymistä. Tietotekniikka on arkipäiväistynyt ja sen merkitys on lisääntynyt. Ohjelmointivirheiden riskit ovat samalla kasvaneet, kun useammat yhteiskunnan toiminnot luottavat ohjelmiin. Turvallinen ohjelmointitapa on helpompi oppia, jos ensimmäisillä ohjelmointikursseilla saa palautetta riskialttiin koodin kirjoittamisesta. Pahimmassa tapauksessa huonosti toteutettu järjestelmä päätyy koko kurssin suorittamista haittaavaan virhetilaan opiskelijan tahattoman virheen takia.

Haittaohjelmatutkimuksessa tutkitaan binäärimuotoisia ohjelmia ja pyritään tunnistamaan niiden aikeet, ennen kuin ne pääsevät haittaamaan järjestelmän toimintaa ja turvallisuutta. Haittaohjelmia tutkitaan staattisella ja dynaamisella analyysillä, mutta haitallisia ohjelmia ei yleensä päästä tutkimaan niiden lähdekoodimuodossa. Ohjelmointiharjoitusten tarkistimille annetaan syötteenä lähdekoodi, josta saatetaan kääntää mahdollisesti haitallinen ohjelma. Miten haittaohjelman lähdekoodi voitaisiin havaita?

Tämä kandidaatintyö tutkii tuntemattoman lähdekoodin staattista analyysiä tapana arvioida, onko ohjelman suorittaminen turvallista. Tutkimuksen tavoitteena on kartoittaa ohjelmointitehtävien automaattisten tarkistimien toteutukseen liittyviä turvallisuushaasteita ja löytää ratkaisuja kirjallisuudesta.

Tässä luvussa määritellään työn tavoite ja rajataan tutkittava ongelma, toisessa luvussa kerrotaan käytetystä aineistosta sekä tutkimusmenetelmistä, kolmannessa luvussa käydään läpi kirjallisuustutkimuksessa tehdyt havainnot ja viimeisessä luvussa muodostetaan tulosten pohjalta johtopäätöksiä.

Tämän työ vastaa seuraaviin tutkimuskysymyksiin:

- Voiko staattisella analyysillä varmistua lähdekoodin turvallisuudesta?

- Voiko kotitehtävätarkistimien turvallisuutta parantaa lisäämällä staattista analyysiä?
- Mikä on kurssilla opetettavan ohjelmointikielen vaikutus turvallisuusriskeihin?

1.1 Tavoite

Työn tavoitteena on löytää menetelmiä, jolla voidaan vahvistaa ohjelmointiharjoitusten arvosteluun käytettäviä työkalujen turvallisuutta, jotta ne selviäisivät tahattomasti ja tahallisesti tehdyistä hyökkäyksistä eheinä ja toimintakykyisinä. Työssä tutkitaan kahta Teknillisellä korkeakoululla (TKK) käytettyä tarkistinta ja vertaillaan niiden turvallisuusominaisuuksia. Tämän jälkeen tutkitaan muutamia järjestelmien vaatimuksiin sopivia ja luotettavia staattisen analyysin työkaluja. Järjestelmän olennaisia vaatimuksia ovat nopea palaute, väärien hälytysten määrän minimointi ja käytettävyys. Turvallisuuden lisääminen ei saa merkittävästi haitata järjestelmän toimintaa.

Tarkistin on ohjelma, joka suorittaa opiskelijalta saadun harjoituksen, testaa sen toimintaa ja pisteyttää tehtävän. Yleensä tarkistin on osa suurempaa kurssinhallintajärjestelmää, joka huolehtii tulosten kirjaamisesta, opiskelijoiden tunnistamisesta ja muista kurssin järjestelyihin liittyvistä asioista. Tässä työssä keskitytään ainoastaan tarkistimeen ja sen turvallisuuteen.

Väärä hälytys tarkoittaa sellaista virhehavaintoa, joka johtaa opiskelijan antaman harmittoman syötteen hylkäämiseen. Tämä johtaa arvostelun epäluotettavuuteen ja mahdollisesti ansiokkaiden suoritusten virheelliseen hylkäämiseen. Scheme-robon tilastojen perusteella opiskelijat palauttavat harjoitustöitään juuri ennen määräaikaa (Saikkonen et al., 2001). Opiskelijoiden usko järjestelmän oikeudenmukaisuuteen voi horjua, jos järjestelmä antaa paljon vääriä hälytyksiä viime hetkillä ja pisteitä jää saamatta virheen takia. Tämä aiheuttaa kurssihenkilökunnalle lisätyötä.

Tässä työssä käytettävyydellä tarkoitetaan järjestelmän käyttäjän tuntumaa ohjelman käytön helppoudesta ja sujuvuudesta. Turvallisuuden lisäämisestä ei saa aiheutua loppukäyttäjälle näkyvää muutosta järjestelmän toiminnassa. Kohtuullinen lisäys arvosteluun kuluvaan aikaan ja yksityiskohtaisempi palaute ovat kuitenkin sallittuja poikkeuksia.

Monissa järjestelmissä opiskelijat voivat yrittää ratkaista tehtäviä useamman kerran ja korjata ratkaisuista löytyneitä virheitä palautteen perusteella. Palaute on nopeaa, jos opiskelija ehtii saada tehtävistään palautteen, ennen kuin siirtyy muihin toimiin. Tällöin opiskelija pystyy työskentelemään tehokkaammin ja tehdä yhtä asiaa kerrallaan.

1.1.1 Tarkistimen turvallisuus

Ohjelmointiharjoituksen tarkistimessa voidaan varautua tietoturvaongelmiin kahdella tasolla: huolehtimalla tarkistimen toteutuksen turvallisuudesta ja tutkimalla syötteenä tulevaa ohjelmakoodia tavoitteena havaita sen suorittamisesta aiheutuvia sivuvaikutuksia ennen vihamielisen koodin suorittamista. Tässä työssä tutkitaan opiskelijalta tulevaa ohjelmakoodia.

Toteutuksen turvallisuudella tarkoitetaan järjestelmän toteuttamista turvallisella ohjelmointitavalla, eristämistä muista järjestelmistä ja muita turvallisuuteen vaikuttavia teknisiä valintoja. Tarkistimen pitää olla turvallisesti toteutettu, jotta sen antamat pisteet ja palaute tulevat varmasti tarkistimelta itseltään eivätkä esimerkiksi opiskelijan koodista. Tarkistimen tulee olla eristetty muusta järjestelmästä, jotta ohjelmointivirhe tai hyökkäys ei aiheuta häiriötä muussa järjestelmässä tai vaikuta arvostelutietojen eheyteen. Muita turvallisuuteen liittyviä teknisiä valintoja ovat esimerkiksi käytettyjen kirjastojen toteutuksen turvallisuus, arkkitehtuurin luotettavuus ja käytetyn ohjelmointikielen tyypilliset riskit.

Tämä työ olettaa tarkistimeen kohdistuvien riskien olevan pääasiassa tahattomia ohjelmointivirheitä. Todelliset riskit ovat varmasti monitahoisempia, mutta tarkempi riskianalyysi ei kuulu tämän työn alueeseen. Tarkistimet edellyttävät, että ohjelmointiharjoitukset ovat tarkasti rajattuja ja niiden syötteet ja tulosteet tarkasti määriteltyjä. Tästä syystä ne soveltuvat parhaiten peruskursseille, joilla on vähemmän soveltavaa harjoittelua. Opiskelijoilla ei pitäisi olla syitä tai tarpeellisia kykyjä järjestelmän murtamiseen. Kursseille voi osallistua kokeneempia ohjelmoijia, mutta he menestyvät todennäköisesti keskimääräistä paremmin (Hagan ja Markham, 2000). Myös heille tehtävien ratkaiseminen on helpompaa kuin tietojärjestelmän murtaminen. Omalla henkilöllisyydellä esiintyminen ja tutkinto tavoitteena vähentävät todennäköisesti motivaatiota rikoksiin.

1.2 Rajaukset

Ohjelmointiharjoituksien ohjelmakoodia voidaan arvioida ennen ja jälkeen käännöksen. Nykyaikaiset haittaohjelmat pyrkivät vaikeuttamaan koodin automaattista analysointia salaamalla ja pakkaamalla jo käännettyä ohjelmaa. Tällaiset menetelmät on rajattu tämän työn ulkopuolelle, koska koodin kirjoittaja ei pääse vaikuttamaan ohjelmakoodiin sen kääntämisen aikana tai sen jälkeen.

Ohjelmointitehtävien arvosteluun liittyy myös vilpin riski. Harjoitustehtäviä voidaan kopioida eikä etenkään etäopetuksen tapauksessa tehtävien tekijän henkilöllisyydestä voida olla täysin varmoja (Carter et al., 2003). Tämä työ ei kuitenkaan etsi ratkaisua näihin ongelmiin.

Koodin suorittaminen virtualisoiduissa tai emuloiduissa ympäristöissä on vaativa ongelma-alue (Kesti, 2010). Tämä työ ei ota kantaa virtualisointiin liittyviin ongelmiin, mutta on silti suositeltavaa ajaa kotitehtävätarkistinta rajoitetussa ympäristössä.

2 Aineisto ja menetelmät

Tässä kandidaatintyössä tutustutaan ensin kahteen Teknisellä korkeakoululla käytettyyn tarkistusjärjestelmään: Scheme-roboon ja Goblinin tarkistimeen EXPACA:an. Järjestelmiä tarkastellaan niistä kirjoitetun tieteellisen kirjallisuuden avulla. Tarkistimien lisäksi tutustutaan yleisimpiin ohjelmistoihin liittyviin turvallisuusuhkiin. Tarkistimia tutkimalla perehdytään myös kurssilla opetettavaksi valitun ohjelmointikielen turvallisuusvaikutukseen.Viimeiseksi kartoitetaan ilmaiseksi saatavia staattisen analyysin työkaluja, joilla voitaisiin mahdollisesti täydentää valittujen tai muiden tarkistimien turvallisuutta tai opiskelijalle annettavaa palautetta. Työn tutkimusmenetelmä on kirjallisuustutkimus.

2.1 Tarkistimet

Goblin ja Scheme-robo ovat TKK:n ohjelmointikursseilla käytettyjä järjestelmiä. Valitsin ne lähempään tarkasteluun oman käyttökokemukseni perusteella. Lisäksi ne edustavat kahta hyvin erilaista lähestymistapaa ja antavat siksi tarpeeksi laajan kuvan käsiteltävästä ongelma-alueesta.

2.1.1 EXPACA

Goblin on TKK:lla kehitetty WWW-pohjainen kurssinhallintajärjestelmä. Alunperin se kehitettiin C-kielisten ohjelmointiharjoitusten arvosteluun ja sitä on sittemmin käytetty ainakin C++-, Java- ja XML-harjoitusten arvostelussa. Sen mukana toimitetaan EXPACA-tarkistin, jota hallitaan XML-pohjaisella konfigurointikielellä. EXPACA kutsuu ensin kääntäjää ja suorittaa ohjelmakoodin antamalla sille komentoja virtuaalikonsolilla ja lukemalla sen syötettä. Goblin-järjestelmä ei analysoi koodia muuten kuin kääntämisen muodossa. (Hiisilä, 2005)

2.1.2 Scheme-robo

Scheme-robo on TKK:lla kehitetty Schemellä toteutettu tarkistin, joka pohjautuu metasirkulaariseen tulkkiin (Abelson et al., 1996). Turvallisen hiekkalaatikon luomiseksi Scheme-kielestä on rajattu pois tiedostojen käsittelytoiminnot sekä eval-komento, jolla voisi suorittaa mielivaltaista koodia ja siten vapautua hiekkalaatikosta. Lisäksi tiettyjen

komentojen käyttöä voidaan rajata tehtäväkohtaisesti. (Saikkonen et al., 2001) (Lilja ja Saikkonen, 2003)

Eri lähteissä SCHEME-ROBO määritellään sekä TRAKLA-järjestelmään tukeutuvaksi kokonaiseksi kurssinhallintajärjestelmäksi (Saikkonen et al., 2001) että kotisivullaan tarkistimeksi (Lilja ja Saikkonen, 2003). Tässä työssä viittaan selkeyden vuoksi pelkkään tarkistimeen. TRAKLA on TKK:n tietorakennekurssia varten kehitetty kurssinhallintajärjestelmä (Korhonen ja Malmi, 2000).

Kokonaisten ohjelmien sijaan SCHEME-ROBO ottaa syötteekseen yksittäisiä Scheme-proseduureja, joiden paluuarvon perusteella harjoitukset pisteytetään. Tällä tavoin vältytään palautteen muotoilun vaikutukselta arvosteluun sekä mahdollisilta jäsentelijän (*englanniksi parser*) toteutukseen liittyviltä riskeiltä. SCHEME-ROBO voi myös verrata ohjelman rakennetta opettajan antamaan mallirakenteeseen ja hylätä tehtäviä niistä kutsuttavien komentojen nimien perusteella (Saikkonen et al., 2001). Viimeksi mainitut eivät ole turvallisuusominaisuuksia, mutta niitä voitaisiin käyttää sellaisina pienin muutoksin.

2.2 Yleisimmät turvallisuusriskit

CERT on Yhdysvaltain kansallinen tietoturvaviranomainen, joka tiedottaa ohjelmista löydetyistä haavoittuvuuksista. Merkittävä osuus CERT:n turvallisuussuosituksissa mainituista ongelmista viittaavat puskurin ylivuotoon liittyviin ongelmiin (US-CERT). Tällä tarkoitetaan tilannetta, jossa ennalta määritellyn kokoiseen puskuriin kirjoitetaan enemmän dataa kuin puskurissa on tilaa. Myöhemmässä ohjelman suorituksessa ohjelmaprosessi voidaan kaapata mielivaltaisen koodin suorittamiseen käyttäen puskurin ulkopuolelle kirjoitettua dataa (Tevis ja Hamilton, 2004). Tällainen haavoittuvuus on tyypillistä ohjelmointikielissä, joissa muistin varaaminen ja osoittaminen on ohjelmoijan vastuulla. Nykyaikana tämä koskee lähinnä C- ja C++-kieliä.

Samat kielet ovat semantiikastaan johtuen herkkiä taulukon indeksien käyttämiselle mielivaltaiseeen muistialueeseen viittaamiseen. Esimerkiksi Ada ja Java ratkaisevat tämän ongelman epäsuoralla ajonaikaisella tarkituksella. Lisäksi on monia muita ongelmia, jotka ovat tyypillisiä vain C- ja C++-koodissa. Funktionaalisten ohjelmointikielen suunnittelu on merkittävästi erilaista: Funktioilla ei ole sivuvaikutuksia, indekstointia ei pääosin käytetä eikä muistiin viitata suoraan. (Tevis ja Hamilton, 2004)

Monet dynaamisesti muistia käsittelevät ja tilattomat funktionaaliset kielet eivät määrittele suoraan muistia käsittelevää toiminnallisuutta, joten niissä puskurin ylivuoto ei ole ongelma ellei tulkin tai kääntäjän toteutus ole virheellinen. Dynaamisia kieliä ovat esimerkiksi Python ja Scheme. Scheme ja Haskell ovat puolestaan funktionaalisia kieliä.

Toinen tapa haitata järjestelmän toimintaa on palvelunesto. Sillä tarkoitetaan järjestel-

män vakauden horjuttamista siten, että sen toiminta hidastuu merkittävästi tai estyy kokonaan. Tarkistimet varautuvat yleensä näihin ongelmiin rajaamalla muistinkäyttöä ja rajaamalla suoritusajan kohtuulliseen ylärajaan.

On olemassa myös monia muita mahdollisia hyökkäyksiä, kuten dynaamisen linkkerin ohjaaminen haavoittuvaan kirjastoon, DNS-väärennökset ja UNIX-signaalien käyttö (Tevis ja Hamilton, 2004). Tällaiset hyökkäykset eivät kuitenkaan ole olennaisia tarkistimien tapauksessa, vaikka ne tuleekin huomioida suuremman kurssinhallintajärjestelmän suunnittelussa.

2.3 Staattinen analyysi

Ohjelmakoodin staattiseen analyysiin liittyy useita vaikeasti ratkeavia ongelmia. Esimerkiksi ei ole olemassa tunnettua yleisesti pätevää tapaa osoittaa ohjelmallisesti, päättyykö ohjelman suoritus koskaan. Kuitenkin automaattisesti tarkistettavat ohjelmointiharjoitukset ovat yleensä ratkaisuja hyvin rajattuun ja tunnettuun ohjelmointiongelmaan. Lisäksi ohjelman pitkäksi venyvä suoritusaika on riittävä peruste ratkaisun hylkäämiselle.

Monet ohjelmakoodin staattiseen analyysiin liittyvät ongelmat ovat ihmiselle helppoja. Osa ohjelmakoodia tarkastelevista työkaluista etsii epäilyttäviä kohtia ja antaa ne ihmisen tutkittavaksi. (Taft, 2008) Puoliautomaattiset menetelmät eivät kuitenkaan kuulu tämän työn tutkimusalueeseen, koska ne estävät automaattisesta tarkistamisesta saatavan nopean palautteen edun ja ovat työläitä erityisesti suurilla kursseilla.

Koodin turvallisuutta analysoivia työkaluja on tarjolla hyvin paljon C- ja C++-ohjelmille. Yleisimpiä työkalujen havaitsemia ongelmia ovat puskurin ylivuodot, nollaosoittimeen viittaaminen ja taulukoiden indekseihin liittyvät ongelmat. Näiden syy voi olla joko tahallinen tai tahaton. Tevis ja Hamilton (2004) sekä Heffley ja Meunier (2004) tutkivat ryhmää staattisia analysaattoreita. Työkaluja on tarjolla kymmeniä, joista osa on kaupallisia ja monet saatavilla ilmaiseksi. Molemmissa tutkimuksissa havaittiin, että työkalut antavat huomattavan määrän vääriä hälytyksiä ja niiden joukosta oli vaikeaa löytää oikeita ongelmia.

Tahallisten hyökkäysten estämiseksi yksinkertaisin keino on käyttää tekstihakua havaitsemaan komentoja, joita hyvissä aikeissa olevan opiskelijan ei tulisi käyttää. Esimerkiksi Scheme-robo rajoittaa Schemen kielioppia nimenomaan turvallisuuteen perustuen.

Toisaalta matalan tason kielillä voidaan suorittaa koodia tavoilla, jotka eivät paljastu tekstihaulla. Funktiohin voidaan viitata nimien sijasta osoitteilla, funktion nimi esittää ohjelmakoodin joukossa sotkettuna tai kutsua vaihtoehtoista toteutusta. Näitä tekniikoita käytetään muun muassa virusten aikeiden peittelyyn, jotta ne eivät jäisi kiinni virustorjunnassa. Moser et al. (2007) esittelevät tutkimuksessaan monimutkaistustekniikoita

(*englanniksi obfuscation*), joilla staatinen analysaattori voidaan ohjata ratkaisemaan NP-täydellisiä ongelmia. Tutkimus käsittelee staattisen analyysin harhauttamiseen toimivia tekniikoita virustorjuntaohjelmistojen perspektiivistä.

Vaikeisiin ongelmiin perustuva monimutkaistaminen ei kuitenkaan riipu käytettävästä kielestä, jolloin ne toimivat yhtä hyvin esimerkiksi C-kielellä kirjoitettujen ohjelmien analyysin estämiseen. Toisaalta ohjelmointiharjoitusten tapauksessa myös staattisen analyysin suoritusaikaa voidaan käyttää tehtävän hylkäysperusteena, jolloin selkeät harhautusyritykset tarkistetaan manuaalisesti.

Ohjelmoinnin peruskursseilla tällaiset haasteet ovat kuitenkin lähinnä teoreettisia. On todennäköistä, että suurin osa opiskelijoista vasta opettelee ohjelmointia, kotitehtävävastaukset lähetetään pääasiassa omilla tunnuksilla kirjautuneena ja monimutkaistetun haittaohjelman tekeminen on monimutkaisempaa kuin tehtävänannon seuraaminen. On kuitenkin hyvä tiedostaa, että staattinen analyysi ei riitä ainoaksi työkaluksi turvallisuuden parantamisessa.

2.4 Staattisen analyysin työkalut

Heffley ja Meunier (2004) tutkivat ohjelmien haavoittuvuuksia etsiviä työkaluja. He havaitsivat, että monet niistä tuottavat niin paljon vääriä varoituksia, etteivät ne ole käytännöllisiä. Työkalujen joukosta kuitenkin erottui yksi hyödyllinen työkalu, Pscan, joka onnistui antamaan luotettavia tuloksia rajatulla alueella. Lisäksi Tevis ja Hamilton (2004) kartoittavat joukkoa työkaluja, joiden joukossa on muutama vähän vääriä hälytyksiä tuottava työkalu. Hekin mainitsivat Pscanin ja lisäksi Bell Labsilla kehitetyn UNO:n.

Työkalujen joukossa oli muun muassa koodin joukkoon tehtäviin merkintöihin pohjautuva Splint (Tevis ja Hamilton, 2004), jonka käyttö automaation osana saattaisi olla hankalaa. Virhehavaintojenkin määrä oli suuri. Tästä huolimatta sillä voisi olla arvoa opiskelijoiden oppimisen tukena.

2.4.1 Pscan

Pscan on avoimen lähdekoodin työkalu, jolla voidaan etsiä riskialttiita printf-tyyppisten funktioden kutsuja lähdekoodista. Heffley ja Meunier (2004) totesivat tutkimuksessaan työkalun olevan toiminnaltaan hyvin rajattu ja siksi se tuottaa vertailujoukossaan vähiten vääriä hälytyksiä. Tutkituissa tapauksissa kaikki löydäkset eivät olleet hyödynnettävissä hyökkäyksessä, mutta osoittivat kuitenkin vaarallista ohjelmointitapaa. Toisaalta myös ?-operaattori esti ohjelmaa huomaamasta oikeita haavoittuvuuksia. Pscan etsii seuraavanlaisia koodirivejä:

```
sprintf(puskuri, muuttuja);         // Varoitus
sprintf(puskuri, "%s", muuttuja);   // Oikein
sprintf(puskuri, totuusarvo ? muuttuja1 : muuttuja2)
```

Mikäli ensimmäisen tapauksen muuttuja tulee tarkistamattomana vihamieliseltä käyttäjältä, voi siihen sisällyttää muotoilumerkintöjä. Ne voidaan mahdollisesti korvata käyttäjän määräämillä arvoilla ohjelman myöhemmässä vaiheessa. Jälkimmäisessä tapauksessa näin ei tapahdu. Viimeinen rivi on esimerkki virheestä, jota Pscan ei huomaa. (Heffley ja Meunier, 2004)

2.4.2 UNO

UNO on Bell Labsissa kehitetty työkalu, joka keskittyy nimensä mukaisesti kolmen ongelman havaitsemiseen: alustamattoman muuttujan käyttö (Use of uninitiated variable), nollaosoittimeen viittaaminen (Nil-pointer references) ja taulukon ulkopuolinen indeksointi (Out-of-bounds array indexing). Tekijänsä Holzmannin mukaan sen suunnittelulähtökohtana on ollut korkea signaali–kohinasuhde. Tällä tarkoitetaan hyödyllisten varoitusten määrän maksimointia. Tämän lisäksi UNO on helposti laajennettavissa. (Tevis ja Hamilton, 2004) (Holzmann, 2002)

3 Tulokset

3.1 Scheme-robon ja EXPACA:n turvallisuusuhat

Kirjallisuustutkimuksen perusteella SCHEME-ROBO on turvallisuusratkaisuiltaan onnistunut tarkistin. Yksi osasyy on se, ettei Schemeen liittyviä turvallisuusuhkia ei ole tutkittu aktiivisesti. Kieli on pääosin akateemisessa käytössä eikä ole siksi hyökkääjien kannalta mielenkiintoinen. Toisaalta kielen funktionaalisuuden takia se ei myöskään määrittele monia yleisemmin käytössä oleville kielille tyypillisiä ongelmia. Toisaalta SCHEME-ROBOn toteutuksessa on huomioitu eval-lauseeseen ja tiedostonkäsittelyyn liittyvät ongelmat, joiden kautta rajatun tulkin turvallisuus voitaisiin ohittaa helposti. Schemen vahvalla metasirkulaaristen tulkkien tuella on tässä merkittävä vaikutus.

EXPACA ei analysoi koodia staattisesti kääntäjää lukuunottamatta. Järjestelmä ei sellaisenaan tue ulkoisia työkaluja. Hiisilän mukaan esimerkiksi Valgrindilla tuotetulla palautteella voisi kuitenkin olla opiskelijoiden oppimisen kannalta arvoa. EXPACA-tarkistin on toteutettu C++-kielellä, mutta Hiisilä erikseen mainitse, että turvallinen ohjelmointitapa olisi ollut tärkeä tekijä sen suunnittelussa. Turvallisuusnäkökohtana hän toteaa, että EXPACA voidaan suorittaa erillisessä hakemistorakenteessa. (Hiisilä, 2005) Tämä tarkoit-

tanee chroot-ympäristöä, joista karkaamisen on todettu olevan helppoa (Simes, 2002). Toisaalta chroot-hakemistorakenne on pienellä vaivalla siirrettävissä virtualisoituun tai emuloituun ympäristöön.

Goblin-järjestelmä ei modulaarisella suunnittelullaan ole herkkä hyökkäyksille, mutta sillä ajettavat harjoitustehtävät on kirjoitettu täysimittaisilla ohjelmointikielillä. Niiden toiminnallisuutta ei ole rajattu, mikä voi aiheuttaa turvallisuusongelmia. Tästä johtuen ohjelman suoritus täytyy eristää riittävästi muusta ympäristöstä.

3.2 Analyysityökalujen käytettävyys

Pscan ja UNO vaikuttavat molemmat lupaavilta työkaluilta. Niiden havaitsema ongelma-alue on hyvin rajattu ja niiden antamat tulokset ovat keskimääräistä luotettavampia. Tämän lisäksi UNO on helposti laajennettavissa. Toisaalta laajentamalla saatetaan aiheuttaa vain lisää vääriä hälytyksiä. Molemmat ovat saatavissa lähdekoodeineen ilmaiseksi, mikä helpottaa niiden käyttöönottoa.

Tämän työn puitteissa kumpaakaan työkalua ei ole kokeiltu käytännössä, joten myöskään niiden käytännöllisyydestä kotitehtävätarkistimessa ei ole riittävää näyttöä. Toisaalta työkalut ovat yksinkertaisia käyttää ja paljastavat yleisiä ja vakavia ongelmia. Vaikka ne eivät sopisikaan tarkistimen kanssa käytettäviksi, niiden käytön opettaminen ohjelmoinnin peruskursseilla saattaisi auttaa opiskelijoita ymmärtämään turvallista ohjelmointitapaa ja turvallisuuden merkitystä opintojen alusta vaiheesta alkaen.

Koska molemmat ohjelmat ovat komentorivityökaluja, niitä pitäisi voida käyttää myös rinnakkain. Työkalujen havainnot eivät ole päällekäisiä, joten niitä voidaan käyttää täydentämään toistensa tuloksia. Näillä työkaluilla voidaan varmistaa, että niiden tunnistamia virheitä sisältävää koodia ei koskaan ajeta tarkistimessa. Tällä tavalla ne tekevät tarkistimesta hieman turvallisemman, vaikka eivät takaakaan täydellistä tulosta.

3.3 EXPACA:n kehitysmahdollisuudet

Lisäämällä EXPACA:an mahdollisuus ajaa ulkoisia työkaluja ennen koodin kääntämistä, voitaisiin koodille tehdä turvallisuusanalyysi ennen sen suorittamista. Kirjallisuuden perusteella analyysi voitaisiin tehdä Pscanilla, UNO:lla tai molemmilla peräkkäin. Tällöin olisi kuitenkin huomioitava, että analyysin päättymisestä ei voida olla varmoja. Toisaalta EXPACA osaa havaita koodin liian pitkän suoritusajan. Samaa koodia voitaisiin käyttää uudelleen analysaattoritukea toteutettaessa.

Sekä Pscan että UNO tulostavat ohjelmasta löytyvät virheet virtuaalikonsolille, jolloin niiden antamalla palautteella voitaisiin helposti vaikuttaa arvosteluun ja palautteena saa-

tu tulos voitaisiin ohjata suoraan opiskelijalle. Teoriassa helpoin tapa toteuttaa tällainen toiminnallisuus olisi muuttaa EXPACA:n konfiguraatiota siten, että käännetyn ohjelman sijasta ajettaisiin komentojono, joka analysoi lähdekoodin ja suorittaa vasta sen jälkeen itse ohjelman. Goblinin pisteytys perustuu ohjelman tulosteeseen. Analysaattorilta tuleva odottamaton tuloste ja odotetun syötteen puuttuminen kokonaan johtaisi tehtävän hylkäämiseen.

Tutkimuksessa ei myöskään löytynyt viitteitä siihen, että EXPACA:n turvallisuutta olisi erikseen tutkittu. Jo pelkkä ohjelmakoodin katselmointi saattaisi antaa käsityksen turvallisuuteen panostamisen tarpeellisuudesta. Katselmoinnin apuna voitaisiin käyttää tässä työssä mainittuja staattisia analysaattoreita. Niillä voisi arvioida tarkistimen turvallisuutta ja samalla tutkia ovatko ne tarpeeksi luotettavia järjestelmän osaksi.

3.4 Ohjelmistonkehitystavan vaikutus tarkistimiin

Hiisilä (2005) mainitsee diplomityönsä monessa vaiheessa, että jokin ominaisuus on jätetty toteuttamatta ajanpuutteen vuoksi. Tarkistimen lisäksi järjestelmään kuuluu muita osia ja ajatuksia muuhun kuin tarkistimen parantamiseen on runsaasti. Scheme-robossa ei ole ollenkaan kurssinhallintatoimintoja, jotka ovat tarpeellisia kurssin järjestämiseksi.

Ohjelmoinnin massakurssien järjestämiseksi tarvitaan muitakin järjestelmiä kuin kotitehtävätarkistin ja kurssihenkilökunta toteuttaa tarvittavia työkaluja työn ohessa eikä ajanpuutteesta johtuen tarkistimien turvallisuuteen ehditä paneutua. Riskeihin ei olla välttämättä perehdytty tarkasti eivätkä ne välttämättä ole koskaan realisoituneet.

Kotitehtävien automaattinen tarkistaminen on laaja ongelmakenttä, jota ainakin TKK:lla tehdään muun työn ohessa. Tästä huolimatta esimerkiksi puskurin ylivuodot ovat yhtälailla yleisiä ongelmia myös kaupallisissa sovelluksissa, joiden kehittämiseen osallistuu jopa satoja päätoimisia kehittäjiä ja testaajia. Toisaalta Hiisilä (2005) toteaa, että työkalun valintaan vaikutti kaupallisten tarkistintyökalujen heikko mukautettavuus.

4 Johtopäätökset

Automaattinen arvostelu on monimuotoinen ongelmakenttä, jossa turvallisuus on vain yksi sivupolku. Monet olemassaolevat järjestelmät ovat kurssihenkilökunnan muun työn ohessa toteuttamia. Luotettava tarkistin tarvitsee ympärilleen myös muita kurssinhallintatyökaluja, jolloin eheän kurssikokonaisuuden järjestäminen on työkalun elinkaaren alussa tärkeämpää kuin turvallisuuden yksityiskohtien hiominen. Esimerkiksi Goblinia käsittelevässä diplomityössä todetaan toistuvasti, että jotakin tiettyä muuta näkökulmaa ei työn puitteissa ehditty tutkimaan. Tästä syystä turvallisuutta on tehokkainta parantaa

käyttäen helposti saatavia ja valmiita työkaluja.

Ohjelmointiharjoitusten automaattiseen arvosteluun liittyvät turvallisuusseikat ovat vaikeasti todettavia ja väärien hälytysten riski on suuri. Monien staattisen analyysien luotettavuus on automaattisen arvostelun tarpeisiin nähden riittämätön. Monet työkalut luottavat siihen, että löydökset käydään manuaalisesti läpi, mikä ei massakursseilla onnistu. Tästä johtuen on tärkeää, että tarkistimen toteutus katselmoidaan ja analysoidaan tarkasti tietoturvaongelmien varalta. Lisäksi se tulee eristää muista järjestelmistä ja erityisesti kurssinhallintajärjestelmästä, jottei onnistuneellakaan hyökkäyksellä voida vaikuttaa arvosteluun tai kurssin järjestelyihin. Tässä korostuu esimerkiksi järjestelmän hajauttamisen tai virtualisoinnin tärkeys. Tehtävien tarkistaminen emulaattorissa on mielenkiintoinen aihe jatkotutkimukselle.

Toisaalta jo rajallisiakin haavoittuvuuksia havainnoivilla ohjelmilla voi olla suuri vaikutus etenkin tahattomista virheistä johtuvissa tilanteissa. Jos puskurin yli vuotavaa ohjelmaa ei koskaan suoriteta, vältetään mahdollisia ongelmia. Ongelman staattisessa analyysissä havaitsevat työkalut myös tarjoavat opiskelijalle aiheellista palautetta turvallisen ohjelmointitavan noudattamisesta. Selkeä palaute on arvokkaampaa, kuin ongelman realisoituessa saatu virheilmoitus. Esimerkiksi UNIX-ympäristössä saatu ”Segmentation fault-virhe ei välttämättä auta vian löytämisessä ollenkaan, kun taas lähdekoodia analysoimalla vika voidaan paikallistaa oikealle riville. Oman kokemukseni mukaan ainakaan TKK:n tietotekniikan opinto-ohjelmaan ei ole kuulunut lainkaan staattisen analyysin työkaluja. Tällöin pitäisi kuitenkin arvioida työkaluja toisesta perspektiivistä, mikä on oivallinen aihe jatkotutkimukselle.

Ohjelmointiharjoituksissa käytettävällä kielellä on suuri vaikutus turvallisuuteen. Esimerkiksi TKK:lla aiemmin järjestetyllä Scheme-ohjelmointikurssilla käytetty Scheme-robo rajaa kielen mahdollisesti vaaralliset ominaisuudet työkalun ulkopuolelle onnistuneesti. Tällä tavoin saadaan eliminoitua kokonaisia ongelmakategorioita. Tämän toteuttaminen ei kuitenkaan ole ominaista kaikille ohjelmointikielille, vaan edellyttää monille funktionaalisille kielille tyypillisiä tulkin kirjoittamiseen tarkoitettuja ominaisuuksia. Turvallisen rajoitetun C-tulkin tai -kääntäjän toteuttaminen on mielenkiintoinen aihe jatkotutkimukselle. Samoin Scheme-robossa oleva ohjelman mallirakenteen varmistavan työkalun toteuttaminen esimerkiksi C:lle tai Javalle on haastava ja mielenkiintoinen tutkimuskohde.

Kielen valintaan liittyvät turvallisuusnäkökulmat on hyödyllistä huomioida automaattista arvostelua harkitsevan kurssin suunnittelussa. Onnistunut valinta voi turvallisempien tarkistimien lisäksi johtaa opiskelijoiden turvallisuustietämyksen lisääntymiseen.

Lähteet

H. Abelson, G.J. Sussman, J. Sussman ja A.J. Perlis. *Structure and interpretation of computer programs*. Mit Press Cambridge, MA, 1996. ISBN 0262011530.

J. Carter, K. Ala-Mutka, U. Fuller, M. Dick, J. English, W. Fone ja J. Sheard. How shall we assess this? *ACM SIGCSE Bulletin*, 35(4):107–123, 2003. ISSN 0097-8418.

D. Hagan ja S. Markham. Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin*, 32(3):25–28, 2000. ISSN 0097-8418.

J. Heffley ja P. Meunier. Can source code auditing software identify common vulnerabilities and be used to evaluate software security? 2004. ISSN 1530-1605.

A. Hiisilä. Kurssinhallintajärjestelmä ohjelmoinnin perusopetuksen avuksi. Diplomityö, Teknillinen korkeakoulu, Espoo, 2005. Saatavissa `http://goblin.hut.fi/goblin/diplomityo.pdf`. Viitattu 25.4.2011.

G. Holzmann. UNO: Static source code checking for user-defined properties. *World Conf. on Integrated Design and Process Technology (IDPT)*. Citeseer, 2002.

V.-J. Kesti. Virtualisointiratkaisun valinta haittaohjelmatutkimukseen. Kandidaatintyö, Aalto yliopisto, Espoo, 2010.

A. Korhonen ja L. Malmi. Algorithm simulation with automatic assessment. *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, sivut 160–163. ACM, 2000. ISBN 1581132077.

T. Lilja ja R. Saikkonen. Scheme-robo manual, 2003. `http://www.cs.hut.fi/Research/Scheme-robo/`. Scheme-robon WWW-sivu. Viitattu 25.4.2011.

A. Moser, C. Kruegel ja E. Kirda. Limits of static analysis for malware detection. *acsac*, sivut 421–430. IEEE Computer Society, 2007.

R. Saikkonen, L. Malmi ja A. Korhonen. Fully automatic assessment of programming exercises. *ACM SIGCSE Bulletin*, osa 33, sivut 133–136. ACM, 2001. ISBN 1581133308.

Simes. How to break out of a chroot() jail, 2002. `http://www.bpfh.net/simes/computing/chroot-break.html`. Viitattu 25.4.2011.

S. T. Taft. Systematic Scanning for Malicious Source Code. *Technologies for Homeland Security, 2008 IEEE Conference on*, sivut 173–175. IEEE, 2008.

J.E.J. Tevis ja J.A. Hamilton. Methods for the prevention, detection and removal of software security vulnerabilities. *Proceedings of the 42nd annual Southeast regional conference*, sivut 197–202. ACM, 2004. ISBN 1581138709.

US-CERT. Technical cyber security alerts. `http://www.us-cert.gov/cas/techalerts/index.html` Viitattu 4.5.2011.