Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi

Kandidaatintyö

4. toukokuuta 2011

Ferrix Hovi

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Ferrix Hovi
Työn nimi:	Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi
Päiväys:	4. toukokuuta 2011
Sivumäärä:	11
Pääaine:	Ohjelmistotuotanto ja -liiketoiminta
Koodi:	T3003
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	TkT Ari Korhonen (Tietotekniikan laitos)

Tämä on TIK.kand-kurssin LaTeX-pohja, jota voi vapaasti käyttää. Koko zip-paketin voi ladata kurssin Noppa-sivulta `https://noppa.tkk.fi/noppa/kurssi/TIK.kand/materiaali/`. Mukana on esimerkkejä LaTeX:n käytöstä.

Teksti on peräisin TIK.kand-kurssin historiallisesta LaTeX-pohjasta, jota kurssin koordinaattori Jukka Parviainen päivitti tammikuussa 2011. Lisäksi suomen kielen lehtori Sanni Heintzmann kirjoitti rakenteellisia vinkkejä.

Tiivistelmä on muusta työstä täysin irrallinen teksti, joka kirjoitetaan tiivistelmälomakkeelle vasta, kun koko työ on valmis. Se on suppea ja itsenäinen teksti, joka kuvaa olennaisen opinnäytteen sisällöstä. Tavoitteena selvittää työn merkitys lukijalle ja antaa yleiskuva työstä. Tiivistelmä markkinoi työtäsi potentiaalisille lukijoille, siksi tutkimusongelman ja tärkeimmät tulokset kannattaa kertoa selkeästi ja napakasti. Tiivistelmä kirjoitetaan hieman yleistajuisemmin kuin itse työ, koska teksti palvelee tiedonvälitystarkoituksessa laajaa yleisöä.

Avainsanat:	avain, sanoja, niitäkin, tähän, vielä, useampi, vaikkei, niitä, niin, montaa, oikeasti, tarvitse
Kieli:	Suomi

Aalto University
School of Science
Degree Program of Computer Science and Engineering

ABSTRACT OF
BACHELOR'S THESIS

Author:	Ferrix Hovi
Title of thesis:	Automatic Security Assessment of Programming Assignments
Date:	May 4, 2011
Pages:	11
Major:	Ohjelmistotuotanto ja -liiketoiminta
Code:	T3003
Supervisor:	Professor (tem) Tomi Janhunen
Instructor:	D.Sc. (Tech) Ari Korhonen (Department of Computer Science Engineering)
Here goes the abstract (english)	
Keywords:	key, words, the same as in FIN/SWE
Language:	Finnish

Sisältö

1 Johdanto

Ohjelmoinnilla on olennainen osa tietotekniikan opetuksessa ja ohjelmoinnin opiskelussa käytännön harjoittelulla on merkittävä rooli. Ripeästi saatavalla palautteella on oppimista tehostava vaikutus. Kurssit ovat monesti, esimerkiksi TKK:lla osallistujamääriltään suuria, jolloin tehtävien arvosteleminen tehokkaasti käsin esimerkiksi kokonaisen vuosikurssin kokoisilla kursseilla on käytännössä mahdotonta. Automaattisen arvostelun edut ovat myös merkittävimpiä juuri opintojen alussa, jolloin sovelletaan vähemmän (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, joka pitää ottaa huomioon myös ohjelmointiharjoitusten tarkistimissa. Parhaassa tapauksessa riittävän tarkasti havaittu turvallisuusriski tarjoaa opiskelijalle mahdollisuuden ymmärtää ja oppia tekemästään virheestä turvallisessa ympäristössä ennen työelämään siirtymistä. Pahimmillaan opiskelijan tahaton virhe johtaa koko järjestelmän virhetilaan.

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.

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 TKK:lla käytettyä tarkistinta ja vertaillaan niiden toteutuksista johtuvia turvallisuusominaisuuksia. Tämän jälkeen yritetään löytää muutamia järjestelmien vaatimuksiin sopivia ja luotettavia staattisen analyysin työkaluja. Järjestelmän olennaisia vaatimuksia ovat nopea palaute, väärien hälytysten määrän minimointi ja käytettävyys. Menetelmät eivät 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 autentikoinnista ja muista

kurssin järjestelyihin liittyvistä asioista. Tässä työssä keskitytään ainoastaan tarkistimen toimintaan ja turvallisuuteen.

Väärä hälytys tarkoittaa sellaista virhehavaintoa, joka johtaa opiskelijan antaman harmittoman syötteen hylkäämiseen. Tämä johtaa käytännössä arvostelun epäluotettavuuteen ja mahdollisesti jopa hyvien suoritusten virheelliseen hylkäämiseen. Scheme-robon tilastojen perusteella opiskelijat palauttavat harjoitustöitään juuri ennen määräaikaa (Saikkonen et al., 2001). Jos järjestelmä antaa paljon vääriä hälytyksiä viime hetkillä, saattaa opiskelijoiden usko järjestelmän oikeudenmukaisuuteen horjua. Tämä saattaa aiheuttaa kurssihenkilökunnalle lisätyötä.

Käytettävyydellä tarkoitetaan tässä työssä järjestelmän käyttäjälle tulevaa tuntumaa ohjelman käytön helppoudesta ja sujuvuudesta. Pyrkimyksenä on, ettei loppukäyttäjälle aiheudu turvallisuuden lisäämisestä minkäänlaista näkyvää muutosta järjestelmän toiminnassa poislukien kohtuullinen lisäys arvosteluun kuluvaan aikaan.

Nopealla palautteella tarkoitetaan yksittäisen harjoituspalautuksen tarkistamiseen kuluvaa läpimenoaikaa. Läpimenoajan tulisi olla sellainen, että opiskelija pystyy tarkastuttamaan harjoituksensa järjestelmässä ilman työnkulun katkaisevaa keskeytystä.

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.

Toteutuksen turvallisuudella tarkoitetaan ohjelmakoodin turvallista toteutusta, eristämistä muista järjestelmistä ja muita teknisiä valintoja. Tarkistimen pitää olla turvallisesti toteutettu, jotta sen antamat pisteet ja palaute tulevat varmasti tarkistimelta itseltään eikä esimerkiksi opiskelijan koodista. Tarkistimen tulee olla eristetty muusta järjestelmästä, jotta ohjelmointivirhe tai hyökkäys ei aiheuta häiriötä muussa järjestelmässä tai vaikuta esimerkiksi arvosanatietojen eheyteen. Muita teknisiä valintoja ovat esimerkiksi käytettyjen kirjastojen virhealttius, arkkitehtuurin luotettavuus ja käytetyn ohjelmointikielen ominaisuudet.

1.1.2 Riskianalyysi

Tämä työ olettaa tarkistimeen kohdistuvien riskien olevan pääasiassa tahattomia ohjelmointivirheitä. Todelliset riskit ovat varmasti monitahoisempia, mutta parempi riskimalli edellyttää syvällistä tutkimusta, joka ei kuulu tämän työn alueeseen.

Automaattinen tarkistus edellyttää, että ongelmat ovat tarkasti määriteltyjä, jolloin nii-

den hyödyt ovat pienempiä kursseilla, joilla edellytetään enemmän soveltamista. Tämä tarkoittaa käytännössä ensimmäisiä peruskursseja, joilla opiskelijoilta ei odoteta aiempaa ohjelmointikokemusta. Kursseilla voi toki olla kokeneempia ohjelmoijia, mutta he todennäköisesti menestyvät kurssilla keskimääräistä paremmin (Hagan ja Markham, 2000). Opiskelijoilla ei tämän perusteella pitäisi olla syitä tai tarpeellisia kykyjä vilppiin. Lisäksi pyrkimys läpäistä kurssi todennäköisesti vähentää motivaatiota häiritä järjestelmää tahallisesti.

1.2 Rajaus

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 lähdekoodin 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. Tämä työ ei kuitenkaan etsi ratkaisua näihin ongelmiin. (Carter et al., 2003)

Koodin ajaminen 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 TKK:lla käytettyyn tarkistusjärjestelmään: Scheme-robOon ja Gobliniin. Järjestelmiä tarkastellaan niistä kirjoitetun tieteellisen kirjallisuuden avulla.

Tarkistimien lisäksi tutustutaan yleisimpiin ohjelmistoihin liittyviin turvallisuusuhkiin ja esimerkiksi kurssilla opetettavan ohjelmointikielen vaikutukseen.

Viimeiseksi kartoitetaan ilmaiseksi saatavia staattisen analyysin työkaluja, joilla voitaisiin mahdollisesti täydentää valittujen tai muiden tarkistimien turvallisuutta tai opiskelijalle annettavaa palautetta.

2.1 Tarkistimet

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

2.1.1 Goblin

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. (Hiisilä, 2005)

Hiisilä (2005) mainitsee diplomityössään, ettei Goblin analysoi koodia muuten kuin kääntämisen muodossa.

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 (Saikkonen et al., 2001) sekä eval-komento, jolla voisi suorittaa mielivaltaista koodia ja siten vapautua hiekkalaatikosta. Lisäksi tiettyjen komentojen käyttöä voidaan rajata tehtäväkohtaisesti. (Lilja ja Saikkonen, 2003)

Eri lähteissä Scheme-robo määritellään sekä TRAKLA-järjestelmään (Korhonen ja Malmi, 2000) tukeutuvaksi kokonaiseksi kurssinhallintajärjestelmäksi (Saikkonen et al., 2001) että kotisivullaan tarkistimeksi (Lilja ja Saikkonen, 2003). Tässä työssä viittaan selkeyden vuoksi pelkkään tarkistimeen.

Kokonaisten ohjelmien sijaan Scheme-robo ottaa syötteekseen yksittäisiä Scheme-funktioita, joiden paluuarvon perusteella harjoitukset pisteytetään. Tällä tavoin vältytään palautteen muotoilun vaikutukselta arvosteluun sekä mahdollisilta parserin toteutukseen liittyviltä riskeiltä. (Saikkonen et al., 2001)

Scheme-robo voi myös verrata ohjelman rakennetta opettajan antamaan antamaan malliirakenteeseen ja hylätä tehtäviä niistä kutsuttavien komentojen nimien perusteella. (Saikkonen et al., 2001) Nämä eivät sinänsä ole turvallisuusominaisuuksia, mutta niitä voitaisiin käyttää myös sellaisina pienin muutoksin.

2.2 Yleisimmät turvallisuusriskit

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

Tevis ja Hamilton (2004) kertovat myös, että samojen kielien semantiikasta johtuen ne ovat herkkiä taulukon indeksien käyttämiselle mielivaltaiseen muistialueeseen viittaamiseen (engl. out-of-bounds array indexing). Esimerkiksi Ada ja Java ratkaisevat tämän ongelman implisiittisellä ajonaikaisella tarkistuksella. Samassa tutkimuksessa kerrotaan monista ongelmista, jotka ovat tyypillisiä vain C- ja C++-koodissa. Tevis ja Hamilton toteavat myös, että funktionaalisten ohjelmointikielien suunnittelu on merkittävästi erilaista. Funktioilla ei ole sivuvaikutuksia, indeksointia ei pääosin käytetä eikä muistiin viitata suoraan.

Monet dynaamisesti muistia käsittelevät ja tilattomat funktionaaliset kielet eivät määrittele suoraan muistia käsittelevää toiminnallisuutta, joten niissä puskurin ylivuoto ei ole määritelty 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 toimintaan on palvelunesto, jolla tarkoitetaan järjestelmän vakauden horjuttamista siten, että sen toiminta hidastuu merkittävästi tai estyy kokonaan. Tarkistimet varautuvat yleensä näihin ongelmiin rajaamalla muistinkäyttöä ja rajaamalla suoritusajan kohtuulliseen maksimiin.

On olemassa myös monia muita mahdollisia hyökkäyksiä, kuten dynaamisen linkkerin ohjaamisen haavoittuvaan kirjastoon, DNS-hyökkäyksen ja signaalien käytön Tevis ja Hamilton (2004). Tällaiset hyökkäykset eivät kuitenkaan ole olennaisia tarkistimien tapauksessa, vaikka ne voikin olla hyvä 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. Automaattisesti tarkistettavat ohjelmointiharjoitukset ovat kuitenkin yleensä ratkaisuja hyvin rajattuun ja tunnettuun ohjelmointiongelmaan. 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) tutkivat ryhmää

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. Näitä tekniikoita käytetään muun muassa virusten aikeiden peittelyyn, jotta ne eivät jäisi kiinni virustorjunnassa. Moser et al. (2007) esittelevät tutkimuksessaan monimutkaistustekniikoita (engl. 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 aivan yhtä hyvin esimerkiksi C-kielellä kirjoitettujen ohjelmien analyysin estämiseen. Toisaalta ohjelmointiharjoitusten tapauksessa myös staattisen analyysin suoritusaikaa voidaan käyttää tehtävän hylkäysperusteena, joten selkeät harhautusyritykset voidaan tarkistaa esimerkiksi 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 varmasti monimutkaisempaa kuin tehtävänannon seuraaminen. On kuitenkin hyvä tiedostaa, että staattinen analyysi ei ehdoitta riitä ainoaksi työkaluksi.

2.4 Staattisen analyysin työkalut

Heffley ja Meunier (2004) tutkivat ohjelmien haavoittuvuuksia etsiviä työkaluja ja havaitsivat, että monet niistä tuottavat niin paljon vääriä varoituksia, etteivät ne ole käytännöl-

lisiä. 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.

Työkalujen joukossa oli muun muassa koodin joukkoon tehtäviin merkintöihin pohjautuva Splint (Tevis ja Hamilton, 2004), jonka käyttö automaation osana saattaisi olla hankalaa ja 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. Työkalu oli Heffley ja Meunier (2004) tutkimuksen perusteella toiminnaltaan hyvin rajattu ja siitä syystä 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.

sprintf(buffer, variable); Bad! Possible security breach! sprintf(buffer, ”

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 ulkopuoliseen indeksointiin (Out-of-bounds array indexing). Sen suunnittelulähtökohtana on ollut korkea signaali–kohinasuhde. Tämän lisäksi UNO on myös helposti laajennettavissa. (Tevis ja Hamilton, 2004) (Holzmann, 2002)

3 Tulokset

3.1 Scheme-robon ja Goblinin turvallisuusuhat

Kirjallisuustutkimuksen perusteella Scheme-robo on turvallisuusratkaisuiltaan onnistunut tarkistin. Yksi osasyy on se, ettei Schemeen liittyviä turvallisuusuhkia ei ole tutkittu aktiivisesti, koska 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 on-

gelmat, joiden kautta rajatun tulkin turvallisuus voitaisiin ohittaa helposti. Schemen vahvalla metasirkulaaristen tulkkien tuella on tässä merkittävä vaikutus.

Hiisilä (2005) mainitsee diplomityössään, ettei Goblin analysoi koodia staattisesti kääntäjää lukuunottamatta. Hän toteaa, että järjestelmä ei sellaisenaan tue ulkoisia työkaluja. Samalla hän toteaa, että esimerkiksi Valgrindilla tuotetulla palautteella voisi olla opiskelijoiden oppimisen kannalta arvoa. Hiisilän mukaan EXPACA-tarkistin on toteutettu C++-kielellä, mutta ei erikseen mainitse, että turvallinen ohjelmointitapa olisi ollut tärkeä tekijä sen suunnittelussa. Turvallisuusnäkökohtana hän mainitsee, että EXPACA voidaan suorittaa erillisessä hakemistorakenteessa. Tämä tarkoittanee chroot-ympäristöä, joista karkaamisen on todettu olevan helppoa (Simes, 2002). Toisaalta chroot-hakemistorakenne on pienellä vaivalla siirrettävissä virtualisoituun tai emuloituun ympäristöön.

Goblin ei modulaarisella suunnittelullaan ole erityisen herkkä hyökkäyksille, mutta sen ongelmat liittyvät siihen, että sillä ajettavat harjoitustehtävät on kirjoitettu täysimittaisella ohjelmointikielellä, joiden toiminnallisuutta ei ole millään tavalla rajoitettu. 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. Tästä syystä jo pelkästään niihin viittaaminen ohjelmoinnin peruskurssilla saattaisi auttaa opiskelijoita ymmärtämään turvallista ohjelmointitapaa opintojen varhaisesta vaiheesta alkaen.

Koska molemmat ohjelmat ovat komentorivityökaluja, niitä pitäisi voida käyttää myös rinnakkain. Tätä varten tulisi kuitenkin vertailla, ovatko niiden havainnot pääosin päällekäisiä vai toisiaan täydentäviä.

3.3 Goblinin kehitysmahdollisuudet

Lisäämällä EXPACAan mahdollisuus ajaa ulkoisia työkaluja ennen koodin kääntämistä, voitaisiin koodille tehdä turvallisuusanalyysi ennen sen suorittamista. Kirjallisuuden pe-

rusteella analyysi voitaisiin tehdä Pscanilla, UNO:lla tai molemmilla peräkkäin. Tällöin olisi kuitenkin huomioitava, että analyysin päättymisestä ei voida olla varmoja. Toisaalta EXPACA osaa jo havaita koodin liian pitkän suoritusajan, jolloin tämän ominaisuuden suorittamiseen voitaisiin mahdollisesti käyttää samaa toiminnallisuutta kuin itse koodin suorittamiseen.

Sekä Pscan että UNO tulostavat ohjelmasta löytyvät virheet virtuaalikonsolille, jolloin niiden antamalla palautteella voitaisiin helposti vaikuttaa arvosteluun ja palautteena saatu tulos voitaisiin ohjata suoraan opiskelijalle. Teoriassa helpoin tapa toteuttaa tällainen toiminnallisuus olisi muuttaa EXPACAn konfiguraatiota siten, että käännetyn ohjelman sijasta ajettaisiin skripti, joka analysoi lähdekoodin ja suorittaa vasta sen jälkeen itse ohjelman. Goblinin pisteytys perustuu ohjelman tulosteeseen. Tällöin odottamaton analysaattorilta tuleva tuloste ja odotetun syötteen puuttuminen kokonaan johtaisi käytännössä tehtävän hylkääntymiseen.

Tutkimuksessa ei myöskään löytynyt viitteitä siihen, että EXPACAn turvallisuutta olisi erikseen tutkittu. Jo pelkkä ohjelmakoodin kriittinen tarkastelu saattaisi antaa käsityksen turvallisuuteen panostamisen tarpeellisuudesta.

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. Toisaalta SchemeROBOssa ei ole ollenkaan kurssinhallintatoimintoja, jotka ovat tarpeellisia kurssin järjestämiseksi.

Ohjelmoinnin massakurssien järjestämiseksi tarvitaan muitakin järjestelmiä kuin kotitehtävätarkistin ja kurssihenkilökunta toteuttaa tarvittavia työkaluja työn ohessa. Käytännön syistä johtuen tarkistimien turvallisuuteen ei välttämättä ehditä paneutua.

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 Hiisilä (2005) toteaa diplomityössään toistuvasti, että jotakin tiettyä muuta näkökulmaa ei työn puitteissa ehditty tutkimaan. Tästä syystä turvallisuutta on varmasti 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 ole käypä oletus. Tästä johtuen on ensiarvoisen tärkeää, että tarkistimen toteutus arvioidaan 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 vaikutus.

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. Tällainen palaute on myös todennäköisesti paljon seikkaperäisempää, kuin ongelman realisoituessa saatu virheilmoitus. Esimerkiksi UNIX-ympäristössä saatu ”Segmentation fault”virhe ei välttämättä auta vian löytämisessä ollenkaan, kun lähdekoodia analysoimalla vika voidaan paikallistaa oikealle riville. Oman kokemukseni mukaan ainakaan TKK:n tietotekniikan opinto-ohjelmaan ei ole kuulunut lainkaan staattisen analyysin työkaluja. Tällaisessa tapauksessa pitäisi kuitenkin arvioida työkaluja laajemmasta 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 voisi olla mielenkiintoinen aihe jatkotutkimukselle. Samoin Scheme-robossa oleva ohjelman mallirakenteen varmistavan työkalun toteuttaminen esimerkiksi C:lle tai Javalle voisi olla haastava ja mielenkiintoinen tutkimuskohde.

Kielen valintaan liittyvät turvallisuusnäkökulmat on hyödyllistä huomioida automaattista arvostelua harkitsevan kurssin suunnittelussa. 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.