Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi

Kandidaatintyö

5. toukokuuta 2011

Ferrix Hovi

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Ferrix Hovi
Työn nimi:	Ohjelmointiharjoitusten turvallisuuden automaattinen arviointi
Päiväys:	5. toukokuuta 2011
Sivumäärä:	20
Pääaine:	Ohjelmistotuotanto ja -liiketoiminta
Koodi:	T3003
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	Dosentti 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 5, 2011
Pages:	20
Major:	Ohjelmistotuotanto ja -liiketoiminta
Code:	T3003
Supervisor:	Professor (tem) Tomi Janhunen
Instructor:	Docent 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. Tästä johtuen opetusta joudutaan toteuttamaan massakursseina, joiden osallistujamäärät ovat kokonaisia vuosikursseja. Ohjelmoinnin opiskelussa käytännön harjoittelulla on merkittävä rooli ja ripeästi saatava palaute on olennainen osa tehokasta oppimista. Tämän saavuttaminen tehtäviä käsin tarkistamalla on suurilla kursseilla mahdotonta. Tästä syystä on monet opettajat, yliopistot ja yhtiöt ovat kehittäneet työkaluja, joilla opiskelijat voivat tarkistuttaa ohjelmointiharjoituksensa automaattisesti, saada palautetta ja kurssin suorittamiseksi vaadittuja 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 ja siitä käännetyn ohjelman ajonaikaista tarkkailua tavoitteena arvioida, onko ohjelman suorittaminen turvallista. Tutkimuksen tavoitteena on kartoittaa ohjelmointitehtävien automaattisten tarkistimien toteutukseen liittyviä turvallisuushaasteita ja yrittää löytää ratkaisuja kirjallisuudesta.

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. 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 nimenomaan itse tarkistimen toimintaan ja turvallisuuteen.

Väärä hälytys tarkoittaa sellaista virhehavaintoa, joka johtaa opiskelijan antaman harmittoman syötteen hylkäämiseen.

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 eriytetty 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.

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

Scheme-robon kirjoittamisessa tunnistettuja turvallisuusongelmia:

Tiettyjen kutsujen etsiminen lähdekoodista. Struktuurin analysointi <- parempi sana. Eval-kutsun kieltäminen I/O -primitiivit Hiekkalaatikko

Ceilidh -> CourseMarker Moodle VIOPE Aki Hiisilä Goblin

Käyttäjän tunnistaminen: Järjestelmän tulee tunnistaa käyttäjä ja mihin ryhmään käyttäjä kuuluu.

Satunnaisuus plagioinnin hallinnassa. Kielituki

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 käännön aikana tai sen jälkeen.

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.

Heffley ja Meunier (2004) tutkivat ohjelmien haavoittuvuuksia etsiviä työkaluja ja havaitsivat, että monet haavoittuvuuksia tutkivat työkalut tuottavat niin paljon vääriä varoituksia, etteivät ne ole käytännöllisiä. Työkalujen joukosta kuitenkin erottui yksi hyödyllinen työkalu, Pscan, joka onnistui antamaan luotettavia tuloksia rajatulla alueella.

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.

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.

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 kartoitetaan ensin olemassaolevaa tarkistinvalikoimaa ja tutustutaan niiden piirteisiin turvallisuusnäkökulmasta. Tämän jälkeen tarkastellaan lähemmin kahta hyvin erilaista TKK:lla käytettyä tarkistusjärjestelmää: Scheme-roboa ja Goblinia. Järjestelmiä tarkastellaan niistä kirjoitetun tieteellisen materiaalin avulla.

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

Viimeiseksi pohditaan lähemmin tarkasteltujen tarkistimien teknisistä valinnoista johtuvia turvallisuusseikkoja ja esimerkiksi kurssilla käytettävän ohjelmointikielen vaikutusta.

2.1 Tarkistimet

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 virtuaalikon-

solilla 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 tiedostoiden 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)

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). 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 ohjelmakoodin rakennetta johonkin opettajan määrittämään malliin ja hylätä tehtäviä niistä löytyvien funktiokutsujen nimien perusteella. (Saikkonen et al., 2001) Nämä eivät sinänsä ole turvallisuusominaisuuksia, mutta niitä voitaisiin käyttää myös sellaisina pienin muutoksin.

2.1.3 Lähemmin tarkasteltavat 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 siten laajasti koko tarkasteltua ryhmää.

2.2 Yleisimmät turvallisuusriskit

Merkittävä osuus CERT:n turvallisuussuosituksissa mainituista ongelmista viittaa puskurin ylivuotoon liittyviin ongelmiin (**?**). Tällä tarkoitetaan tilannetta, jossa ennalta määritellyn kokoiseen puskuriin kirjoitetaan enemmän dataa kuin puskurissa on tilaa ja myöhemmässä ohjelman suorituksessa ohjelmaprosessi voidaan kaapata mielivaltaisen koo-

din suorittamiseen käyttäen puskurin ulkopuolelle kirjoitettua dataa. Haavoittuvuus on tyypillistä ohjelmointikielissä, joissa muistin varaaminen ja osoittaminen on ohjelmoijan vastuulla. Nykyaikana tämä tarkoittaa lähinnä C- ja C++-kieliä. Dynaamiset ja funktionaaliset kielet eivät käsittele muistia suoraan, joten niissä puskurin ylivuoto ei ole mahdollinen 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.

2.3 Staattinen analyysi

Koodin turvallisuutta analysoivia työkaluja on tarjolla hyvin paljon C- ja C++-ohjelmille. Yleisimpiä työkalujen havaitsemia ongelmia ovat puskurin ylivuodot, null-dereferensoinnit ja taulukoiden indekseihin liittyvät ongelmat. Näiden syy voi olla joko tahallinen tai tahaton.

Tahallisten hyökkäysten estämiseksi yksinkertaisin keino on käyttää tekstihakua havaitsemaan komentoja, joita hyvissä aikeissa olevan opiskeijan 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 obfuskointitekniikoita, joilla staatinen analysaattori voidaan ohjata ratkaisemaan NP-täydellisiä ongelmia. Tutkimus käsittelee staattisen analyysin harhauttamiseen toimivia virustorjuntaohjelmistojen perspektiivistä.

Osa tekniikoista on kuitenkin kieliriippumattomia, 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 obfuskoidun haittaohjelman tekeminen on varmasti monimutkaisempaa kuin tehtävänannon seuraaminen. On kuitenkin hyvä tiedostaa, että staattinen analyysi ei riitä ainoaksi työkaluksi.

Tevis ja Hamilton (2004)

2.3.1 Pscan

Pscan on avoimen lähdekoodin työkalu (tarkista), jolla voidaan etsiä puskurin ylivuotoja lähdekoodista. Työkalu oli Meunier ja Laalaa tutkimuksen perusteella toiminnaltaan hyvin rajattu ja siitä syystä tuottaa vertailujoukossaan vähiten vääriä hälytyksiä.

2.3.2 Splint

Splint on analyysityökalu, joka yrittää havaita koodista löytyviä ongelmia sen joukkoon kirjoitettujen merkintöjen (engl. annotation) perusteella. Sen toimintamalli edellyttää ohjelmakoodin kirjoittajan tai tulkitsijan tunnistavan kyseiseen ohjelmaan liittyviä riskejä. Lisäksi ohjelmaa testattaessa selvisi, että työkalun väärien hälytysten määrä oli oli 75%.

2.3.3 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.

Alustamattomat muuttujat ja taulukon ulkopuolelle osoittaminen tarkoittavat käytännössä katsoen samaa kuin puskurin ylivuoto.

2.4 Sallitut ohjelmointikielet

Harjoituksissa käytetyn ohjelmointikielen valinnalla voi olla merkittävä vaikutus koodin turvallisuuteen. Esimerkiksi monissa imperatiivisissa ohjelmointikielissä ongelmalliset puskurin ylivuodot ja erilaiset rinnakkaisuuteen liittyvät ongelmat eivät toteudu monissa funktionaalisissa ohjelmointikielissä.

Esimerkiksi scheme-robo käyttää ohjelmointiharjoitusten arviointiin metasirkulaarista tulkkia, joka rajaa kielen riskialttiit toiminnot pois.

Kääntäjätoteutuksen valinta, turvallisuusauditointi... laalaa

2.4.1 Koodin ominaisuuksien rajaaminen

(Voiko mielivaltaisilla rajoilla, kuten koodin pituudella, rajapinnoilla, testikattavuudella, kompleksisuudella tai vastaavilla vaikuttaa niihin vihamielisiin kikkoihin, jotka mahtuvat mukaan?)

2.5 Haskell

Laiska tulkki saattaa kuluttaa kaikki resurssit loppuun sopivalla syötteellä

2.6 Ajonaikainen analyysi

2.6.1 Valgrind

Valgrind on työkalu, jolla voidaan havaita ohjelman toiminnassa tapahtuvia

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 kovin mielenkiintoinen. Toisaalta kielen funktionaalisen luonteen takia se ei myöskään määrittele suositummille 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.

Hiisilä (2005) mainitsee diplomityössään, ettei Goblin analysoi koodia staattisesti kääntäjää lukuunottamatta. Hän toteaa, että järjestelmä ei sellaisenaan tue työkaluja, mutta tunnistaa, 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 eriyttää riittävästi muusta ympäristöstä.

3.2 Tutkittujen työkalujen soveltuvuus

Splint pohjautuu koodin joukkoon kirjoitettaviin merkintöihin. Niiden syöttäminen edellyttää koodin kirjoittajalta riskien tuntemusta, mikä ei todennäköisesti ole kohtuullinen olettamus ohjelmoinnin perusopetuksessa.

Merkintöjen automaattinen syöttäminen ohjelmakoodin joukkoon saattaa niinikään olla kohtuuttoman vaivalloinen tapa tutkia koodia.

Toisaalta työkalun käytön harjoitteleminen saattaisi olla hyvä tapa tutustuttaa ohjelmoinnin opiskelijoita turvalliseen ohjelmointiin.

3.3 Tarkistimen auditointi

3.4 Ohjelmointikielen vaikutus

3.5 Mahdolliset syyt

? 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 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. Käytännön syistä johtuen tarkistimien turvallisuuteen ei välttämättä ehditä paneutua.

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 kurssikokonaisuuden järjestäminen on tärkeämpää kuin turvallisuuden yksityiskohtien hiominen. Esimerkiksi Hiisilä (2005) toteaa diplomityössään toistuvasti, että jotakin tiettyä näkökulmaa ei työn puitteissa ehditty tutkimaan.

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 hyväksyttävä 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. Oman kokemukseni mukaan ainakaan TKK:n tietotekniikan opinto-ohjelmaan ei ole kuulunut lainkaan staattisen analyysin työkaluja.

Ohjelmointiharjoituksissa käytettävällä kielellä on suuri vaikutus turvallisuuteen. Esimerkiksi TKK:lla aiemmin järjestetyllä Scheme-ohjelmointikurssilla käytetty Scheme-robo rajaa kielen mahdollisesti vaarattomat ominaisuudet työkalun ulkopuolelle onnistuneesti. Tällä tavoin saadaan eliminoitua kokonaisia ongelmakategorioita. Tällaisen tomintamallin toteutus ei kuitenkaan ole yhtä helppoa kaikilla ohjelmointikielillä, vaan edellyttää monille funktionaalisille kielille tyypillisiä tulkin kirjoittamiseen tarkoitettuja ominaisuuksia. Tästä syystä samanlaisen ratkaisun löytäminen esimerkiksi C-kielelle voisi olla haastava jatkotutkimuksen aihe.

Turvallisuusnäkökulmasta funktionaaliset ohjelmointikielet aiheuttavat paljon vähemmän potentiaalisia ongelmia kuin matalan tason imperatiiviset kielet. Myös dynaamiset kielet ovat immuuneja osalle näistä ongelmista. Näiden seikkojen tulisi kuulua ohjelmoinnin peruskursseihin, jossa monet opiskelijat tutustuvat ohjelmointiin ja siihen liittyviin ongelmiin ensimmäistä kertaa. Kyky valita tarkoitukseen sopiva ohjelmointikieli useamman vaihtoehdon joukosta, voisi hyvinkin auttaa kokonaisten ongelmakategorioiden poistamisessa työelämässä. Vastaavasti on turvallisempaa opettaa muiden alojen opiskelijoille ohjelmointia korkean tason kielillä, jotka eivät ole yhtä alttiita tietoturva-aukoille. Tällöin ohjelmoinnista jää käteen enemmän ilmaisuvoimaa ja vähemmän mahdollisia ongelmia.

http://ieeexplore.ieee.org.libproxy.tkk.fi/stamp/stamp.jsp?tp=&arnumber=1265998

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

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

Veli-Jussi 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.

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.

ST 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.