Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

Ilkeiden jutukkeiden pyöräyttäminen

Kandidaatintyö

31. helmikuuta 2011

Ferrix Hovi

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Ferrix Hovi
Työn nimi:	Ilkeiden jutukkeiden pyöräyttäminen
Päiväys:	31. helmikuuta 2011
Sivumäärä:	999
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:	Dances with evil thingamajiggies
Date:	February 31, 2011
Pages:	999
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

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 vahventaa ohjelmointiharjoitusten arvosteluun käytettäviä työkaluja, jotta ne selviäisivät tahattomasti ja tahallisesti tehdyistä hyökkäyksistä eheinä ja toimintakykyisinä. Järjestelmän olennaisia vaatimuksia ovat nopea palaute, väärien hälytysten minimointi ja käytettävyys. (Joku lähde tai parempi tavoiteasettelu tähän?) Vahvennusmenetelmät eivät saa merkittävästi haitata järjestelmän toimintaa.

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

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

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 hyö-

dyllinen 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.

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

Ohjelmointiharjoituksen tarkistimessa voidaan varautua tietoturvaongelmiin kahdella tasolla: huolehtimalla tarkistimen turvallisuudesta ja tutkimalla syötteenä tulevaa ohjelmakoodia kriittisesti.

(Pitäisikö olla joku referenssitoteutus? Webcatin tutkimuksessa on keskitytty oppimisvaikutusten tutkimukseen.)

2.1 Staattinen analyysi

2.1.1 Pscan

(Oiskohan hyvä kerätä tietoa työkaluista, joista on apua itse tarkistimen vahvistamisessa?)

2.2 Sallitun toteutuksen rajaaminen

2.2.1 Kielen rajaaminen

(Voiko käytettyä ohjelmointikieltä rajoittaa siten, että selkeästi haitalliset komennot eivät ole saatavilla?)

2.2.2 Koodin ominaisuuksien rajaaminen

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

2.3 Ajonaikainen analyysi

2.3.1 Debuggeri

(Saavutetaanko koodin ajamisella debuggerissa etua, jos jotain jäi huomaamatta? Mitä haasteita debuggaamiseen liittyy?)

3 Tulokset

4 Johtopäätökset

Lähteet

J. Heffley ja P. Meunier. Can source code auditing software identify common vulnerabilities and be used to evaluate software security? *System Sciences, 2004.Proceedings of the 37th Annual Hawaii International Conference on*, 2004. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1265654`. ID: IEEE_XPLORE_NEW; ID: IEEE/IEE Electronic Library.

S. T. Taft. Systematic Scanning for Malicious Source Code. *Technologies for Homeland Security, 2008 IEEE Conference on*, sivut 173–175, 2008. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4534444`. ID: IEEE_XPLORE_NEW; ID: IEEE/IEE Electronic Library.