Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Opinnäytetyö

Juho Lehtonen

Java-sovelluksen kopiosuojaus

Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä Kilosoft Oy, valvojana arkkitehtisuunnittelija Jari Länsiö
Tampere 05/2010

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tekijä	Juho Lehtonen
Työn nimi	Java-sovelluksen kopiosuojaus
Sivumäärä	41
Valmistumisaika	05/2010
Työn ohjaaja	lehtori Erkki Hietalahti
Työn teettäjä	Kilosoft Oy, valvoja arkkitehtisuunnittelija Jari Länsiö

TIIVISTELMÄ

Tämä tutkimustyö käsittelee Kilosoft Oy:n tuotekehityksessä olevan sovelluksen kopiosuojausta. Työn tavoitteena oli perehtyä kyseisen Java-sovelluksen mahdollisiin suojaustapoihin.

Työssä tutkittiin Java-sovelluksen suojaukseen liittyviä tekniikoita ja puntaroitiin kunkin edut ja haitat. Tämän lisäksi työssä selvitettiin yleisiä tietoturvan käsitteitä ja sitä, mitä ne tarkoittavat Java-ohjelmointikielessä, sekä käytiin yleispiirteittäin läpi Java-ajoympäristön rakenne.

Tutkimuksissa selvisi, että ainoa mahdollinen tapa suojata kohteena ollut sovellus on käyttää Excelsior JET -kääntäjää. Tutkimustuloksiin vaikuttivat rajoittavat tekijät, kuten suojaamiseen käytettävän ajan vähyys sekä sovelluksen kompleksinen rakenne. Tutkimustuloksien luotettavuus ja suojatun sovelluksen toimivuus todennettiin Kilosoft Oy:n testausryhmän suunnittelemien testien avulla. Suojatun sovelluksen toiminnoissa ei havaittu puutteita eikä häiriöitä.

Työn sisällön perusteella aiheeseen tarkemmin perehtymättömän lukijan toivotaan saavan hyvän peruskäsityksen Java-ohjelmointiin liittyvästä tietoturvasta sekä suojaamiseen käytettävistä työkaluista. Työn lukijalta odotetaan kuitenkin Java-ohjelmoinnin perustietämystä.

Työssä esiintyvissä kuvissa käytetään taustatyöksi tutkittuja työkaluja ja tekniikoita, jotka esitellään opinnäytetyön edetessä.

Tässä työssä ei käsitellä Kilosoft Oy:n sovellusta, joka oli kopiosuojauksen kohteena.

Avainsanat: Java, sovelluksen suojaaminen, takaisinkääntäminen, monimutkaistus, salaus, konekieli

TAMK University of Applied Sciences
Programme in information technology
Software engineering

Author(s)	Juho Lehtonen
Name of the report	Protecting Java application from Reverse Engineering
Number of pages	41
Graduation time	May 2010
Thesis supervisor	lecturer Erkki Hietalahti
Commissioned by	Kilosoft Ltd, supervisor software architect Jari Länsiö

ABSTRACT

The purpose of this thesis was to research the right protection model againts reverse engineering for the Kilosoft Ltd.'s application. It also covers basic knowledge needed to protect Java applications from reverse engineering.

This thesis gives reader basic understanding of protecting Java application, security in general and what does security mean in Java programming language.

The tools and techniques featured in this thesis were developed as background work for this thesis.

Taking into account the constrains the only possible way to protect the application was to use Excelsior JET compiler tool. Compilation didn't affect to the application's functionality. The results was verified by Kilosoft Ltd's test group.

People with basic Java programming knowledge should get a good introduction and right tools for securing Java based application.

This thesis does not cover introduction of Kilosoft Ltd's application.

Keywords	Java, Reverse Engineering, decompile, obfuscation, encryption, machine language

Alkusanat

Tämä opinnäytetyö on tehty helmikuun ja huhtikuun välisenä aikana 2010. Aiheen valintaan vaikutti Kilosoft Oy:ssä joulukuun alussa aloittamani työsuhde. Työskentelen tuotekehitysryhmässä, joka tekee opinnäytteen pohjana olevaa sovellusta.

Aihetta voidaan pitää onnistuneena valintana minulle, sillä sen tekeminen tuntui mielekkäältä ja se lisäsi tietämystäni Java-sovelluksen suojaamisesta. Tehdessäni 7,5-tuntista päivää töissä ja töiden ulkopuolella illat, viikonloput sekä juhlapyhät opinnäytetyötä, tuntui kahden kuukauden opinnäytetyörypistys todella rankalta ajalta. Huomattakoon tosin, että aikamääreet eivät tulleet yrityksen puolelta vaan halustani valmistua.

Tampereella 5. huhtikuuta 2010

Juho Lehtonen

Sisällysluettelo

Termit ja lyhenteet

SDK	(Software Development Kit) on tyypillisesti kokoelma kehitystyökaluja, joiden avulla voidaan toteuttaa sovelluksia. /1/
JDK	(Java Development Kit) on Sun Microsystemsin tuote Java-sovellusten suunnittelijoille. Javan julkistamisen jälkeen se on ollut käytetyin Java-kielen SDK.
JRE	(Java Runtime Enviroment) on Javan ajoympäristö, joka tarvitaan käännettyjen ohjelmien suorittamiseen. /1/
API	(Application Programming Interface) sovellusohjelmointirajapinta.
J2SE	(Java 2 Standard Edition) on Java-ohjelmointikielen standardi, joka sisältää yleisimmät ominaisuudet, graafiset käyttöliittymät, tietokanta- ja XML-rajapinnat. /20/
J2EE	(Java 2 Enterprise Edition) on Java-ohjelmointikielen standardi, jolla on mahdollista tuottaa palvelinsovelluksia niin, että sovelluksia voi ajaa millä tahansa J2EE-palvelimella. /21/
JSP	(Java Server Pages) on skriptaukseen perustuva ohjelmointitekniikka, jossa HTML-koodin sekaan on mahdollista lisätä Java-kielellä kirjoitettua koodia.
Java-pavut	(JavaBeans) ovat Java-ohjelmointikielellä toteutettuja komponentteja, joita voidaan uudelleenkäyttää. Java-papuja voidaan kutsua JSP-sivuilla. /22/
AOT	(ahead-of-time)-käännöksessä koodi muunnetaan binääriseksi ennen sovelluksen suoritusta. Tällöin suorituskykyä ei mene sovelluksen suorituksen aikana kääntämiseen.
JIT	(Just-in-time)-käännoksessä koodi muunnetan binääriseksi suorituksen aikana, mikä saattaa vähentää sovelluksen suorituskykyä.

GNU	(GNU's Not Unix) on projekti, jonka tarkoituksena on kehittää täysin vapaa käyttöjärjestelmä. GNU suunniteltiin yhteensopivaksi Unixien kanssa. /14/
MinGW	(Minimalist GNU) on paketti Windows-käyttöjärjestelmille. Paketti koostuu GNU-kääntäjäkokoelmasta (GCC) sekä muista tarvittavista GNU-tavutiedostoista natiivin Windows-sovelluksen kehittämiseen. /15/
x86	on Intelin kehittämän suoritinarkkitehtuurin nimi. Sen mukaisille suorittimille yhteisiä piirteitä ovat peruskäskykanta, rekisterit ja segmentoitu muistiarkkitehtuuri. /16/

1 JOHDANTO

Tämä tutkimustyö käsittelee Java-sovellusten tietoturvaa ja erityisesti sen kopiosuojausta. Työ voidaan jaotella selkeästi kahteen eri osaan, teoriaan ja käytännön läheiseen osuuteen. Työn osat muodostavat yhtenäisen kokonaisuuden, jonka avulla aiheeseen perehtymätön lukija saa hyvän perustietämyksen Java-sovelluksien suojaamisesta.

Teoriaosuudessa tarkastellaan myös tietoturvaa yleisenä käsitteenä ja käsitteen merkitystä Java-ohjelmoinnissa sekä määritellään suojauksen kohteena olevan sovelluksen riskit. Teoriaosuudessa käydään myös läpi asiat, jotka tekevät Javan tavukooditiedostosta niin haavoittuvan. Osuuden lopuksi listataan mahdolliset sovelluksen suojaustavat hyötyineen ja haittoineen sekä niiden käyttökelpoisuus yrityksen sovellukselle.

Luvut viisi ja kuusi muodostavat opinnäytetyön käytännönläheisen osuuden. Osuudessa käydään läpi teoriaosuuden pohdinnan tuloksena syntyneet suojaustavat, monimutkaistus ja käyttöjärjestelmälle nativiiksi käännetty konekoodi, joita on ideaalinen kokeilla kyseiselle sovellukselle. Osuudessa selviää, kuinka hyvin suojaustavat selvisivät työstään, kun kohteena oli äärimmäisen kompleksinen ja montaa eri tekniikkaa hyödyntyvä sovellus. Käytännönläheisen osuuden lopuksi arvioidaan suojaustavaksi valitun työkalun käyttökokemukset sekä työkalun sopivuus sovelluksen suojaamiseen.

2 TIETOTURVA

Tietoturvallisuudella tarkoitetaan yleisesti tietojen, järjestelmien ja palveluiden suojaamista sekä normaali- että poikkeusoloissa hallinnollisten ja teknisten toimenpiteiden avulla. Tietoturvallisuus rakentuu tiedon kolmesta pääominaisuudesta: luottamuksellisuudesta, eheydestä sekä käytettävyyden turvaamisesta.

Java-ohjelmoinnissa “tietoturva” sanalla voidaan tarkoittaa useita eri asioita. Todellisuudessa tietoturvakäsite muodostaa kokonaisuuden useasta eri osa-alueesta.

2.1 Tietoturvan käsitteet

Tietoturva voidaan määritellä sille tyypillisten käsitteiden avulla. Sovellusta voidaan pitää sitä turvallisempana, mitä useampi käsite toteutuksessa on otettu huomioon. Tietoturvalle tyypillisiä käsitteitä ovat seuraavat:

Luottamuksellisuus: tiedot, järjestelmät ja palvelut ovat saatavilla vain niille oikeutetuille eikä niitä luvatta paljasteta tai muutoin saateta sivullisten tietoon.

Eheys: tiedot, järjestelmät tai palvelut eivät ole laitteisto- tai ohjelmistovikojen, luonnontapahtumien tai oikeudettoman inhimillisen toiminnan seurauksena muuttuneet tai tuhoutuneet.

Käytettävyys: tiedot, järjestelmät ja palvelut ovat tarvittaessa niihin oikeutetuille esteettä hyödynnettävissä.

Pääsynvalvonta: tietoa tai tietojärjestelmää ei voi käyttää ilman lupaa.

Todentaminen: osapuolten todentamisella (autentikointi) tarkoitetaan osapuolten (henkilö tai järjestelmä) luotettavaa tunnistamista keskenään. Todentamisella viitataan yleensä kohteen ominaisuuksiin, mitä kohteella on hallussaan tai mitä kohde tietää. Hyvin suunniteltu todennusmenetelmä on yhdistelmä edellisistä menetelmistä. Usein järjestelmät toteuttavat tunnistamisen ja todentamisen samaan aikaan.

Kiistämättömyys: todisteita luodaan sen varmistamiseksi, ettei yksikään tietojen käsittelyn tai siirron osapuoli voi jälkikäteen kiistää osuuttaan siihen (juridinen sitovuus). Tietojen ja tietoaineistojen osalta kiistämättömyys voi koskea esimerkiksi tietoa siitä, kuka on muokannut tietoja. Sähköisessä viestinnässä kiistämättömyys tarkoittaa muun muassa toimenpiteitä, joilla varmistutaan viestin lähettäjästä ja viestin vastaanottajasta.

Tunnistaminen: menettely, jolla yksilöidään kohde, kuten käyttäjä tai järjestelmä.

2.2 Tietoturva osa-alueina

Organisaation ja käyttäjien toimenpiteet (suunnittelu, toteutus, valvonta) tietoturvallisuuden osa-alueiden toteuttamiseksi voidaan jakaa kahdeksaan alueeseen:

- hallinnollinen ja organisatorinen tietoturvallisuus
- henkilöstöturvallisuus
- fyysinen turvallisuus
- tietoliikenneturvallisuus
- laitteistoturvallisuus
- ohjelmistoturvallisuus
- tietoaineistoturvallisuus
- käyttöturvallisuus /10/

Tämä tutkimustyö sijoittuu ohjelmistoturvallisuuden sekä tietoaineistoturvallisuuden osa-alueille.

2.3 Tietoturva Java-ohjelmointikielessä

Java-ohjelmointikielessä voidaan soveltaa yleisen tietoturvan käsitteitä. Ennen kuin voidaan alkaa kehittää turvallista sovellusta, pitää määrittellä päämäärät. Java-sovellus voidaan teoriassa todeta tietoturvalliseksi, jos siihen on lisätty seuraavat ominaisuudet:

Suojattu vahingoittavilta sovelluksilta. Haittaohjelmat eivät saa vahingoittaa käyttäjän ympäristöä esimerkiksi levittämällä toistuvasti viruksia.

Suojattu tunkeilijoilta. Ohjelmien tulisi estää tunkeilijoiden pääsyn yksityisen tietokoneen tai -verkon tietoihin.

Todennettu. Osapuolten, sekä isännän että ohjelman käyttäjät, henkilöllisyys tulisi olla varmistettu.

Salattu. Ohjelman lähettämä ja vastaanottama, niin verkossa kulkeva kuin tietokantaan tallennettu, data tulisi salata.

Tarkistettu. Mahdolliset turvallisuuskriittiset operaatiot tulisi aina kirjata talteen jäljitettävyyden parantamiseksi.

Hyvin määritelty. Tietoturva tulisi aina määritellä jo etukäteen hyvin.

Tarkistus. Operaatioden säännöt tulisi asettaa jo etukäteen ja muistaa myös tarkastaa ne.

Kopiosuojattu. Valmiin sovelluksen kopiointi ja uudelleenkäyttö kolmannen osapuolen toimesta tulisi estää. /5/

2.4 Sovelluksen riskien määrittely

Suojattava sovellus koostuu ryppäästä ohjelmistomoduuleja. Yksi asennettu sovellus muodostaa aina itsenäisen yksikön. Suojattavan sovelluksen päätarkoitus on yksiköiden välinen automatisoitu keskustelu sekä verkkojen hallinta. Tuotetta valmistava yritys tarvitsee teknistä suojaa sovelluksensa komponenteille ja suojaa laittomia sovelluksen muutoksia vastaan. Yksi tärkeimmistä vaatimuksista on, että suojaus ei saa vaikuttaa negatiivisesti verkon yhteyksien nopeuteen tai sovelluksen suorituskykyyn.

Tiedossa on uhkia, jotka altistavat sovelluksen väärinkäytölle. Sovelluksen käyttäjät pääsevät fyysisesti käsiksi ohjelmiston tiedostoihin ja tämä mahdollistaa seuraavat uhkat:

a) Sovellus voidaan yrittää kopioida useammalle koneelle ilman tuotteen valmistajan lupaa.

b) Asiakkaalle käännettyä ohjelman käyttöversiota voidaan yrittää kääntää takaisin lähdekoodiksi, mikä voisi olla teoriassa erittäin hyödyllistä kilpailevalle yritykselle.

Tutkimustyö käsittelee kaupallisen sovelluksen kopiosuojausta. Suojauksella yritetään vaikeuttaa sovelluksen takaisin lähdekoodiksi kääntämistä siten, että se ei olisi enää kannattavaa kilpailevalle yritykselle.

3 TAKAISINKÄÄNTÖ (DECOMPILE)

Takaisinkääntämisellä tarkoitetaan sovelluksen tavukooditiedoston kääntämistä lähdekoodiksi. On tilanteita, jolloin takaisinkääntäminen on paras, ellei jopa ainoa, tapa saada sovellus takaisin lähdekoodiksi. Tässä muutama syy suorittaa takaisinkääntö:

- palauttaa lähdekoodi, joka on vahingossa hukkunut
- oppia tietyn ominaisuuden tai tempun toteuttamistapa
- vianetsiminen sovelluksesta tai kirjastosta, jolla ei ole hyvää dokumentointia
- korjatakseen virheen itse kolmannen osapuolen ohjelmasta, josta ei lähdekoodia ole saatavilla eikä ole aikaa odottaa tulevaa korjausta
- oppia suojaamaan koodia ulkopuolisilta

Takaisinkääntö on nimensä mukaisesti käänteinen toimenpide koodin kääntämiselle. Samaan tapaan kuin lähdekoodin kääntäminen tavukoodiksi, voidaan suorittaa myös tavukoodin takaisinkääntö lähdekoodiksi. Toteuttaakseen takaisinkääntämisen tarvitaan siihen oikeat työkalut. Useimmat takaisinkääntäjät pystyvät tuottamaan tavukoodista lähes alkuperäisen tasoisen lähdekoodin. Takaisinkääntäjistä löytyy niin ilmaisia versiota kuin kauppallisiakin. Takaisinkääntäjän ominaisuudet kannattaa kuitenkin selvittää tarkasti ennen ostoa.

3.1 Mikä tekee takaisinkääntämisestä mahdollisen?

Java -lähdekoodia ei käännetä samaan tapaan binääriseksi konekoodiksi kuten esimerkiksi C/C++ -lähdekoodi. Käännettäessä Java -lähdekoodia syntyy tuloksena välimallin tavukoodi, joka ei ole sidoksissa juuri tiettyyn käyttöjärjestelmään. Tavukoodi on mahdollista tulkita tai kääntää. Tämä mahdollistaa käytännössä korkeamman tason ohjelmointikielen muuntamista alemman tason tavukoodiksi. Tavukoodi sisältää kaiken tärkeän informaation, joka löytyy lähdekoodista. Vaikka lähdekoodista löytyvät kommentit ja muotoilut katoavat käännettäessä, kaikki metodit, muuttujat ja ohjelman logiikka löytyvät tavukoodista. Käännetty tavukoodi ei ole alimman tason konekieltä, minkä vuoksi sen muoto muistuttaa lähdekoodia. Java-virtuaalikoneelle määritellään käännöksessä ohjeet, jotka vastaavat Java-

ohjelmointikielen operaatioita ja avainsanoja. Kuvassa 1 on esitetty, miltä Java-ohjelman metodi näyttää tavukoodin ohjeina.

Metodi lähdekoodissa

```
public String getDisplayName() {
    return getUserName() + " (" + getHostName() + ")";
}
```

Metodi ohjeina tavukoodissa

```
 0 new #4 <java/lang/StringBuffer>
 3 dup
 4 aload_0
 5 invokevirtual #5 <covertjava/decompile/MessageInfoComplex.getUserName>
 8 invokestatic #6 <java/lang/String.valueOf>
11 invokestatic #6 <java/lang/String.valueOf>
14 invokespecial #7 <java/lang/StringBuffer.<init>>
17 ldc #8 < (>
19 invokevirtual #9 <java/lang/StringBuffer.append>
22 aload_0
23 invokevirtual #10 <covertjava/decompile/MessageInfoComplex.getHostName>
26 invokevirtual #9 <java/lang/StringBuffer.append>
29 ldc #11 <)>
31 invokevirtual #9 <java/lang/StringBuffer.append>
34 invokestatic #6 <java/lang/String.valueOf>
37 invokestatic #6 <java/lang/String.valueOf>
40 areturn
```

KUVA 1. Lähdekoodin metodi tavukoodin ohjeina

Kuten kuvasta 1 ilmenee, tavukoodin ohjeista löytyy yhtäläisyyksiä verrattaessa sitä lähdekoodin metodiin. Takaisinkääntäjä lataa tavukoodin ja yrittää rakentaa lähdekoodin ohjeiden avulla. Luokkissa olleiden metodien ja muuttujien nimet yleensä säilyvät tavukoodissa, mutta metodien parametrien ja paikallisten muuttujien nimet ovat hävinneet. Mikäli tavukoodista löytyy myös vianmäärittelyyn käytetyt tiedot, pystytään takaisinkäännettyyn lähdekoodiin lisäämään myös alkuperäisten parametrien nimet sekä rivinumerointi. Tämä helpottaa entisestään takaisinkääntämistä. /8/

Kuten kappaleessa kävi ilmi, kolmannen osapuolen sovelluksen kääntäminen takaisin lähdekoodiksi on erittäin yksinkertaista. Tämän vuoksi se luo merkittävän riskin esimerkiksi sovelluksen logiikan kopioinnille omaan sovellukseen. Takaisinkääntämistä on erittäin vaikea todistaa tapahtuneeksi, joten siitä ei jää edes kiinnijäämisen pelkoa.

3.2 Takaisinkääntämiseen tarkoitettujen työkalujen vertailu

Tutkimustyössäni ei löytynyt ilmaisista takaisinkääntäjistä, kuten JAD:stä tai JODE:sta, tärkeitä toiminnallisia puutteita verrattaessa niitä kaupallisiin työkaluihin. Taulukossa 1 on listattu käytetyimmät takaisinkääntäjät ja kerrottu, kuinka hyvin takaisinkääntäjä tukee kehittyneiden kielien rakenteita, kuten sisäisiä luokkia ja anonyymeja toteutuksia. Vaikka tavukoodi on ollut rakenteeltaan hyvin samanlainen JDK 1.1:stä asti, on tärkeää käyttää takaisinkääntäjää, jota päivitetään jatkuvasti ylläpitäjien toimesta. Kannattaa siis muista päivittää takaisinkääntäjä uusimpaan versioon kun uusi JDK julkaistaan. /7/

TAULUKKO 1. Takaisinkääntäjien vertailu

TYÖKALUT/ ARVIO	LISENSSI	KUVAUS
JAD/Erinomainen	Ilmainen (ei ammattikäyttöön)	JAD on todella nopea, luotettava ja älykäs takaisinkääntäjä. Sillä on täysi tuki sisäisille luokille, nimettömille toteutuksille sekä muille kehittyneille kieliominaisuuksille. Luotu koodi on helppolukuista ja hyvin organisoitua. Useat muut takaisinkääntöympäristöt käyttävät komentoriviltä JADia takaisinkääntömoottorinaan.
JODE/Erinomainen	GNU, julkinen lisenssi	JODE on todella hyvä takaisinkääntäjä. JODE on kirjoitettu Javalla ja se on saatavilla alkuperäisenä lähdekoodina osoitteessa SourceForge.net. Se ei välttämättä ole yhtä nopea ja laaja kuin JAD, mutta se tuottaa erinomaisia tuloksia, ajoittain jopa helppolukuisempia kuin JAD.
Mocha/Kohtalainen	Ilmainen	Mocha on ensimmäinen hyvin tunnettu takaisinkääntäjä, joka on saanut aikaan paljon lakikiistoja, mutta myös paljon herättänyt innostusta. Mocha teki ilmeiseksi, että Java-lähdekoodi voi olla jälleenrakennettu lähes alkuperäiseen muotoonsa. Julkista koodia ei ole päivitetty 1996 vuoden jälkeen, vaikka Borland arvatenkin päivitti ja yhdisti sen Jbuilderiin.

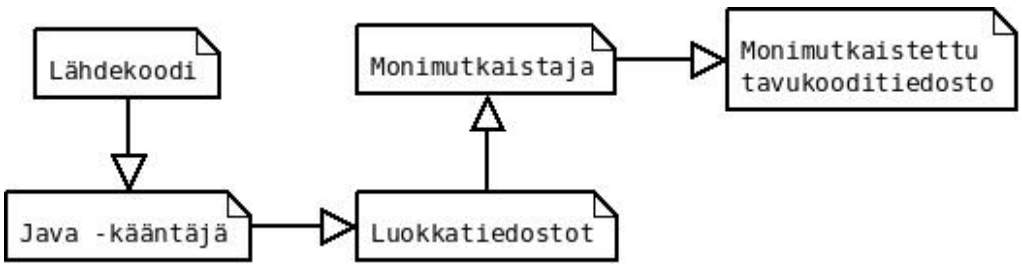
Vaikka markkinoilta löytyykin muita takaisinkääntäjiä, ovat JAD ja JODE toiminnallisuuksiltaan useimmiten riittäviä käyttäjän tarpeisiin ja tämän takia laajasti käytettyjä. Monet tuotteista tarjoavat graafisen käyttöliittymän, vaikka varsinaisen työn tekee yksinkertainen takaisinkääntäjä. Hyviä esimerkkejä näistä ovat muun muassa Decafe, DJ ja Cavaj, jotka käyttävät JADea takaisinkääntämiseen. Tästä syystä en ottanut niitä lainkaan arvioitavaksi. /9/

4 TUNNETUT SOVELLUKSEN SUOJAUSTAVAT

Päätavoite sovelluksen suojauksessa on tehdä valmiin ohjelmistotuotteen kääntäminen takaisin lähdekoodiksi entistä vaikeammaksi. Hyvin suojatun sovelluksen takaisin lähdekoodiksi kääntäminen ei ole taloudellisesti kannattavaa ja voi olla jopa teknisesti mahdotonta. Tässä luvussa tarkastellaan monimutkaistajaa (obfuscator), salausta (encyption), natiiveja konekieliä, sovelluksen suorittamista palvelimella sekä muita suojausmahdollisuuksia. Taskasteltavaksi otetaan kuitenkin vain Java-sovelluksille sopivimmat suojaustekniikat.

4.1 Monimutkaistaja (obfuscator)

Tänä päivänä internet tarjoaa monta ilmaista avoimeen lähdekoodiin perustuvaa takaisinkääntäjää (decompiler), joilla käännetty sovellus on helppo kääntää takaisin lähdekoodiksi. Tämä tekee oman sovelluksen ja sen arvokkaiden salaisuuksien suojaamisen entistä vaikeammaksi. Takaisinkääntäjille on kuitenkin ilmestynyt vihollinen: monimutkaistaja, joka tekee parhaansa vaikeuttaakseen takaisinkääntämistä. Monimutkaistus suoritetaan ohjelmistotaloissa valmiille ja testatulle sovellukselle. Monimutkaistettu sovellus kuitenkin testataan vielä ennen asiakkaalle lähetystä. Kuvassa 2 on esitetty sovelluksen matka lähdekoodista monimutkaistettuun tavukooditiedostoon. Lähdekoodi käännetään luokkatiedostoiksi, jotka monimutkaistetaan ja tuloksena syntyy monimutkaistettu tavukooditiedosto.



KUVA 2. Lähdekoodista monimutkaistettu tavukooditiedosto

4.1.1 Monimutkaistus yksinkertaisimmillaan

Koodin monimutkaistaja on nykyään yksi käytetyimmistä tavoista suojata Javalla ohjelmoitu sovellus takaisinkääntämiseltä. Monimutkaistaja muuttaa sovelluksen koodin epärationaaliseksi, vaikka se toimii täysin samalla tavalla kuin alkuperäinen koodi. Tämä tekee ohjelmiston toimintalogiikan ymmärtämisestä huomattavasti vaikeampaa ja täten hankaloittaa takaisin kääntämistä.

Kuvien 3 ja 4 esimerkissä, luokan "`Testitapaus`" tiedostoista muodostuu uusi luokka "`t`" monimutkaistajan avulla. Tuloksena syntynyt luokka ei täsmää alkuperäisen luokan kanssa ja on huomattavasti hankalalukuisempaa kuin alkuperäinen koodi. Molemmat ohjelmat kuitenkin toimivat täysin samalla tavalla.

```
1 class Testitapaus {
2    public Testitapaus() {
3        int num=1;
4    }
5    public String metodi(String teksti){
6        return teksti;
7 }
```

KUVA 3. Monimutkaistamaton luokka

Käyttämällä yksinkertaisinta monimutkaistajaa (esimerkiksi KlassMasteria), kaikki luokan nimet muunnetaan sekalaisiksi kuten kuvassa 4 näkyy. Tämän lisäksi rivinumerointi poistuu kokonaan, joten todellisuudessa merkit tulisivat yhdessä jonossa ilman minkäänlaista jäsentelyä. /7/

```
1 class t {
2    public static boolean t;
3    public t() {
4        int t=1;
5    }
6    public String t(String a){
7        return a;
8 }
```

KUVA 4. Monimutkaistettu luokka

Kuten kuvista 3 ja 4 selviää, Testitapaus -luokka on muuttunut KlassMaster -monimutkaistajan avulla luokaksi `t`. Myös `metodi(java.lang.String)` on muuttunut vaihtoehtoisesti muotoon `t(java.lang.String)`. Metodin nimi `t()`, on huomattavasti vaikeampi ymmärtää kuin alkuperäinen `metodi()`, sillä metodien nimet yleensä luodaan metodia kuvaavaksi ohjelmiston kehitystä helpottamaan. Tämän lisäksi voidaan huomata verratessa alkuperäistä tavukoodia monimutkaistettuun tavukoodiin, että rivinumerointi on poistunut kokonaan. Tämä vaikeuttaa sovelluksen takaisinkääntäjiä entisestään. Metodien alkuperäiset nimet muutetaan pysyvästi monimutkaistuksessa. Monimutkaistettu tavukooditiedosto ei sisällä mitään jälkiä alkuperäisistä metodien, luokkien tai muuttujien nimistä.

4.1.2 Muut tunnetut monimutkaistustekniikat

Vaikka esimerkin monimutkaistaja ei tee muuta kuin vaihtaa luokan nimet ja poistaa rivinumeroinnin, on olemassa muitakin tapoja monimutkaistaa sovelluksen kääntämistä takaisin lähdekoodiksi. Yksi suosittu tapa epäselventää lähdekoodia on muuttaa joitakin luokassa olevien symboleiden nimiä niin, että ne eivät tarkoita mitään. Muutettava saattaa olla esimerkiksi avainsana kuten “private” tai vielä pahempaa, jokin täysin merkityksetön symboli. Jotkin virtuaalikoneista, etenkin selaimet, eivät suhtaudu suopeasti kyseiseen temppuun. Kun muuttujalla on symboli (esimerkiksi “=”) nimenä, Java-ohjelmointikielen määrittelyn vastaisesti, jotkin virtuaalikoneista ohittavat sen automaattisesti.

Tekniikat, joita jotkin monimutkaistajat käyttävät, ovat tarkoitettu yksinomaan tietyille takaisin kääntäjille kuten Mocha:lle tai JODE:lle. Kuten kuvien 5 ja 6 esimerkki havainnollistaa: koodiin lisätään niin sanottuja vääriä ohjeita, jotka eivät kuitenkaan vaikuta ohjelman ajoon. Huonot ohjeet kuitenkin onnistuvat kaatamaan takaisinkääntäjän ja tuloksena käännös epäonnistuu.

```
Method void main(java.lang.String[])
 0 new #4
 3 invokespecial #10
 6 return
```

KUVA 5. Alkuperäisen tavukoodin ohjeet

```
Method void main(java.lang.String[])
 0 new #4
 3 invokespecial #10
 6 return
 7 pop
```

KUVA 6. Monimutkaistuksessa väärät ohjeet lisätty tavukoodiin

Kuten kuvasta 6 näkyy, suoritukselle on lisätty "`pop`" -ohje suoritettavaksi vasta palautuksen jälkeen. Tähän perustuukin tekniikan pääajatus: on ilmiselvää, että funktio ei tee enää mitään palautuksen jälkeen. Tällä varmistutaan siitä, että kaikkia ohjeita ei tulla todellisuudessa koskaan suorittamaan. Useat takaisinkääntäjät eivät pysty kääntämään kyseistä koodia kaatumatta. Takaisinkääntäjä ei saa käännettyä tavukoodia lähdekoodiksi, sillä lähdekoodin rakenne ei vastaa kääntäjälle tuttua rakennetta. Valitettavasti saatavilla on niin paljon takaisinkääntäjiä, että sovelluksen suojaaminen niiden kaikkien varalta on mahdotonta.

Tämän lisäksi tunnetuimmat monimutkaistajat pitävät sisällään seuraavat tekniikat: suunnittelumallin-, hallinnan- ja tietojen monimutkaistukset.

Suunnittelumallin monimutkaistukset muokkaavat nimensä mukaan ohjelmiston sunnittelumallin rakennetta kahdella eri tavalla: se vaihtaa tunnisteiden nimet ja poistaa virheenkorjausilmoitukset. Tämä vähentää huomattavasti ohjelman informatiivisuutta ja takaisinkääntäminen vaikenee entisestään. Useimpia suunnitelumallien monimutkaistuksia ei pysty peruuttamaan, koska ne käyttävät "yksisuuntaisia" funktioita kuten tunnistuksien muuntamista sattumanvaraisiksi symboleiksi ja kommenttien, käyttämättömien metodien sekä virheenpaikannusilmoitusten poistamista. Vaikka suunnitelumallien monimutkaistuksilla ei pystytä estämään sovelluksen kääntämistä takaisin lähdekoodiksi, pystytään sillä ainakin lisäämään

takaisinkääntämisen kuluja. Suunnittelumallien monimutkaistukset ovat kaikista tunnetuimpia ja laajimmin käytettyjä monimutkaistajia. Lähes kaikki Java-ohjelmointikielessä tunnetut monimutkaistajat sisältävät tämän tekniikan.

Hallinnan monimutkaistukset muuttavat ohjelman hallintaa. Pääperiaate on hyvin yksinkertainen: monimutkaistaja luo esimerkiksi metodin `A()` lisäksi metodin `A_bug()` ja molemmat laitetaan `if` -lauseeseen.

```
1 if(PREDICATE) {
2 A_bug();
3 } else A();
```

KUVA 7. Hallinnan monimutkaistus

Kuvan 7 esimerkissä `PREDICATE` -muuttuja on suunniteltu niin, että se on aina epätosi(false) ja ohjelma valitsee `A()`:n virheellisen `A_bug()` sijasta. Tämä vaikeuttaa ohjelman toimintalogiikan ymmärtämistä.

Tietojen monimutkaistajat rikkovat tietorakenteita, joita ohjelmassa käytetään ja salaavat tunnisteet. Tämä metodi muokkaa muun muassa riippuvuussuhteita sekä rakentaa uudelleen taulukoita. Tietojen monimutkaistajat muuttavat perusteellisesti ohjelmiston tietorakenteita. Ne tekevät monimutkaistetusta ohjelmistosta niin vaikealukuisen, että se on lähes mahdotonta kääntää uudelleen lähdekoodiksi. /13/

4.1.3 Monimutkaistuksen hyödyt ja haitat

Monimutkaistettu ohjelmisto on takaisinkäännettynä vaikealukuista. Vaikka monimutkaistus ei estä sovelluksen takaisin lähdekoodiksi kääntämistä, tekee se parhaimmillaan ohjelmiston rakenteesta niin monimutkaista, että kääntämiseen kuluu kilpailijalta todella paljon resursseja. Monimutkaistuksessa ei häviä sovelluksen suorituskykyä eikä sovelluksen toiminnallisuutta. Luokkien, metodien ja muuttujien nimet muunnetaan lyhyemmiksi, sattumanvaraisiksi, merkeiksi ja käyttämättömät metodit poistetaan kokonaan. Tämän ansiosta tavukooditiedoston koko pienenee noin

20% riippuen käytetystä työkalusta.

Monimutkaistuksessa kaikki nimet vaihdetaan merkityksettömiin arvoihin. Kuka tahansa voi kääntää sovelluksen takaisin askel askeleelta. Tämä tapahtuu siten, että luodaan monimutkaistetuista paketeista UML -kaavioita ja annetaan oletusten perusteella merkityksellisiä nimiä kaikille paketeille, luokille ja elementeille. Monimutkaistajan poistaessa kaikki informatiiviset tulosteet on myös virheen paikannus huomattavasti vaikeampaa. Myös ohjelman logiikka ja algoritmit ovat luettavissa monimutkaistuksen jälkeen.

4.2 Koodin salaus

Tiedon salauksessa (encryption) selväkielinen teksti muunnetaan matemaattisten algoritmien ja avaimen avulla salattuun muotoon. Salattu tieto on kuitenkin mahdollista myös palauttaa takaisin ymmärrettävään muotoon purkamalla salaus (decryption). Tietojen salaus voidaan suorittaa siihen tarkoitetulla työkalulla. Salaus vähentää suorituskykyä ja lisää tavukooditiedoston kokoa. /3/

Hyvä esimerkki salaukseen perustuvasta työkalusta on ClassGuard, joka salaa Java-luokat 128 bittisellä AES -salauksella. AES -avain luodaan sattumanvaraisesti jokaisella työkalun käynnistyskerralla. Salattua sovellusta ajettaessa salauksen purkamisesta huolehtii purkamista varten suunniteltu luokanlataaja ennen kuin tavukoodi syötetään Java-virtuaalikoneelle. ClassGuard ottaa syötteenä jar -tiedostoja.

4.2.1 Salauksen hyödyt

Salauksen toteutus kuluttaa hyvin vähän ohjelmiston kehitykseen käytettäviä resursseja, sillä se on toteutettavissa myös itse ilmaiseksi. Salaus on mahdollista toteuttaa myös monelle eri käyttöjärjestelmälle.

4.2.2 Salauksen haitat

Salauksen suurin ongelma keskittyy suorituskykyyn. Luokkien, metodien ja muuttujien salaus syö erittäin paljon prosessorin tehoja. /4/ Tiedon salauksen purkaminen on yhtä raskasta prosessorille kuin itse salaaminenkin. Tämän lisäksi salauksen toteuttaminen ja ylläpito on erittäin hankalaa sekä aikaa vievää. Salausalgoritmien käyttö kasvattaa myös tavukooditiedostojen kokoa huomattavasti. Salaus laitteiston avulla on suositeltavampaa verrattaessa ohjelmiston salaukseen. Laitteistotason salauksessa on kuitenkin riski, että käyttäjä kaappaa viestin ja purkaa ohjelmiston salauksen. /5/ Laitteistosalauksen haitoiksi voidaan lukea myös ohjelmiston rajoitteinen siirrettävyys. Laite joka suorittaa salauksen, tai sen purkamisen, tulisi olla tuettu kyseisellä käyttöjärjestelmällä. Tämä ei kuitenkaan ole mahdollista kaikissa tilanteissa.

4.3 Natiivi konekieli

Just-In-Time eli JIT-kääntäjä kääntää dynaamisesti Java-tavukoodin suorituksen aikana ja se käyttää optimointitekniikoita nopeuttaakseen natiivin konekielen suorittamista. JIT -kääntäjä optimoi ja kääntää vain sen osan Java-ohjelman tavukoodista joka suoritetaan. Käyttämällä AOT-kääntäjää tavukooditiedosto on jo valmiiksi käännetty konekielelle. Konekielistä tavukoodia on mahdotonta automatisoidusti kääntää takaisin lähdekoodiksi.

AOT (Ahead-Of-Time)-kääntäjiä kutsutaan myös “staattisiksi kääntäjiksi” sekä “natiivikoodin kääntäjiksi”. Jälkimmäinen termi on näistä käytetympi, mutta samalla teknisesti katsottuna kauempana totuutta koska myös JIT-kääntäjä tuottaa natiivikoodia.

Työkalut, joilla sovellus on mahdollista kääntää käyttöjärjestelmille natiiviksi konekieleksi, perustuu tulkkina toimiviin kirjastoihin. Jokaiselle Java-ohjelmointikielestä löytyvälle funktiolle löytyy yhtäläinen funktio esimerkiksi C++ -ohjelmointikielestä. Kirjastot käyvät Java-sovelluksen luokat yksi kerrallaan läpi, rakentaen samalla yhtäläistä C++ -sovellusta. Työkalun AOT-kääntäjä ottaa syötteenä

jar -tiedostoja ja luokkatiedostoja, jonka jälkeen se tuottaa kohdekäyttöjärjestelmälle tavanaomaisen natiivin suoritettavan tiedoston. Tavanomaisella suoritettavalla tiedostolla tarkoitetaan esimerkiksi Windows-käyttöjärjestelmillä exe-päätteisiä tiedostoja. /16/

4.3.1 Konekielen hyödyt

JIT-kääntäjä toimii sovelluksen ajoympäristössä, joten se jakaa prosessorin ja muistin resurssit käännettävän sovelluksen kanssa. AOT-kääntäjä suoritetaan ohjelmistokehittäjän järjestelmässä, jolloin resursseja ja käännösaikaa ei mene hukkaan. Tämä tekee mahdolliseksi resurssien optimaalisen käytön. Hyöty moninkertaistuu, jos sovellusta ajetaan esimerkiksi sulautetussa järjestelmässä tai vanhemmassa PC:ssä, missä JIT-kääntäjillä ei ole tarpeeksi resursseja toimimiseen. Optimoivan AOT-kääntäjän tuottama natiivi konekoodi on yhtä vaikea kääntää takaisin lähdekoodiksi kuin C++ -kielellä ohjelmoidun sovelluksen. Tämän lisäksi suorituskykyä ei häviä.

Java-sovellukset kärsivät usein hitaasta käynnistymisestä. Jokaisella Java-sovelluksen käynnistymiskerralla koodi tulkitaan, profiloidaan ja käännetään JIT:llä. Tämän vuoksi Java-ohjelmien käynnistysajat ovat huomattavasti huonompia kuin samojen ohjelmien natiivikäännösten. Hitauden voi huomata myös verratessa vasteaikoja ensimmäisen sovelluksen käynnistyskerran sekä useamman käynnistyksen jälkeen tehtävän käynnistyksen välillä. Näiden syiden takia Java-ohjelmointikieltä pidetään edelleen hitaana vaihtoehtona. Natiivi suoritus tapahtuu suoraan laitteistossa ilman tulkkaus-, profilointi- ja käännösvälivaihetta. Se mahdollistaa nopeamman käynnistymisen ja parhaat vasteajat. Lisäksi AOT-kääntäjälle kehitellyt suoritettavat tiedostot ei ole riippuvaisia JRE:stä.

4.3.2 Konekielen haitat

Luokat, jotka sovellus lataa dynaamisesti ajonaikana, eivät välttämättä ole sovelluksen kehittäjän käytettävissä. Niitä voivat olla esimerkiksi jotkin kolmannen osapuolen laajennuksista, dynaamiset välityspalvelimet sekä muut luokat, jotka luodaan ajonaikana. Ajoympäristön tulee pitää sisällään Java-koodia varten tarvittavan JIT-kääntäjän. Luokat, jotka on ladattu joko järjestelmän tai sovelluksen luokkalataajan toimesta voidaan tavanomaisesti kääntää natiivikoodiksi. Sovellukset, jotka käyttävät mukautettuja luokkien lataajia, voidaan kääntää natiivikoodiksi vain osittain ellei AOT-kääntäjä ja ajoympäristö ole tietoisia kyseisten luokkalataajien toimintatavasta.

JIT-kääntäjällä on etulyöntiasema AOT-kääntäjiin siinä, että se voi valita koodin tuottamismallin laitteistolle sopivaksi. Se voi esimerkiksi käyttää Intel MMX/SSE/SSE2 laajennuksia nopeuttaakseen liukulukujen laskentaa.

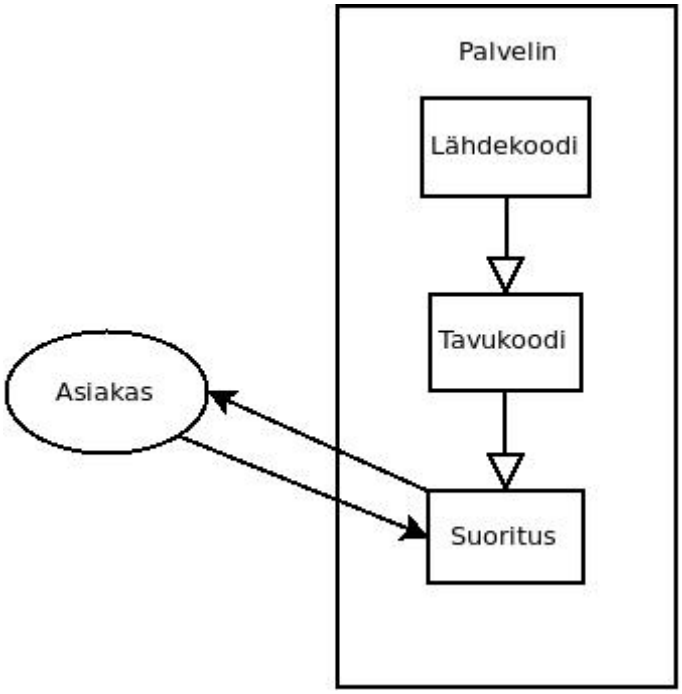
4.4 Muut suojausmahdollisuudet

Erilaisia suojausmahdollisuuksia on lukemattomia. Ohjelmistokehittäjän tulee pitää mielessään, että suojauksen voi suorittaa myös toteuttamalla ohjelmalle jonkin riippuvuuden, jota ilman sovellus ei toimi oikein. Riippuvuus voi olla joko laitteistokohtainen, esimerkiksi alaluvussa tarkemmin esitelty USB-lukkomoduuli tai osa ohjelmistosta, joka on suoritettu kokonaan palvelimella.

4.4.1 Ohjelman suoritus palvelimella

Yksi vaihtoehto suojauksen toteutukselle on siirtää osa ohjelmiston toiminallisuudesta palvelimen puolelle. Kuten kuva 8 havainnollistaa, käyttäjän sovellus lähettää palvelimelle pyynnön ohjelmiston suorituksesta ja palvelin palauttaa takaisin ainoastaan tuloksen käyttäjän sovellukseen, joka ainoastaan kuvaa tulokset käyttäjälle. Sovelluksen

takaisinkääntäminen on erittäin hankalaa, sillä sovelluksen toiminnalliseen osaan palvelimen päässä ei päästä fyysisesti käsiksi.



KUVA 8. Osa toiminnallisuudesta suoritettu palvelimella

Kun toiminnallisuutta siirretään palvelimen puolelle, ohjelmistoon käsiksi pääseminen hankaloituu huomattavasti. Suojaustekniikka vähentää myös käyttäjän sovelluksen kuormaa, sillä sovelluksen tehtävä on vain kuvata saamansa tulokset käyttäjälle. Lähes kaiken työn tekee siis palvelimen päässä oleva ohjelma.

Suojausmenetelmä tarvitsee pysyvän yhteyden käyttäjän sovelluksen ja palvelimen välillä. Yhteyden katkeaminen voi vaikuttaa koko sovelluksen suorituskykyyn sekä tekee käyttäjän sovelluksen hyödyttömäksi. Suojausmenetelmä on mahdollinen vain jos tietoliikenne sovelluksen ja palvelimen välillä on vähäistä.

4.4.2 Sovelluksen suojaus ulkoisella laitteella

Sovellus on mahdollista suojata myös ulkoisilla laitteilla kuten USB-lukkomoduulilla (dongle). Tämä tarkoittaa käytännössä sitä, että sovellusta on mahdotonta käyttää,

mikäli kyseinen USB-lukkomoduuli ei ole tietokoneen USB-portissa. Käyttäjä on kytkenyt USB-lukkomoduulin tietokoneeseen ja kaksoisklikannut binääritiedostoa, jonka haluaa suorittaa. Salauksen purkamiseen suunniteltu moottori käynnistyy, hakee salaisen avaimen USB-lukkomoduulilta, purkaa binääritiedoston ja lopuksi suorittaa sen. Käyttäjä ei ole tietoinen kyseisistä prosesseista, sillä ne ajetaan taustalla. Prosessit lisäävät viivettä sovelluksen käynnistysaikaan. Jos USB-lukkomoduuli irroitetaan sovelluksen ajonaikana USB-portista, sovellus huomaa sen ja lopettaa suorituksen. /13/

Lukkomoduuli saattaa esimerkiksi tarjota kahden tason suojauksen: salauksen purkamiseen tarkoitetun moottorin sekä sovellusosan, joka suorittaa laitteistotarkistuksen sovelluksessa. USB-lukkomoduulilla toteutettu suojaus on erittäin vaikea purkaa, sillä pelkällä binääritiedostolla takaisinkääntäjät eivät tee mitään vaan heidän tulisi pureutua myös USB-lukkomoduulin toimintaperiaatteisiin.

Käyttäjä on täysin riippuvainen USB-lukkomoduulista. Jos käyttäjä hukkaa USB -lukkomoduulin tai se vaurioituu esimerkiksi iskusta, sovellus on täysin hyödytön myös käyttäjälle. Tämä synnyttää erittäin suuren riskitekijän varsinkin sovelluksissa, joissa sovelluksen käyttäjä liikkuu USB-lukkomoduulin kanssa maastossa ja sen tulisi kestää kaikkia sääolosuhteita.

4.4.3 Sovelluksen suojaus vesileimalla

Yleisesti vesileimoja on käytetty muun muassa digitaalisissa kuvissa, musiikissa ja teksteissä. Vesileiman päätarkoitus on yksilöidä ohjelmiston todellinen suunnittelija. Kun sovelluksia käännetään takaisin lähdekoodiksi ja käytetään toisen suunnittelemia sovelluksia, on selvää, että ohjelmistosta kopiodaan kokonaisia lohkoja. Näin ollen, mikäli ohjelmiston uudelleenkäyttäjä ei ole täysin selvillä varastamiensa lohkojen toiminnasta, saattaa luokkiin olla piilotettuna vesileima. Vesileiman tehokkuus perustuu sovelluksen käyttäjän tietämättömyyteen sen olemassaolosta.

Tärkeiden luokkatiedostojen sekaan piilotettu ja salattu vesileima on sen suunnittelijan

toimesta helppo löytää. Vesileiman käyttö ei vaikuta ohjelmiston suorituskykyyn tai sen toimintaan. Vesileima ei todellisuudessa suojaa sovellusta varkaudelta. Vesileiman avulla voidaan saada varas vastuuseen, mutta jo toisen ohjelmiston tutkiminen vaatii varmoja todisteita.

4.5 Suojaustapojen soveltuvuus kyseiselle sovellukselle

Kuten huomata saattaa, jokaisella suojaustavalla on niin hyödyt kuin haitatkin. Mikään suojaustavoista ei ole myöskään täysin ongelmaton, sillä suojauksen kohteena oleva sovellus on monen eri tekniikan summa.

Ohjelman suoritus palvelimella: Sovellus asennetaan ympäristöön, joka ei kykene olemaan yhteydessä palvelimeen. Myös verkon kuormitus tulee olemaan parhaimmillaan erittäin raskasta sovelluksen käytössä.

Koodin salaus: Vaatisi liikaa ohjelman uudelleen kirjoittamista ja ylläpito olisi liian aikaavievää. Salausalgoritmien käytöstä tulisi tehdä päätös jo sovelluksen suunnitteluvaiheessa. Tällöin sovelluksen toteutuksessa voitaisiin käyttää salauksen tukemia ratkaisumalleja. Tämän lisäksi sovelluksen pyörittämiseen käytettävän tietokoneen minimivaatimukset tulisivat luultavasti kasvamaan.

USB -lukkomoduuli: Ei tuo juurikaan lisää turvallisuutta yksinään. Jos tavukooditiedosto saadaan takaisinkäännettyä lähdekoodiksi, voidaan avaimen kysely kommentoida lähdekooditiedostosta kokonaan pois. Tällöin USB -lukkomoduuli on tarpeeton.

Vesileima: Ei suojaa tiedostoa takaisinkääntämiseltä, joten tapa jätetään tässä vaiheessa tarkastelun ulkopuolelle.

5. RATKAISUMALLINA MONIMUTKAISTUS

Monimutkaistus valittiin ensimmäiseksi kokeiltavaksi ratkaisumalliksi puhtaasti sen takia, että se on yleisin takaisinkääntämistä hankaloittava suojaustapa. Internetistä löytyy useita monimutkaistukseen tarkoitettuja työkaluja. Ennen monimutkaistuksen suoritusta täytyy kuitenkin tutustua muutamaan eri monimutkaistuksen toteuttavaan työkaluun, jotta tiedetään valita parhaiten projektiimme yhteensopiva monimutkaistaja. /7/

5.1 Monimutkaistajien vertailu

Internet on pullollaan monimutkaistajia, joista osa on ilmaisia ja osa hyvinkin kalliita. Erona ilmaisen ja kaupallisen version välillä on ainoastaan muutama toiminnallisuus, jotka saattavat parhaassa tapauksessa olla hyödyttömiä juuri monimutkaistustyön alla olevalle projektille. Vertailuun valittiin kolme kaupallista sekä yksi ilmainen monimutkaistaja. Vertailtavien monimutkaistajien valintaan vaikutti toimintontojen määrä sekä sen ajantasaisuus. Taulukossa 2 on vertailtu monimutkaistajia sekä niiden ominaisuuksia.

TAULUKKO 2. Monimutkaistajien vertailu /17/ /18/

Monimutkaistaja	**Kokeilussa**	**Lisenssi**	**Mainostetut toiminnot**
DashO-Pro v6.3.3	kokeiluversio	$895.00	Muuttujien nimeäminen, merkkijonojen salaus, metatietojen monimutkaistus, informatiivisten tulosteiden poisto, kyky muuntaa tietorakennetta
Zelix KlassMaster v5.3.1	kokeiluversio	$399.00	Muuttujien nimeäminen, merkkijonojen salaus, älykäs tallennus, informatiivisten tulosteiden poisto, kyky muuntaa tietorakennetta, Rivinumeroinnin poisto
RetroGuard (Lite) v2.3	Ilmainen (ei-ammattikäyttöön)	$139.00	Muuttujien nimeäminen, informatiivisten tulosteiden poisto
ProGuard v4.4	Ilmainen	0$	Muuttujien nimeäminen, informatiivisten tulosteiden poisto, rivinumeroinnin poisto, käyttämättömien luokkien, kirjastojen ja muuttujien poisto

5.2 Mahdolliset ongelmat ja niiden yleiset ratkaisut

Monimutkaistuksen pitäisi olla turvallinen tapa suojella sovelluksen toiminnallisuutta. On kuitenkin tapauksia, joissa koodin monimutkaistus saattaa vahingossa rikkoa toimivan sovelluksen. Seuraavassa kappaleessa tarkastelemme yleisimpiä ongelmia ja niihin suositeltuja ratkaisuja.

5.2.1 Dynaaminen luokan lataaminen

Pakettien, luokkien, metodien ja muuttujien uudelleen nimeäminen toimii hyvin niin kauan kuin nimi on muutettu pysyvästi kauttaaltaan koko järjestelmään. Monimutkaistaja varmistaa, että jokainen staattinen viittaus tavukoodin sisällä on päivitetty vastaavaan uuteen nimeen. Koodi voi tosin suorittaa dynaamisen luokan latauksen käyttäen esimerkiksi Class.forName() tai ClassLoader.loadClass() -metodeita ohittaen alkuperäisen luokan nimen. Tämä voi johtaa ClassNotFound -poikkeukseen. Modernit monimutkaistajat ovat melko hyviä käsittelemään kyseisiä tapauksia, yrittäen vaihtaa merkkijonon vastaamaan uutta nimeä. Jos merkkijono on luotu ajonaikana tai luettu properties -tiedostosta, monimutkaistaja ei siltikään pysty käsittelemään tapausta. Hyvät monimutkaistajat kuitenkin tuottavat vianselvitysraportin suorituksesta, joka opastaa ajonaikaisten ongelmien jäljille. Paras tapa käsitellä kyseiset ongelmat on jättää luokat, jotka aiheuttavat ongelmia, kokonaan monimutkaistamatta.

5.2.2 Heijastus (Reflection)

Heijastusta käyttävät yleisesti ohjelmat, joilla on tarve tutkia tai muutella Java -virtuaalikoneessa ajettavan sovelluksen ajonaikaista käyttäytymistä. Heijastukset vaativat käännön aikana tiedon metodien ja kenttien nimistä, joten se vaikuttaa myös monimutkaistukseen. Kuten edellisessäkin ongelmassa myös tässä on etua monimutkaistajan hyvästä vianselvitysraportista. Jos ajonaikainen virhe johtuu

monimutkaistuksesta, tulee metodien ja kenttien nimet, joihin viitataan Class.getMethod- tai Class.getField -metodeissa, jättää kokonaan monimutkaistamatta.

5.2.3 Sarjallistaminen (Serialization)

Sarjallistamisella tarkoitetaan Java -olion muuntamista tiedostovirtamuotoon, jossa se voidaan esimerkiksi tallentaa tai lähettää. Jos luokan versio tai sen rakenne muuttuu, takaisin sarjallistamisesta (deserialization) saattaa seurata poikkeus. Monimutkaistettu luokka on mahdollista sarjallistaa ja muuntaa takaisin. Takaisinsarjallistaminen ei tule kuitenkaan onnistumaan, mikäli takaisinsarjallistamisen toteuttava luokka on monimutkaistettu ja sarjallistettu luokka on monimutkaistamaton. Tämä ei kuitenkaan ole onneksi yleinen ongelma. Tämä voidaan kuitenkin ratkaista jättämällä sarjallistetut luokat monimutkaistamatta ja välttämällä sarjallistettujen luokkien sekoitusta.

5.2.4 Nimeämiskäytäntöjen rikkominen

Metodien uudelleen nimeäminen voi vahingoittaa suunnitelumalleja kuten Java-papuja (EJB), joissa pavun suunnittelijan edellytetään käyttävän metodeja tietyillä nimillä ja allekirjoituksilla. Papujen niin sanotut soittopyyntömetodit(callback), kuten ejbCreate ja ejbRemove, ei ole määritelty super -luokan tai rajapinnan avulla. Allekirjoitetut soittopyyntömetodit luovat papujen määrittelyt. Muuttamalla soittopyyntömetodien nimiä rikotaan nimeämiskäytäntöä ja samalla tehdään pavuista hyödyttömiä. Kyseisien metodien nimet tulisi aina jättää monimutkaistamatta. /7/

5.2.5 Ylläpidon vaikeudet

Monimutkaistus tekee myös vian määrittelemisen ja etsimisen sovelluksesta vaikeammaksi. Java-ohjelmointikielen poikkeuksien käsittely on tehokas tapa eristää virheellinen koodi. Katsomalla poikkeuksen osoittaman virheen rivinumeroa saadaan

hyvä käsitys missä ja mikä meni väärin. Pitämällä nämä informatiiviset tulosteet lähdekoodissa, saadaan tiedoston nimien ja rivinumeroinnin perusteella ohjelmisto raportoimaan virheen tarkka sijainti. Mikäli monimutkaistus suoritetaan huolimattomasti, voidaan tämä etu menettää. Tällöin suunnittelija näkee pelkästään monimutkaistetut luokkien nimet oikeiden luokkien nimien ja rivinumeroinnin sijaan.

5.3 Monimutkaistuksen käytettävyyden arviointi

Testauksen kohteeksi valittiin aluksi ilmainen ProGuard v4.4 ja sen jälkeen kaupallinen Zelix KlassMaster v5.3.1. Molemmat yritykset kuitenkin kuivuivat kasaan samaan ongelmaan useiden tuntien työn jälkeen. Kuten aiemmassa kappaleessa ilmeni, täytyy suuri osa sovelluksesta jättää monimutkaistamatta. Tämän lisäksi JSP:llä toteutettua osaa sovelluksesta ei voida monimutkaistaa. Tämä johti siihen, että valtaosa sovelluksesta jäi monimutkaistamatta ja sovelluksen takaisinkääntäminen luettavaksi lähdekoodiksi näiltä osin on erittäin helppoa.

Tekemällä muutoksia valtaosaan sovelluksen luokista, voitaisiin teoriassa saada suurempi osa sovelluksesta monimutkaistettua, mutta tämä vaatisi viikkojen ehkä jopa kuukausien työn yhdelle ihmisille. Näin ollen tuotekehityksessä edelleen olevaan sovellukseen, jonka rakenteeseen tulee jatkuvasti suuria muutoksia, on tässä vaiheessa turha ruveta tekemään muutoksia monimutkaistusta varten. Sovelluksen rakenteen muuntaminen monimutkaistamista varten on turhaa, koska rakenne joudutaan luultavasti taas uudelleenkirjoittamaan siinä vaiheessa kun sovellus on valmis ja se luovutetaan asiakkaalle.

6 RATKAISUMALLINA KÄÄNNETTÄVÄT OHJELMOINTIKIELET

Java-sovelluksen ja käyttöjärjestelmäkohtaisen natiivin ohjelmointikielen rajapinnat ovat erittäin vaikeita määritellä sekä rajapintojen liitokset ovat todella virheherkkiä. Tämän takia ratkaisumalli toteutettiin sovellukselle kokonaisuudessaan. Tällöin koko sovellus on muunnettu käyttöjärjestelmäkohtaiseksi natiiviksi konekoodiksi eikä rajapinnoista tarvitse välittää. Tutkimustyö käsittelee sekä ilmaisia, että kaupallisia vaihtoehtoja sovelluksen kääntämiseksi konekielelle.

6.1 Kääntävien työkalujen vertailu

Työkaluja, jotka kääntävät staattisesti Java-luokat tai tavukooditiedoston natiivikieleen löytyy erittäin vähän. Työkaluja löytyi yhteensä kuusi. Näistä kuudesta työkalusta kuitenkin neljä, JNC, Manta, Vortex ja Timber, olivat kelpaamattomia nykyisen Java-ohjelmien kehitys- ja ajoympäristön kanssa. Tähän tarkoitukseen tehtyjä ajantasalla olevia kääntäjiä löytyi vain kaksi: GNU GCJ ja Excelsior JET.

Nämä kaksi jäljelle jäävää työkalua ovat melkein toistensa vastakohdat. Seuraavassa kappaleessa tarkemmin esiteltävä GNU GCJ on ilmainen hyvinkin tarkasti konfiguroitavissa oleva AOT -kääntäjä, joka ei tarjoa minkäänlaisia lisäominaisuuksia.

Excelsior JET on kaupallinen automatisoitu työkalu ja sisältää paljon lisäominaisuuksia kuten muun muassa automatisoidun asennustiedoston toteuttamisen. Työkalun käyttö on nopea oppia sekä se toimi luotettavasti tutkimuksissani. Excelsior JET:llä on laaja ja hyvin arvovaltainen kuluttajakunta, joka varmistaa tuotteen jatkuvan kehityksen sekä tuotteelle luotettavan asiakaspalvelun. /14/

6.2 GNU kääntäjä Java-ohjelmointikielelle (GCJ)

GCJ (The GNU Compiler for the Java Programming Language) on siirreltävissä oleva optimoiva AOT-kääntäjä Java-ohjelmointikielelle. Se osaa kääntää Java-lähdekoodin tavukoodiksi tai suoraan natiiviksi konekoodiksi sekä Java-tavukoodin natiiviksi konekoodiksi. Käännetyt sovelluksen linkitetään GCJ:n ajoympäristöön (libgcj), joka tarjoaa tavanaomaiset luokkakirjastot, roskienkerääjän ja tavukooditulkin. Libgcj pystyy dynaamisesti lataamaan ja tulkitsemaan luokat antaen tuloksena sekoituksen käännetystä ja tulkatusta sovelluksesta. GCJ on yhdistetty GNU Classpathin kanssa. GNU Classpath on GNU-projekti, jonka tarkoituksena on luoda Java-ohjelmointikielen kääntäjille ja virtuualikoneille keskeiset luokkakirjastot ilmaiseksi. /15/

6.2.1 GNU GCJ-ympäristö

GNU GCJ-ympäristön asentaminen päätettiin suorittaa Linux-käyttöjärjestelmälle, jolle se on suunniteltu. GCJ-kääntäjän saa kuitenkin asennettua myös Windows-ympäristöön käyttäen avuksi MinGW:tä. GCJ:tä käyttävän Java-ympäristön asennus Linux-käyttöjärjestelmälle on erittäin helppoa. Pakettien hallinnasta löytyy suoraan *java-gcj-compat* -asennuspaketti, joka sisältää kokoelman kääntämiseen tarvittavia tiedostoja.

Java-ohjelman kääntäminen GCJ:n avulla tapahtuu samalla tavalla kuin GCC:llä C- tai C++ -kielisen ohjelman kääntäminen. Kääntämisen voi jakaa kahteen erilliseen vaiheeseen: käännös ja linkitys. Käännösvaiheessa jar -tiedostosta luodaan o -päätteinen käynnistettävä ja linkitettävä tiedosto. Linkitysvaiheessa luodulle tiedostolle kerrotaan sovelluksen luokka, josta sovelluksen ajaminen tyypillisesti aloitetaan. Vaiheiden tuloksena syntyy käyttäjärjestelmäkohtainen natiivi tavukooditiedosto.

6.2.2 GNU GCJ:n käytettävyyden arviointi

GCJ-kääntöä varten tehty yksinkertainen, kaiken kaikkiaan kolme luokkaa sisältävä, testausohjelma näytti kääntyvän ongelmitta. Siirryttäessä yrityksen sovelluksen kääntämiseen, ongelmat kuitenkin alkoivat. Usean tunnin työn jälkeen lopputulos oli, että sovellukselle tarpeellisten riippuvuuksien lisäily yksi kerrallaan oli erittäin työlästä. Tämän lisäksi tehtiin havainto, että GCJ ei tue kaikkia edes JDK 1.5:ssä olleita ominaisuuksia, kuten esimerkiksi geneeristen tyyppien käyttöä.

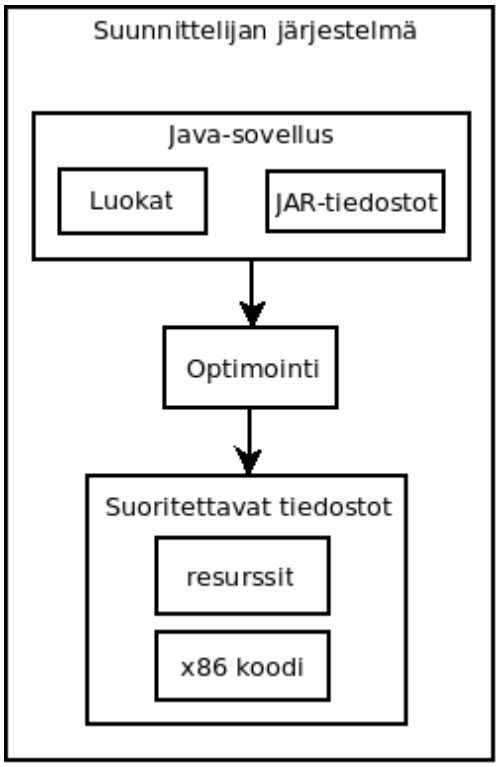
Ominaisuuksien tuen puute on yleinen vika ilmaisten työkalujen käytössä, varsinkin jos työkalun kohteena oleva ohjelmointikieli on nopeassa kehityksessä. Vaikka GCJ:hin saa päivityksiä melko usein, ei GCJ:tä ole saatu pidettyä talkoovoimin Java-kehitysympäristön nopeassa kasvussa mukana. GCJ on liian työläs vaihtoehto sovelluksen suojaamiseen. Jos GCJ:n käyttö saisi kannatusta sovelluksen suojaustavaksi, tarkoittaisi se suurilta osin sovelluksen uudelleenkirjoittamista. Tuotekehityksessä oleva sovellus kokee joka tapauksessa suuria muutoksia vielä ennen sen luovutusta asiakkaalle. Tämän takia nyt GCJ:llä tehtävä suojaus ei olisi enään käyttökelpoinen luovutusajankohtana. On otettava myös huomioon, että sovelluksen luovutuksen takarajan lähestyessä ei välttämättä ole paljoa aikaa uhrattavaksi sovelluksen kopiosuojaukselle.

6.3 Excelsior JET -kääntäjätyökalu

Excelsior JET on kaupallinen Java SE teknologiaan perustuva sekä AOT-kääntäjän ympärille rakennettu työkalu, joka mahdollistaa sovelluksen kääntämisen konekoodiksi. Kääntäjä muuntaa siirrettävissä olevan Java-tavukoodin laitteistolle ja käyttöjärjestelmälle optimoiduksi suoritettavaksi tiedostoksi. Käännetyssä tiedostossa on Java-ajoympäristö kokonaisuudessaan mukaanlukien JIT-kääntäjä. JIT-kääntäjä mahdollistaa luokkien käsittelyn, joita ei jostain syystä haluta kääntää pakettiin mukaan. Excelsior JET on läpäissyt Sun Microsystemsin käyttämän tekniikoiden yhteensopivuutta testaavan kokeen (JCK) Java SE 6:lle. Se on myös sertifioitu Sun

Microsystemsin toimesta useille Windows- ja Linux-käyttöjärjestelmille, jotka käyttävät Intel x86 -yhteensopivia laitteita. Kaiken kaikkiaan Excelsior JET sisältää kolme ratkaisumallia.

Suorituskyvyn optimointi, joka mahdollistaa suorituskyvyn parantamisen ilman lähdekoodiin tehtäviä muutoksia tai laitteistopäivityksiä. Java-sovellus tyypillisesti sisältää luokkatiedostoja ja resurssitiedostoja, jotka on pakattu yhteen tai useampaan jar-tiedostoon. Kuten kuva 9 havainnollistaa, Excelsior JET:n avulla voidaan muuntaa luokat optimoiduksi x86-koodiksi ja luoda natiivisti suoritettava tiedosto Windows:lle tai Linux:lle. Alkuperäisiä Java-luokkia ei tarvita sovelluksen ajamiseen.



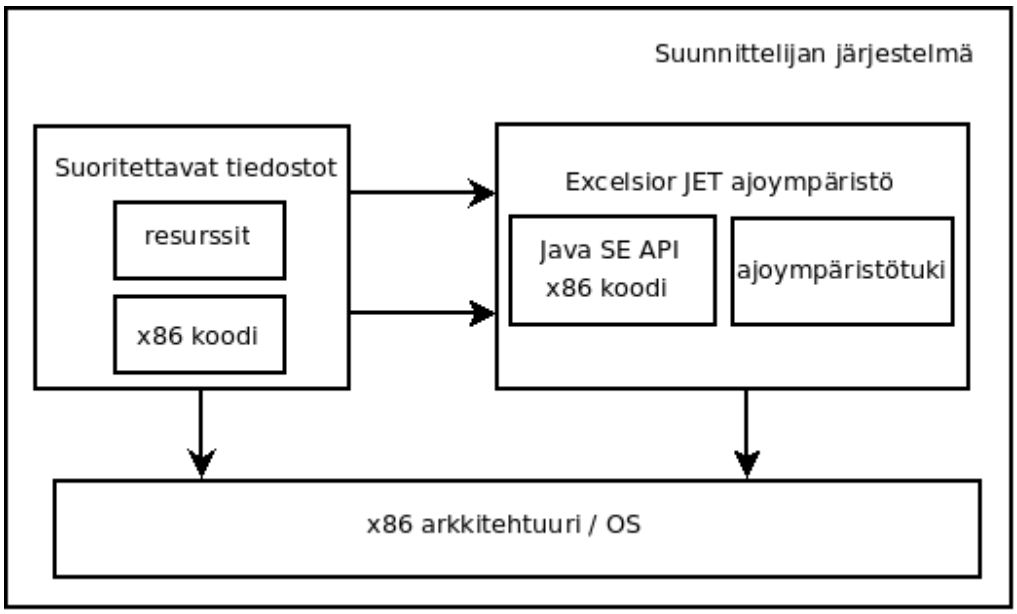
KUVA 9. Suorituskyvyn optimointi

Monimutkaistus, joka suojaa sovellusta takaisinkääntäjiltä ilman sovelluksen suorituskyvyn heikkenemistä.

Sovelluksen asennustiedoston luonti, jolla on mahdollista luoda pienikokoinen ja ammattimainen asennuspaketti ilman riippuvuuksia Java-ajoympäristöön. Asennuspaketti sisältää kaiken tarvittavan: käännetyn sovelluksen, Excelsior JET

-ajoympäristön sekä muut tarvittavat tiedostot, jotka halutaan toimittaa kätevästi asiakkaalle.

Kuva 10 havainnollistaa kuinka sovelluksen suoritettavat tiedostot yhdistyvät Excelsior JET -ajoympäristöön, joka tarjoaa J2SE API -luokat ja tuen tarpeellisille Java-sovelluksen rutiineille kuten roskienkeräämiselle. Tämä mahdollistaa Java-sovelluksen ajamisen suoraan laitteiston päällä.



KUVA 10. Ajoympäristö tarjoaa käännetyn tiedoston

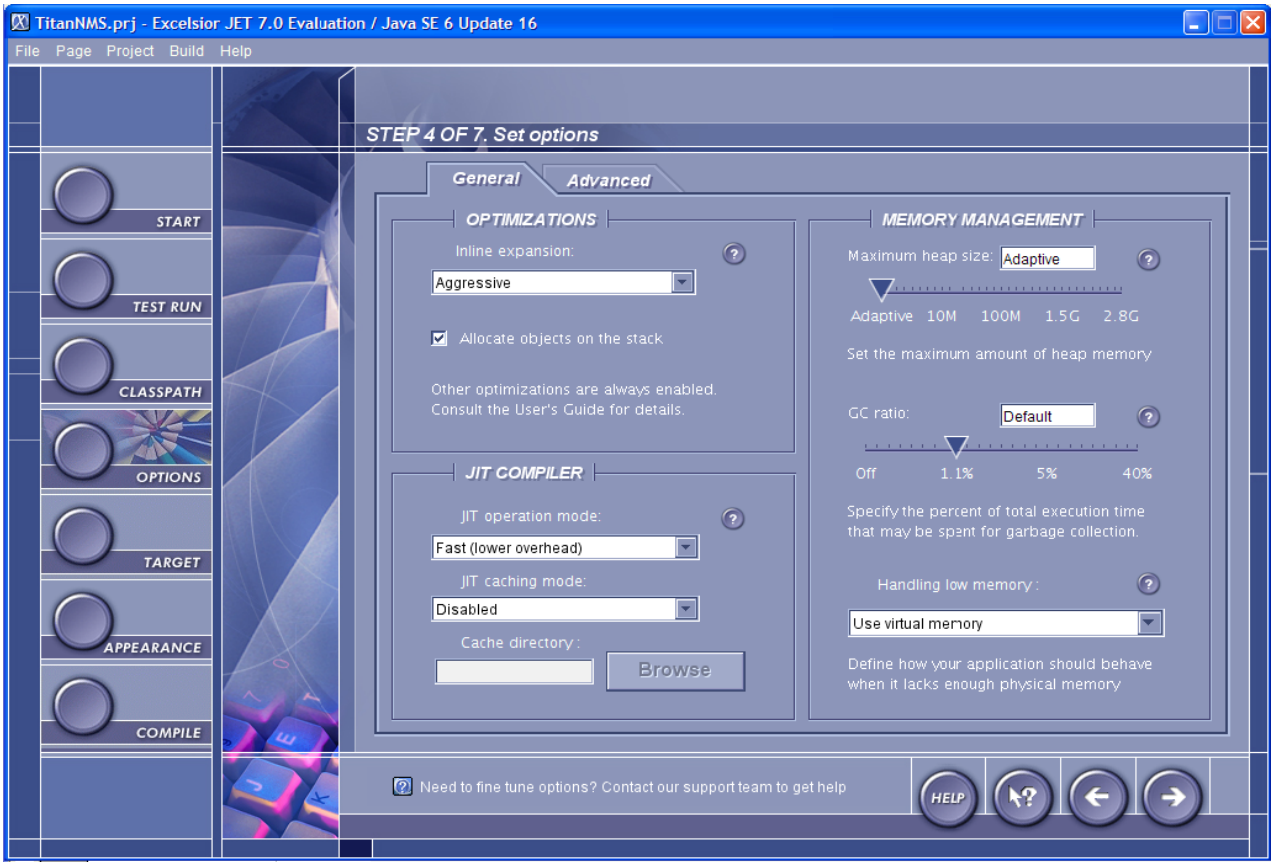
Vaikka kopiosuojattava sovellus onkin toteutettu J2EE -teknologialla, palveli kyseinen työkalu tarkoitusta. Jokainen J2EE:llä toteutettu sovellus ajetaan J2SE:n virtuaalikoneen päällä. Tämän takia Excelsior JET oli sopiva myös tämän J2EE-sovelluksen kääntämiseen ja ajamiseen. /16/

6.3.1 Excelsior JET -ympäristö

Excelsior JET:stä löytyy asennuspaketit niin Windows- kuin Linux-käyttöjärjestelmille. Eroavaisuus paketeissa syntyy siinä, että käännettäessa Windowsille natiivia exe-

tavutiedostoa, täytyy käännös tapahtua myös Windows-käyttöjärjestelmässä. Sama koskee myös Linux-käyttöjärjestelmän natiiveja konekooditiedostoja. Käyttöjärjestelmien kesken ristiin kääntäminen ei ole mahdollista. Asennus suoritettiin aluksi Windowsille ja mikäli sovelluksen kääntäminen onnistuu, toistetaan asennus myös Linux -käyttöjärjestelmälle. Kääntämiskokeilu suoritettiin Excelsior JET:n sivuilta löytyvällä työkalun arviointiin tarkoitetulla testauspaketilla, jonka saa kokeilukäyttöön 90 päiväksi. Kääntämisessä käytettiin myös Excelsior JET:n standardin version asetuksia. Tuotteesta löytyy myös versiot ammattilaisille ja yrityksille, joissa lisämaksua vastaan saa käyttöönsä lisäominaisuuksia.

Windows-käyttöjärjestelmälle ympäristön asentaminen on hyvin suoraviivaista. Sovelluksen kääntäminen koostuu seitsemästä ohjatusta askeleesta. Kuvassa 11 nähdään asetus -askeleen määrittely.



KUVA 11. Käännettävän sovelluksen asetusten määrittely

6.3.2 Excelsior JET:n käytettävyyden arviointi

Excelsior JET:n avulla sovelluksen suojaus kävi melko vaivattomasti molemmissa Linux- ja Windows-käyttöjärjestelmissä. Käännetyn sovelluksen ajaminen kävi moitteitta järjestelmässä, jossa sovelluksen kääntö oli tapahtunut. Aluksi hämennystä aiheutti käännetyn sovelluksen ajaminen järjestelmässä, josta Excelsior JET -ajoympäristöä ei löydy. Tämän lisäksi sovelluksen käyttämien kirjastojen kanssa oli ongelmia. Ongelmat kuitenkin ratkesivat lisäämällä kirjastot asennuspakettiin ja lisäämällä käynnistettävän tiedoston ominaisuuksiin kirjaston sijainti (java.library.path=lib).

Työkalussa on mahdollista valita käännetäänkö sovellus kokonaisuudessaan yhdeksi konekieliseksi tiedostoksi vai suoritetaanko käännös komponenttitasolla jokaiselle komponentille erikseen. Tutkimustyöni sovellukselle oli mahdollista suorittaa vain käännös kokonaisuudessaan. Tämä tarkoittaa samalla myös sitä, että työkalun monimutkaistusta ei saatu suoritettua onnistuneesti.

Sovelluksen koko tulee kääntämisen myötä kasvamaan monikertaiseksi. Sovelluksen jar -tiedoston koko ennen kääntämistä on noin 24 megabittiä ja kääntämisestä syntynyt exe -tiedosto on 112 megabittiä. Kasvanut tiedoston koko kuitenkin vapauttaa sovelluksen laitteistovaatimuksista JRE:n, sillä se kuuluu pakettiin mukaan.

Excelsior JET:llä on asiakkaina suuri joukko tekniikan alan kärki yrityksiä. Tämän takia pelko Excelsior -yrityksen äkillisestä häviämisestä markkinoilta on erittäin pieni. Mikäli Excelsior JET:n tuotekehitys ja ylläpito kuitenkin jostain syystä katkeaa, suojattavalle sovellukselle on varmasti siinä vaiheessa jo kerinnyt muodostua pysyvä rakenne. Tällöin voidaan suunnitella ajan kanssa sovelluksen suojausta myös esimerkiksi monimutkaistuksen avulla.

7 YHTEENVETO

Java-sovelluksen suojaamiseen löytyy useita eri suojaustapoja. Harkittaessa sovelluksen suojaukseen käytettävää tekniikkaa ja työkalua on aluksi hyvä pohtia suojattavan sovelluksen rajoitteita. Suojattava sovellus nimittäin sanelee loppujen lopuksi melko tarkalleen, mikä suojaustapa on kyseisellä hetkellä paras.

Tässä tapauksessa Excelsior JET vaikuttaa parhaalta tavalta suojata sovellus takaisinkääntäjiltä. Excelsior JET pystyy kääntämään Java-sovelluksen käyttöjärjestelmän konekielelle, jonka vuoksi sovellusta ei ole vaivanarvoista yrittää kääntää takaisin lähdekoodiksi arvokkaan informaation toivossa. Lisäksi Excelsior JET osaa hoitaa myös monimutkaistuksen, vaikka siitä ei hyödyttykään tutkimuksen kohteena olleessa sovelluksessa. Monimutkaistuksen tuki tuo kuitenkin lisäarvoa yrityksen muiden sovellusten suojaamisessa. Tämän lisäksi Excelsior JET:n avulla sovelluksen saa paketoitua ammattimaiseksi asennuspaketiksi vaivatta.

Excelsior JET on luotettavuudellaan ansainnut Sun Microsystemsin sertifikaatin. Työkalun maineesta kertoo myös sen laaja käyttäjäkunta, johon kuuluu useita suuria yrityksiä kuten muun muassa Hitachi, Mazda, Honda, NASA, Siemens, Toyota, Toshiba ja Cisco.

Excelsior JET:n standardiversion hinnaksi kohteena olleen sovelluksen suojaamiselle tulisi 2500$/käyttäjä, koska työkalusta joudutaan ostamaan sekä Windows- että Linux-käyttöjärjestelmille sopivat versiot. Tämä kuitenkin tuntuu varsin kohtuulliselta hinnalta, ottaen huomioon työntekijöiltä suojaamiseen kuluvat työtunnit sovellusta kohti. Kääntämisten lukumäärään liittyviä rajoitteita työkalulla ei ole. Tulevien sovellusten kopiosuojaus vaivattomasti Excelsior JET:n avulla saattaisi tuoda yllättävää lisäarvoa tuleville työtarjouksille, vaikka sitä ei varsinaisissa sovelluksen vaatimuksissa olisikaan.

Lähteet

Kirjalliset lähteet

/1/ Pistoia, M., Nagaratnam, N., Koved, L. & Nadalin, A. 2003. Enterprise Java Security: Building Secure Java Application. Boston: Addison Wesley.

/2/ Oaks, S. 2001. Java Security. 2. painos. Sebastopol: O' Reilly Media.

/3/ Weiss, J. 2004. Java Cryptography Extensions: Practical Guide for Programmers. San Francisco: ElsevierSan

/4/ Boyd, C. 2003. Information Security, New York: Springer.

/5/ Galbreath, N. 2002. Cryptography for Internet & Database Applications. New York: John Wiley & Sons, Inc.

/6/ Gong, L., Ellison, G. & Dageforde, M. 2003. Inside Java 2 Platform Security: Architecture, API Design, and Implementation, Second Edition. California: Addison Wesley.

/7/ Kalinovsky, A. 2004. Covert Java: Techniques for Decompiling, Patching, and Reverse Engineering. Indianapolis: Sams Publishing.

/8/ Helton, R., & Helton, J. 2002. Java Security Solutions. Indianapolis: Wiley Publishing, Inc.

/9/ Hook, D. 2005. Beginning Cryptography with Java. Melbourne: Wrox Press.

Sähköiset lähteet

/10/ Viestintävirasto. [www-sivu][viitattu 20.2.2010].
http://www.ficora.fi/index/palvelut/palvelutaiheittain/tietoturva.html

/11/ Yliopistojen tietoturva. [www-sivu][viitattu 20.2.2010].
http://www.yliopistojentt.fi/VAHTI-CD/Sivusto/faq/PKI.htm

/12/ TechRepublic. [www-sivu][viitattu 27.2.2010].

http://blogs.techrepublic.com.com/programming-and-development/?p=444

/13/ Palisade. [www-sivu][viitattu 13.3.2010].

http://palisade.plynt.com/issues/2008Jul/defend-reverse-engineering

/14/ The GNU Compiler for the Java. [www-sivu][viitattu 31.3.2010].

http://gcc.gnu.org/java/

/15/ GNU Classpath. [www-sivu][viitattu 31.3.2010].

http://www.gnu.org/software/classpath/

/16/ Excelsior JET. [www-sivu][viitattu 31.3.2010].

http://www.excelsior-usa.com/jet.html

/17/ Zelix KlassMaster. [www-sivu][viitattu 31.3.2010].

http://www.zelix.com/klassmaster/

/18/ PreEmptive Solutions. [www-sivu][viitattu 31.3.2010].

http://www.preemptive.com/products/dasho/

/19/ Mocha, the Java Decompiler. [www-sivu][viitattu 31.3.2010].

http://www.brouhaha.com/~eric/software/mocha/

/20/ Sun Developer Network. [www-sivu][viitattu 31.3.2010].

http://java.sun.com/javase/6/

/21/ Sun Developer Network. [www-sivu][viitattu 31.3.2010].

http://java.sun.com/javaee/

/22/ Sun Developer Network. [www-sivu][viitattu 31.3.2010].

http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp

/23/ GlassGuard. [www-sivu][viitattu 31.3.2010].

http://www.jsecurity.net/classguard/index.html