



TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan koulutusohjelma

TIMO HIRVONEN

TAKAISINMALLINNUKSEN ESTOTEKNIIKAT

Kandidaatintyö

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan koulutusohjelma

HIRVONEN, TIMO: Takaisinmallinnuksen estotekniikat

Kandidaatintyö, 18 sivua, 10 liitesivua

Tammikuu 2008

Pääaine: Ohjelmistotuotanto

Tarkastaja: professori Mikko Tiusanen

Avainsanat: Takaisinmallinnus, takaisinmallinnuksen esto, koodin monimutkaistaminen

Ohjelmistot ja niiden toteutusyksityiskohdat tarvitsevat suojauksia laitonta kopiointia ja kilpailijoita vastaan. Näitä suojauksia kierretään takaisinmallinnuksen avulla. Tämä työ käsittelee takaisinmallinnuksen estotekniikoita, joista esitellään sekä perusidea että yksityiskohtainen esimerkki. Näiden erilaisten estotekniikoiden keskinäinen vertailu antaa erinomaisen lähtökohdan tekniikan soveltuvuuden arvioimiseksi oman ohjelmiston suojaustarpeisiin.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Programme in Information Technology

HIRVONEN, TIMO: Antireversing Techniques

Bachelor of Science Thesis, 18 pages, 10 Appendix pages

January 2008

Major: Software engineering

Examiner: Professor Mikko Tiusanen

Keywords: Reverse engineering, antireversing, code obfuscation

Software and its implementation details need protection against software piracy and competitors. These protections are circumvented using reverse engineering. This thesis provides both the basic idea and a detailed example of multiple antireversing techniques. The comparison of these diverse techniques offers an excellent basis for selecting the most suitable technique for protecting one's application.

SISÄLLYS

1. Johdanto.....	1
2. Takaisinmallinnuksen esto	2
3. Tekniikoita.....	4
3.1. Datan suojaaminen	4
3.2. Inhimilliset tekijät	7
3.3. Suoritettavan tiedoston tekninen suojaaminen.....	8
4. Tulosten arviointi	13
5. Yhteenveto.....	16
Lähteet.....	17
LIITE 1: Esimerkkiohjelma (Leoniak 2007c).....	19
LIITE 2: Kryptaus (Eilam 2005b)	25
LIITE 3: Salauksen purku (Eilam 2005b).....	27

TERMIT JA NIIDEN MÄÄRITELMÄT

Antidebugger	Estetään ohjelman ajaminen debuggerissa (Eilam 2005a, s. 329).
BIOS	Basic Input/Output System. Koodi, joka suoritetaan tietokoneen käynnistyksen yhteydessä. Tunnistaa ja alustaa laitteet niin, että käyttöjärjestelmä voidaan ladata. (Mueller 2006.)
Debugger	Ohjelma, jossa voidaan ajaa ja analysoida toisia ohjelmia. Mahdollistaa yleensä ohjelman suorittamisen käsky kerrallaan, keskeytyskohtien asettamisen, muuttujien arvojen jäljittämisen ja ohjelman tilan muokkaamisen. (Eilam 2005a, s. 116–119.)
Disassembler	Ohjelma, joka tuottaa konekoodista symbolista konekieltä (Eilam 2005a, s. 110–111).
DLL	Dynamically Linked Library. Kirjasto, joka linkitetään ohjelman latauksen tai ajon aikana. (Eilam 2005a, s. 96.)
DRM	Digital Rights Management. Teknologia, jonka avulla julkaisijat ja tekijänoikeudenhaltijat voivat rajoittaa digitaalisen mediatiedoston tai laitteen käyttöä. (Eilam 2005a, s. 7.)
IBM PC	IBM Personal Computer. IBM:n alkuperäinen versio PC-tietokoneesta, jonka jälkeläisiä valtaosa nykyisistä mikrotietokoneista on. (Mueller 2006.)
Jäljitys	Trace. Listausta ohjelman suorittamista käskyistä (Peikari & Chuvakin 2004).
Keskeytyskohta	Breakpoint. Kohta, johon ohjelman suoritus pysähtyy ajattaessa debuggerissa. (Eilam 2005a, s. 15.)
Live-analyysi	Koodin ajaminen ja käyttäytymisen seuraaminen debuggerissa (Eilam 2005a, s. 110).
NTDLL.DLL	Tiedosto, joka toteuttaa osan Windows NT -perheen sisäisesti käyttämästä Native API -rajapinnasta (Eilam 2005a, s. 90–91).
PEB	Process Environment Block. Windows-käyttöjärjestelmien tietorakenne, joka sisältää prosessiin liittyviä parametreja. (Eilam 2005a, s. 381.)
Staattinen analyysi	Disassemblerin tuottaman symbolisen konekielilistauksen analysointi (Kruegel et al. 2004).
TLS	Thread Local Storage. Ohjelmointimenetelmä, jonka avulla eri säikeet voivat viitata samaan staattiseen tai globaaliin muuttujaan, jotka kuitenkin sijaitsevat eri muistiosoitteissa. (Eilam 2005a, s. 546–547.)

- Vastakutsufunktio** Callback function. Ylemmän tason funktio, joka annetaan parametrina alemmalle tasolle. Mahdollistaa ylemmän tason rutiinin kutsumisen alemmalta tasolta. (MSDN Library 2008a.)
- Yksisuuntainen funktio** Funktio, jonka arvo on helppo laskea, mutta arvosta on huomattavasti vaikeampi päätellä, millä syötteellä se on saatu (Schneier 1996, s. 29).

1. JOHDANTO

Ohjelmistoliiketoiminnan merkittävimpiä haasteita on estää ohjelmiston laitton kopiointi ja suojata toteutusyksityiskohdat kilpailijoilta. Näihin haasteisiin on vastattava estämällä *takaisinmallinnusta*, jossa tarvittavat tiedot sovelluksen toiminnasta selvitetään ilman ohjelman dokumentaatiota ja lähdekoodia. Tässä työssä esitellään ja vertaillaan lyhyesti erityyppisiä takaisinmallinnuksen estotekniikoita. Työ antaa lukijalle tarkan käsityksen jokaisen estotekniikan perusideasta ja teknisestä toteutuksesta sekä hyvän lähtökohdan arvioida tekniikan soveltuvuutta oman ohjelmiston suojaustarpeisiin.

Työn rakenne on seuraava: luvussa kaksi perehdytään takaisinmallinnukseen ja sen estotekniikoihin sekä esitellään kriteerit, joilla estotekniikoita vertaillaan. Luvussa kolme käsitellään valitut takaisinmallinnuksen estotekniikat, annetaan jokaisesta käytännön esimerkki ja arvioidaan jokaista estotekniikkaa valittujen kriteerien perusteella. Luvussa neljä kootaan tulokset yhteen ja arvioidaan niiden merkitystä.

2. TAKAISINMALLINNUKSEN ESTO

Termin takaisinmallinnus (englanniksi reverse engineering) alkuperä on laitteiston analysoinnissa. Se on perinteisesti tarkoittanut prosessia, jossa laitteistoa tutkimalla pyritään luomaan siitä kopio. Prosessin suorittaja on joku muu kuin järjestelmän alkuperäinen kehittäjä — ilman alkuperäisten piirustusten tukea. Ohjelmistotekniikassa takaisinmallinnuksen lähestymistapa on samankaltainen, mutta tavoitteena on tarpeellisen suunnittelutason ymmärryksen saavuttaminen toiminnallisen kopion tuottamisen sijaan. Analysointi kohdistuu järjestelmän komponenttien ja niiden vuorovaikutuksen tunnistamiseen sekä korkeamman abstraktiotason esitysmuotojen luomiseen. (Chikofsky & Cross 1990.) Tässä työssä termillä takaisinmallinnus viitataan prosessiin, jossa ihminen ohjelmistotyökalujen avulla muuntaa suoritettavan tiedoston symboliseksi konekieleksi, josta hän selvittää tarvitsemansa tiedot sovelluksen toiminnasta ilman ohjelman dokumentaatiota ja lähdekoodia.

Takaisinmallinnuksen käyttökohteet ovat monipuolisia ja merkittäviä. Esimerkiksi IBM PC -yhteensopivien kloonikoneiden synty edellytti, että Phoenix Technologies Ltd. -yrityksen teknikot takaisinmallinsivat IBM PC:n BIOS:n (Mueller 2006; Lee 2006; Schwartz 2001). Yhteensopivuuden edistäminen onkin yksi takaisinmallinnuksen merkittävimpiä käyttökohteita. Muita ovat esimerkiksi haittaohjelmien analysointi, odottamattomien ohjelmistovikojen paikallistaminen, oman koodin laittoman hyödyntämisen havaitseminen muiden ohjelmistoista, avoimen lähdekoodin luvattoman käytön tunnistaminen ja muiden tuotteista oppiminen (Eilam 2005a, s. ix–x).

Kuten takaisinmallinnukselle, myös sen estämiselle on tarve. Esimerkiksi haittaohjelmat hyödyntävät erilaisia tekniikoita, jotka vaikeuttavat takaisinmallinnusta, jotta analysointi hidastuisi. Kaupallisissa sovelluksissa puolestaan saatetaan haluta suojata oma innovatiivinen toteutus kilpailijalta hankaloittamalla takaisinmallinnusta. Monen kopiosuojauksen tehokkuuden ja turvallisuuden edellytyksenä taas on, että toteutustekniikka pysyy mahdollisimman hyvin salassa takaisinmallinnusyryksistä huolimatta. Nykyaikana takaisinmallinnuksen estotekniikoita tarvitaan myös tehokkaiden DRM-suojauksien toteuttamiseksi.

Tässä työssä vertaillaan keskenään yleisiä takaisinmallinnuksen estotekniikoita, joita havainnollistetaan esimerkkiohjelmilla. Estotekniikka on alustariippumaton, jos sen soveltaminen ei riipu prosessoriarkkitehtuurista eikä käyttöjärjestelmästä, jolle sovellus luodaan. Takaisinmallintajan näkökulmasta on kriteereiksi valittu havaitsemisen vaativuus ja poistamisen helppous, koska estotekniikan vahvuus riippuu nimenomaan siitä, kuinka nopeasti takaisinmallintaja onnistuu löytämään ja poistamaan

valitun estotekniikan suoritettavasta tiedostosta. Takaisinmallinnuksen estotekniikan vaikutuksia suojattavan sovelluksen kehittämiseen ja ylläpitoon on otettu huomioon arvioimalla estotekniikan toteuttamisen vaativuutta ja vaikutusta lähdekoodin ylläpidettävyyteen. Vertailukriteereiksi tekniikoiden kesken on siis valittu alustariippumattomuus, havaitsemisen vaativuus, poistamisen vaikeus, koodin ylläpidettävyyys ja toteuttamisen vaativuus.

Kaikki työn esimerkit ovat PC-arkkitehtuurille. Esimerkkiohjelmina käytetään takaisinmallinnushaasteiksi luotuja, natiiveja Win32-sovelluksia (Eilam 2005b; Leoniak 2007a; Leoniak 2007b; Leoniak 2007c). Idea jokaisessa näistä sovelluksista on sama: käyttäjän on selvitettävä, minkä salasanan ohjelma hyväksyy. Sovelluksista ei ole julkisesti saatavilla dokumentaatiota eikä korkean tason kielellä esitettyä lähdekoodia. Analysoinnissa hyödynnettiin debuggereita OllyDbg (Yuschuk 2007) ja WinDbg (Debugging Tools for Windows 2008), joilla myös työssä symbolisella konekielellä esitetyt listaukset on tuotettu.

3. TEKNIKOITA

Tässä luvussa esitellään yleisiä takaisinmallinnuksen estotekniikoita, jotka on jaettu kolmeen osaan. Tekniikat, joissa analysointia vaikeuttavat toimenpiteet kohdistuvat ohjelman dataan, käsitellään alaluvussa 3.1. Menetelmiä, jotka tekevät ohjelman symbolisen konekielen analysoinnin ihmiselle vaikeaksi, esitellään alaluvussa 3.2. Ohjelman koodin analysoinnin estämisen teknisiä keinoja käsitellään luvussa 3.3. Esiteltävät tekniikat on pyritty valitsemaan niin, että ne olisivat mahdollisimman monipuolinen katsaus erilaisiin vaihtoehtoihin.

3.1. Datan suojaaminen

Yksi takaisinmallinnuksen estämisen motiiveista on ohjelman sisältämän arkaluontoisen datan suojaaminen. Kryptografia on erinomainen keino suojata dataa, mutta salauksen edellytys on, että avain on ohjelman käytettävissä. Mikäli avain on toimitettava suoritettavan tiedoston mukana, se tulisi pystyä piilottamaan mahdollisimman hyvin. Yksi keino tähän on jakaa data pienempiin osiin ja säilöä se esimerkiksi useaan eri vakioon. Samalla tavalla voidaan välttää salasanan säilyttämiseltä yhtenäisenä merkkijonona ohjelman data-alueella.

Datan pilkkominen osiin voidaan toteuttaa yksinkertaisesti ohjelmointikielestä ja alustasta riippumatta. Koodin ylläpidettävyyden säilyttämiseksi pilkkomisen toteutuksen tulisi olla riippumaton pilkottavan datan pituudesta. Muuten esimerkiksi merkkijonon korvaaminen eripituisella vaatisi muutoksia myös pilkkomisalgoritmiin. Lisäksi lähdekoodin puutteellinen kommentointi voi tehdä merkkijonon sisällön selvittämisen vaikeaksi myös koodin ylläpitäjälle. Takaisinmallintajalle tekniikan havaitseminen voi olla hyvinkin helppoa, jopa tarpeetonta. Takaisinmallintajan ei välttämättä tarvitse edes tietää datan alkuperää tai muodostustapaa, jos hän löytää koodista kohdan, jossa data on koostettu takaisin alkuperäiseen muotoonsa. Tekniikan käyttöä tehostaakin, jos pilkottua dataa ei missään vaiheessa ohjelman suoritusta käsitellä kokonaisuutena. Tällöin takaisinmallintaja joutuu selvittämään kaikki datan palaset yhden kokonaisuuden sijaan. Data pilkkomisen poistaminen sinällään on takaisinmallintajalle helppoa. Suurin haaste on löytää ne kohdat koodissa, joissa datan palasiin viitataan.

Mikäli ohjelmassa halutaan vertailla käyttäjän syötettä jotakin tiettyä arvoa vastaan, voidaan vertailu suorittaa laskemalla molemmista sama muunnos ja vertailemalle muunnosten lopputulosta. Funktiota kutsutaan yksisuuntaiseksi, jos sen arvo on helppo laskea, mutta arvosta on huomattavasti vaikeampi päätellä, millä syötteellä se on saatu (Schneier 1996, s. 29). Jos muunnos lasketaan tällaisella

yksisuuntaisella funktiolla, ei muunnoksesta voida palauttaa alkuperäistä versiota kuin kokeilemalla eri vaihtoehtoja. Alkuperäinen salaisuus on siis melko hyvin turvassa, mikäli se esiintyy ohjelmassa ainoastaan muunnetussa muodossaan. Tekniikan turvallisuus riippuukin pitkälti muunnosfunktion ominaisuuksista.

Kuten datan pilkkominen, myös muunnoksen laskeminen voidaan toteuttaa ohjelmointikielestä ja alustasta riippumatta. Koodin ylläpidettävyyttä tekniikka ei juuri heikennä: sen sijaan, että vertailtaisiin syötettä oikeaan, verrataankin syötteen muunnosta oikeasta syötteestä laskettuun muunnokseen. Tällaista ratkaisua käytetään muun muassa UNIX:n sisäänkirjautumisessa salasanan tarkistamiseen (Schneier 1996, s. 52–53). Muunnoksen havaitseminen symbolisesta konekielestä on takaisinmallintajalle melko helppoa. Hänen tarvitsee paikallistaa ainoastaan kohdat, joissa ohjelma käsittelee käyttäjän antamaa syötettä sekä kohta, jossa muunnoksen tulosta vertaillaan oikeaan muunnoksen lopputulokseen. Tekniikan kiertäminen onnistuuikin helpoimmillaan kääntämällä vertailun ehto toisinpäin: jos muunnokset ovat erilaiset, käyttäjän syöte on oikein. Vertailun sijaan muunnoksen lopputulosta kannattaisikin käyttää vakiona, joka ohjaa ohjelman suoritusta mahdollisimman monessa kohtaa. Tällöin takaisinmallintajalle ei riitä minkään yksittäisen tarkastuksen ohittaminen, vaan hän joutuu selvittämään syötteen, jonka muunnos tuottaa oikean lopputuloksen.

```

6900105D  PUSH OFFSET FSC_Level1.690031A0  ; ASCII "salasana"
69001062  PUSH OFFSET FSC_Level1.690020BC  ; ASCII "%s"
69001067  CALL DWORD PTR DS:[<&MSVCR71.scanf>]
6900106D  MOVSX ECX,BYTE PTR DS:[69003042]
69001074  MOVSX EDX,BYTE PTR DS:[69003036]
6900107B  MOVSX EAX,BYTE PTR DS:[6900302E]
69001082  MOV EDI,DWORD PTR DS:[<&MSVCR71.sprintf>]
69001088  PUSH ECX
69001089  MOVSX ECX,BYTE PTR DS:[6900302C]
69001090  PUSH EDX
69001091  MOVSX EDX,BYTE PTR DS:[6900302B]
69001098  PUSH EAX
69001099  MOVSX EAX,BYTE PTR DS:[69003024]
690010A0  PUSH ECX
690010A1  MOVSX ECX,BYTE PTR DS:[69003021]
690010A8  PUSH EDX
690010A9  MOVSX EDX,BYTE PTR DS:[69003020]
690010B0  PUSH EAX
690010B1  PUSH ECX
690010B2  PUSH EDX
690010B3  PUSH OFFSET FSC_Level1.69003158  ; ASCII "%c%c%c%c%c%c%c%c"
690010B8  PUSH OFFSET FSC_Level1.69003310  ; ASCII "Asm07REC"
690010BD  CALL EDI
690010BF  PUSH OFFSET FSC_Level1.69003310  ; ASCII "Asm07REC"
690010C4  PUSH OFFSET FSC_Level1.690031A0  ; ASCII "salasana"
690010C9  CALL DWORD PTR DS:[<&MSVCR71._stricmp>]

```

Ohjelma 1. *Esimerkki merkkijonon pilkkomisesta*

Seuraavaksi tarkastelemme, kuinka esitellyt tekniikat käytännössä ilmenevät symbolisen konekielen tasolla. Ohjelmassa 1 on esitetty osa esimerkiohjelmasta (Leoniak 2007a) symbolisella konekielellä. Käyttäjä on syöttänyt ohjelmalle salasanaaksi ”salasana”. Osoitteessa 0x69001067 luetaan näppäimistöltä käyttäjän antama merkkijono osoitteeseen 0x690031A0. Seuraavilla riveillä luetaan kahdeksan merkkiä yksitellen osoitteesta 0x69003020 alkavasta merkkijonosta ”**Assembly 2007 Reverse-Engineering Challenge - Level 1**”. Ohjelma poimii lihavoituina esitetyt merkit ja muodostaa niistä sprintf-funktion (MSDN Library 2008b) avulla osoitteessa 0x690010BD merkkijonon, jota verrataan osoitteessa 0x690010C9 käyttäjän syötteeseen.

Ohjelman 1 toteutus on ylläpidettävyydeltään heikko: sprintf-funktion parametrit riippuvat pilkotun merkkijonon pituudesta. Lisäksi esimerkiohjelmassa syötetyn ja oikean salasanan vertailuun käytetään _stricmp-funktiota (MSDN Library 2008c), jonka paikantaminen on takaisinmallintajalle helppoa. Sen sijaan, että takaisinmallintaja joutuisi etsimään kaikki salasanan palaset erikseen, oikea salasana löytyy vaivattomasti kokonaisuudessaan _stricmp-funktiolle annettavasta parametrissa osoitteessa 0x690010BF.

```
004011E9  MOV EAX,DWORD PTR DS:[EDX-4]
004011EC  XOR EAX,5528566D
004011F1  XOR AH,BH
004011F3  PUSH ESP
004011F4  MOV DWORD PTR SS:[ESP],EAX
...
00401227  MOV EAX,DWORD PTR DS:[EDX-0A]
0040122A  XCHG DWORD PTR SS:[ESP],EAX
...
0040123A  CMP EAX,DWORD PTR SS:[ESP]
0040123D  POP EAX
0040123E  POP EAX
0040123F  JNE 004012FC
```

Ohjelma 2. Esimerkki yksinkertaisesta muunnoksesta

Ohjelma 2 on esimerkki muunnoksen käytöstä datan suojaamiseen ja siihen on valikoitu kolme kohtaa esimerkiohjelmasta (Leoniak 2007b) symbolisella konekielellä esitettynä. Osoitteessa 0x4011E9 luetaan salasanaa edustavat ohjelman komentorivin neljä viimeistä merkkiä rekisteriin EAX, jolle suoritetaan seuraavassa käskyssä XOR-operaatio vakion 0x5528566D kanssa. Arvolle tehdään toinenkin XOR-operaatio BH-rekisterin arvon kanssa, joka kuitenkin on nolla, mikäli ohjelmaa ei suoriteta debuggerissa (tähän palataan luvun 3.3. esimerkeissä). Näin saatu arvo tallennetaan muistiin osoitteessa 0x4011F4 myöhempää käyttöä varten. Osoitteesta 0x401227 alkava koodi lukee tietyt 4 tavua komentoriviltä ja säilöö ne muistiin. Nämä tavut muodostavat merkkijonon ”.exe” (kaksoissanan 0x6578652E), mikäli ohjelmalle on annettu parametrina nelimerkkinen salasana. Näitä neljää tavua vertaillaan käyttäjän syötteestä laskettuun muunnokseen osoitteessa 0x40123A. Mikäli muunnoksen tuloksena on saatu

0x6578652E, salasana on oikein. XOR-operaation \oplus ominaisuuksien $x \oplus x = 0$ ja $y \oplus 0 = y$ nojalla voidaan siis päätellä, että *vakio* \oplus *salasana* = *muunnos* \Leftrightarrow *salasana* = *vakio* \oplus *muunnos*. Oikea salasana on siis kaksoissana 0x30503343, joka vastaa ASCII-merkkijonoa "C3P0". Mikäli käytetty muunnos olisi XOR-operaation sijaan ollut yksisuuntainen, oikean salasanan löytämiseksi olisi pitänyt laskea muunnos jokaiselle nelitavuiselle salasanalle ja tutkia, mikä niistä tuottaa oikean lopputuloksen.

Esimerkkiohjelman (Leoniak 2007b) tapauksessa takaisinmallintaja haluaa selvittää myös sähköpostiosoitteen, jonka ohjelma tulostaa, kun sille syöttää oikean salasanan. Osoitteessa 0x40123A tehtävän vertailun lisäksi esimerkkiohjelma (Leoniak 2007b) käyttääkin muunnoksen lopputulosta myös osana tulostettavaa sähköpostiosoitetta. Jos takaisinmallintaja siis ainoastaan kääntää ehdon osoitteessa 0x40123F toisinpäin, ohjelma kyllä onnittelee käyttäjää oikean salasanan syöttämisestä, mutta sähköpostiosoite on väärä. Tämä on erinomainen esimerkki muunnoksen lopputuloksen käyttämisestä ohjelman suoritusta ohjaavana vakiona.

3.2. Inhimilliset tekijät

Vaikka takaisinmallinnuksen helpottamiseksi on olemassa joitakin työkaluja, merkittävin rooli on kuitenkin aina takaisinmallinnusta suorittavalla ihmisellä. Sopivin keinoin on mahdollista saada yksinkertaisesta koodista niin monimutkaista, että ihmisen on hyvin vaikea hahmottaa sitä. Eräs tekniikka on lisätä koodiin tiheästi toistuvia hyppykäskejä. Tällöin *live-analyysi*, koodin ajaminen ja analysoiminen debuggerissa (Eilam 2005a, s. 110), on huomattavasti hankalampaa, koska koodista näkee näytöllä vain muutaman rivin kerrallaan. Myös *staattinen analyysi*, disassemblerin tuottaman symbolisen konekieliesityksen analysointi (Kruegel et al. 2004), on melko turhauttavaa, jos muutaman käskyn välein pitää siirtyä lukemaan eri kohdasta listausta.

Toinen tekniikka on arvojen siirtely muuttujasta toiseen. Tämä tekee koodin analysoinnista hitaampaa, koska takaisinmallintaja ei pysty päättämään muistipaikalle mitään tiettyä merkitystä tai yksiselitteistä käyttötarkoitusta. Sen sijaan hän joutuu seuraamaan tarkasti muistipaikkojen arvoja ollakseen selvillä ohjelman tilasta. Koska takaisinmallintajan mielenkiinto kohdistuu usein ohjelmalle annettuun syötteeseen ja sille suoritettaviin operaatioihin, takaisinmallintajan aikaa voi tuhata siirtelemällä syötteen dataa merkityksettömästi muistipaikasta toiseen. Parhaimmillaan takaisinmallintaja turhautuu niin pahasti, että lakkaa edes yrittämästä.

Kolmas tekniikka on yksinkertaisesti lisätä ohjelmaan ylimääräisiä, epäolennaisia käskejä. Takaisinmallintajan on melko vaikea päätellä, onko jokin koodi tärkeää vai merkityksetöntä analysoimatta sitä tarkemmin. Näin takaisinmallintaja saadaan käyttämään aikaa epäolennaiseen. Kun takaisinmallintaja lisäksi tajuaa, että suuri osa koodista on lisätty ainoastaan hänen tehtävänsä hankaloittamiseksi, takaisinmallintajan keskittyminen voi hyvinkin herpaantua, jolloin merkittävät käskyt saattavat jäädä huomaamatta turhien seasta.

Kaikki edellä mainitut estotekniikat ovat alustariippumattomia ja melko vaikeita poistaa. Toisaalta varsinaiselle poistamiselle ei välttämättä ole tarvetta sen jälkeen, kun takaisinmallintaja on havainnut, että koodi on ylimääräistä eikä se tee mitään mielenkiintoista. Suoraan lähdekoodiin toteutettuina estotekniikat heikentävät merkittävästi ohjelman ylläpidettävyyttä, koska selkeän ja ymmärrettävän esitysmuodon sijaan joudutaan ylläpitämään tietoisesti monimutkaiseksi kirjoitettua koodia. Toteuttamisen vaikeus on keskitasoa, kunhan kääntäjä ei optimoi tarkoituksella lisättyä, toiminnan kannalta epäolennaista koodia pois. Ylimääräisen koodin lisäämistä on vaikeahko havaita, johtuen vaikeudesta erottaa olennainen symbolinen konekoodi epäolennaisesta. Toisaalta takaisinmallintajan ei välttämättä ole helppo havaita, milloin hänen keskittymistä koetetaan tietoisesti häiritä ja milloin kyse on vain tavallisesta vaikeasti seurattavasta koodista. Muistipaikkojen sisällön vaihtelevuus on helpompi tunnistaa, koska takaisinmallintaja pystyy melko nopeasti erottamaan, säilyykö muistipaikan sisällön merkitys ja käyttötarkoitus samana vai muuttuuko se. Estotekniikoista helpoiten tunnistettava on kuitenkin tiheästi toistuvien hyppykäskyjen käyttö, koska kääntäjät eivät generoi turhia toistuvia hyppykäskyjä tyhjästä.

Esimerkkihjelmassa (Leoniak 2007c) takaisinmallintajaa häiritään kaikin edellä mainituin keinoin. Liitteessä 1 on esitetty symbolisella konekielellä jäljitys (englanniksi trace) yhdestä kierroksesta silmukassa, jonka tarkoitus on laskea merkkijonon pituus etsimällä merkkijonon päättävä nollatavu. Normaalisti tähän riittäisi muutama konekäsky: liitteen 1 listauksessa käskyjä on yli 250. Erityisen tiheästi toistuvat hyppykäskyt: peräkkäisten hyppyjen väliin jää korkeintaan kolme muuta käskyä. Jo nopean silmäyksen perusteella voi huomata koodin vaihtelevan arvoja osoitteissa EBX+8, EBX+64 ja EBX+78 (EBX säilyy vakiona). Liite 1 on erinomainen osoitus siitä, miten takaisinmallintajalta vaaditaan erittäin hyviä hermoja, ettei hän anna periksi huomattessaan, että hänen pitkään käsky kerrallaan analysoimansa koodi suorittaakin vain triviaalin operaation kuten merkkijonon pituuden laskemisen.

3.3. Suoritettavan tiedoston tekninen suojaaminen

Takaisinmallintamisen edellytyksenä on tavallisesti ohjelman esittäminen symbolisella konekielellä. Usein on myös toivottavaa, että ohjelmalle voi staattisen analyysin lisäksi suorittaa live-analyysin. Tässä luvussa esiteltävät keinot pyrkivät vaikeuttamaan suoritettavan tiedoston muokkaamista ja analysointia debuggerissa, jotka molemmat ovat takaisinmallinnuksen yhteydessä usein käytettyjä keinoja.

Ensimmäinen esiteltävä tekniikka on suoritettavan tiedoston pakkaaminen. Takaisinmallinnuksen näkökulmasta se vaikeuttaa alkuperäisen ohjelman koodin paikantamista, koska ohjelman suoritus alkaa varsinaisen ohjelman sijaan purkajan koodilla. Merkittävämpi vaikutus pakkauksella on kuitenkin tiedoston muokkaamisen kannalta. Takaisinmallintajan on vaikea tehdä suoritettavaan tiedostoon muutoksia (esimerkiksi muuttaa jonkin vertailun ehto), koska hänen tulisi pystyä pakkaamaan muutettu versio. Suoritettavaa tiedostoa ei myöskään pysty sellaisenaan analysoimaan

staatisesti ilman purkamista, koska pakatusta versiosta voi tutkia ainoastaan purkajan koodia.

Kryptaamisella on pitkälti samat edut suoritettavan tiedoston pakkaamisen kanssa: tiedoston muuntelun ja staattisen analyysin estäminen. Kryptaamisella takaisinmallinnusta voidaan kuitenkin estää vielä pakkaamistakin tehokkaammin. Pakkauksen purkamiseksi riittää tuntea käytetty ohjelma tai algoritmi, mutta kryptaamisen purkaminen edellyttää myös oikean salausavaimen tuntemusta. Mikäli tämä avain ei ole suoritettavan tiedoston sisällä, se on kätetty huolellisesti tai se lasketaan ajon aikana esimerkiksi käyttäjän syöteen perusteella, on kryptauksen purkaminen hyvin haastavaa.

Kolmas merkittävä tekniikka on debuggauksen estäminen. Takaisinmallintaja saa ohjelman toiminnasta yleensä huomattavasti paremman käsityksen debuggerissa ajamalla kuin staattiseen analyysiin avulla. Debuggerin tunnistamiseen on lukuisia tapoja, muun muassa tähän tarkoitukseen suunniteltujen käyttöjärjestelmäkutsujen käyttö ja tarkastussummien laskeminen koodista. Jälkimmäinen perustuu siihen, että keskeytyskohdan (englanniksi breakpoint) asettamisella alueelle on sama vaikutus kuin koodin muokkaamisella, sillä yleensä debugger korvaa keskeytyskohdassa olevan käskyn keskeytyskohtakeskeytyksellä (englanniksi breakpoint interrupt), joka on PC:llä käsky INT 3 (Eilam 2005a, s. 331).

Koska suoritettavan tiedoston pakkaaminen ja debuggauksen estäminen ovat hyvin teknisiä keinoja, ne ovat myös riippuvaisia alustasta ja suoritettavan tiedoston formaatista. Jotkin valmiit pakkaustyökalut kuten UPX (Oberhumer et al. 2007) ovat kuitenkin saatavilla usealle eri alustalle. Toisaalta kryptauksen pystyy toteuttamaan samalla koodilla alustasta riippumatta. Tekniikat eivät välttämättä heikennä koodin ylläpidettävyyttä merkittävästi. Tiedoston pakkaaminen voidaan toteuttaa erillisellä työkalulla kääntämisen jälkeen ja kryptauskoodin voi pitää erillään varsinaisesta koodista kuten debuggerin tunnistamisenkin. Tekniikoiden toteuttamisen vaikeus vaihtelee suuresti. Suoritettavan tiedoston pakkaaminen valmiilla työkalulla on hyvin suoraviivaista. Kryptauksen toteuttaminen puolestaan edellyttää joko koodialueen muokkaamista tai data-alueella olevan koodin suorittamista, joten sen toteuttaminen on teknisesti haastavaa. Debuggerin tunnistaminen puolestaan edellyttää tietämystä debuggereiden toimintaperiaatteista tai käytetystä alustasta, jotka saattavat olla teknisesti monimutkaisimmat asiat koko ohjelman toteutuksessa.

Takaisinmallintajan näkökulmasta yleisellä työkalulla tehty suoritettavan tiedoston pakkaaminen on helppo havaita ja purkaa, koska tähän tarkoitukseen on luotu omat työkalut. Debuggauksen esto on erittäin helppo havaita, mikäli ohjelman havaitsee käyttäytyvän normaalista poiketen debuggerissa ajettaessa. Toisaalta mikäli ohjelman käytös muuttuu huomaamattomasti tai vain vähän, voi debuggauksen esto olla hyvinkin vaikea havaita ja paikantaa. Tekniikan poistamiseksi saattaa riittää yhden konekäskyn muuttaminen, mutta esimerkiksi keskeytyskohtien seurauksena muuttuvan tarkastussumman kiertäminen tai poistaminen on huomattavasti hankalampaa. Kryptauksen puolestaan pystyy yleensä tunnistamaan satunnaisen näköisestä datasta,

josta ohjelman suorittaman muokkauksen tuloksena syntyy suoritettavaa koodia (kyseessä voi toki olla muukin muunnos kuin kryptaus). Kryptauksen varsinainen poistaminen voi olla hyvin hankalaa, mutta toisaalta prosessori ei voi suorittaa kryptattua koodia, joten ohjelman on joka tapauksessa purettava kryptaus ennen koodin suoritusta.

Seuraavaksi tarkastelemme esimerkkiohjelmaa (Leoniak 2007b), joka on pakattu suoritettava tiedosto. Kun se annetaan syötteenä Microsoft:n dumpbin-ohjelmalle (Dumpbin 2008), joka listaa ohjelman sektiot, saadaan seuraavanlainen tuloste:

```
Dump of file fsc_level2.exe
```

```
File Type: EXECUTABLE IMAGE
```

```
Summary
```

```
5000 UPX0
1000 UPX1
1000 UPX2
```

Normaalisti suoritettavissa tiedostoissa on ainakin .text- ja .data-sektiot. Esimerkkiohjelman (Leoniak 2007b) tapauksessa dumpbin ilmoittaa kuitenkin vain kolme UPX-alkuista sektiota. UPX on ohjelma suoritettavien tiedostojen pakkaamisen (Oberhumer et al. 2007). Sen suorittaminen valitsimella -t testaa, onko suoritettava tiedosto pakattu UPX:llä. Tällä tavoin voi varmistua, että esimerkkiohjelma (Leoniak 2007b) on UPX-pakattu. Pakkauksen poistaminen onnistuu yksinkertaisesti valitsimella -d. Vaikka esimerkkiohjelman (Leoniak 2007b) UPX-pakkauksen purkaminen onnistuu yksinkertaisella komennolla, ei purettu ohjelmatiedosto toimi identtisesti pakatun kanssa. Syy tähän löytyy ohjelman alustuksesta, jonka jäljestä on esitetty osa ohjelmassa 3.

```
7C9011A4 call    dword ptr [ebp+8]
004070A4 cmp     dword ptr [esp+8],1
004070A9 jne     image00400000+0x70c9 (004070c9)
004070AB push   eax
004070AC mov     eax,dword ptr fs:[00000018h]
004070B2 mov     eax,dword ptr [eax+30h]
004070B5 movzx  eax,word ptr [eax+2]
004070B9 cmp     eax,0
004070BC sete  al
004070BF imul  eax,eax,8
004070C2 sub     byte ptr [image00400000+0x63bd (004063bd)],al
004070C8 pop     eax
004070C9 ret
```

Ohjelma 3. Jäljitys esimerkkiohjelman (Leoniak 2007b) TLS-alustuksesta

Osoitteessa 0x7C09011A4 on NTDLL.DLL:n funktio LdrpCallInitRoutine, jota on kutsuttu saman DLL:n funktiosta LdrpCallTlsInitializers. Näiden funktioiden tehtäviin kuuluu kutsua tarvittaessa moduulille TLS:n alustuksen suorittavaa vastakutsufunktiota (Pietrek 1999). TLS (Thread Local Storage) on ohjelmointimenetelmä, jonka avulla eri säikeet voivat viitata samaan staattiseen tai globaaliin muuttujaan, jotka kuitenkin sijaitsevat eri muistiosoitteissa (Eilam 2005a, s. 546–547). Ohjelmassa 3 TLS:n alustuksen suorittava vastakutsufunktio (englanniksi callback function) alkaa osoitteesta 004070A4. Funktion tehtävä on tarkastaa, onko ohjelmaan kiinnitetty debuggeri. Tämä tapahtuu tutkimalla PEB-tietueen BeingDebugged-elementtiä, jonka arvo luetaan EAX-rekisteriin osoitteessa 0x4070B5. Mikäli prosessia ei debugata, EAX:n arvo on nolla, jolloin osoitteessa 0x4070BC rekisteri AL saa arvon 1, joka kerrotaan seuraavassa käskyssä kahdeksalla. Osoitteessa 4070C2 AL:n arvo vähennetään tavusta osoitteessa 0x4063BD, joka on itse asiassa osa konekäskyä. Jos prosessia debugataan, EAX:n arvo on nolla, joten käsky jää muotoon

```
004063BC  E9 99AFFFFF  JMP fsc_orig.0040135A.
```

Jos prosessia ei debugata, EAX:n arvo on kahdeksan, joten käsky muuttuu muotoon

```
004063BC  E9 91AFFFFF  JMP fsc_orig.00401352.
```

Osoitteessa 0x401352 on seuraava yksinkertainen EBX:n nollaava koodi:

```
00401352  33DB          XOR EBX,EBX
00401354  68 5A134000  PUSH fsc_orig.0040135A
00401359  C3           RETN
```

Seuraavan kerran rekisteriä EBX käytetään ohjelman 2 osoitteessa 0x4011F1, jossa käyttäjän syöttämälle salasanalle tehdään XOR-operaatio EBX:n kanssa. Mikäli ohjelmaa debugataan, EBX ei ole nolla, joten esimerkkiohjelma (Leoniak 2007b) ei hyväksy käyttäjän syöttämää salasanaa, vaikka se olisikin oikein. Koska ohjelman 3 vastakutsufunktio on osa UPX-pakatun version alustusta eikä alkuperäistä ohjelmaa, purettu versio ei sisällä ollenkaan kyseistä vastakutsufunktiota. Niinpä se käyttäytyy aina niin kuin olisi debugattavana. Vaikka esimerkkiohjelman (Leoniak 2007b) tapauksessa ohjelmatiedoston pakkaus onkin erittäin helppo havaita ja poistaa, takaisinmallintajan on helpompi havaita antidebugger-tekniikka pakatusta versiosta. Esimerkkiohjelman (Leoniak 2007b) antidebugger-tekniikka on hyvin hienovarainen, sillä debuggerin tunnistus tehdään ohjelman alustuksessa niin varhaisessa vaiheessa, että esimerkiksi OllyDbg:llä (Yuschuk 2007) kyseistä koodia ei pysty debugaamaan. Toisaalta tekniikan poistamiseksi riittää muuttaa puretusta versiosta osoitteessa 0x4063BC olevan hyppekäskyn kohdeosoitteeksi 0x401352.

Liitteistä 2 ja 3 löytyy esimerkki kryptauksen käytöstä takaisinmallinnuksen estämiseen. Liitteessä 2 on esitetty esimerkkiohjelmassa (Eilam 2005b) käytetty salakirjoitusrutiini ja liitteessä 3 purkurutiini. Liitteessä 3 osoitteessa 0x401785 luetaan

osoitteessa 0x406008 olevan muuttujan arvo rekisteriin EAX, jonka sisältö kopioidaan seuraavassa käskyssä osoitteeseen EBP-9C0:

```
00401785    MOV EAX,DWORD PTR DS:[406008]
0040178A    MOV DWORD PTR SS:[EBP-9C0],EAX
```

Osoitteessa 0x4017DB muistipaikan EBP-9C0 arvoa käytetään XOR-operaation toisena parametrina:

```
004017DB    XOR EAX,DWORD PTR SS:[EBP-9C0]
```

Muistipaikan 0x406008 sisältö toimii siis purkuavaimena. Mielenkiintoista on arvon alkuperä: liitteessä 2 osoitteessa 0x4034D0 kopioidaan rekisterin EAX arvo muistipaikkaan 0x406008. Kyseessä on kryptatun koodin tarkastussumma (Eilam 2005a, s. 402–403). Mikäli kryptattavasta koodista on muutettu jotain, tarkastussumma ei ole oikein eikä myöhemmin suoritettava liitteessä 3 esitetty salauksen purkaminen onnistu.

4. TULOSTEN ARVIOINTI

Taulukon 1 riveillä on esitetty vertailtavat takaisinmallinnuksen estotekniikat ja sarakkeissa arviointikriteerit, joiden täyttymistä on arvioitu luvun 3 tulosten pohjalta kvalitatiivisella asteikolla ”-”, ”±” ja ”+”. Estotekniikoiden järjestys ja luokittelu taulukossa noudattaa luvun 3 rakennetta. Kriteerit on jaettu kaksoisviivalla toteutusta koskeviin ja takaisinmallinnukseen liittyviin.

Taulukko 1. Estotekniikoiden vertailu

	Alusta- riippumattomuus	Koodin ylläpidettävyys	Toteuttamisen helppous	Havaitsemisen vaativuus	Poistamisen vaikeus
Merkkijonojen pilkkominen	+	-	+	-	±
Muunnoksen laskeminen datasta	+	+	+	-	-
Tiheästi toistuvat ylimääräiset hyppykäskyt	+	-	±	-	+
Muistipaikkojen sisällön vaihtelevuus keskenään	+	-	±	±	+
Ylimääräisen koodin lisääminen	+	-	±	+	+
Suoritettavan tiedoston pakkaaminen	±	+	+	-	-
Kryptaus	+	+	-	±	+
Debuggauksen estäminen	-	+	±	±	±

Taulukosta 1 huomataan, että suurin osa menetelmistä on alustariippumattomia. Ainoastaan debuggauksen estäminen saa alustariippumattomuudesta arvosanan ”-”. Syynä on tekniikan vahva riippuvuus sekä käyttöjärjestelmästä että prosessoriarkkitehtuurista. Suoritettavan tiedoston pakkaaminen saa arvosanan ”±”,

koska tekniikka on riippuvainen alustan käyttämästä suoritettavan tiedoston formaatista, mutta työkaluja pakkaamiseen on kuitenkin saatavilla lukuisille eri alustoille.

Koodin ylläpidettävyyden osalta tekniikat jakautuvat kahteen ryhmään: osa ei vaikuta ylläpidettävyyteen juuri ollenkaan (arvosana ”+”), osa taas heikentää sitä merkittävästi (arvosana ”-”). Erityisesti kaikki takaisinmallintajan inhimillisiin ominaisuuksiin keskittyvät tekniikat saivat arvosanan ”-”, kun taas kaikki suoritettavan tiedoston tekniset suojaamiskeinot saivat arvosanan ”+”. Tämä selittyy oletuksella siitä, että edelliset toteutetaan suoraan varsinaisen lähdekoodin sekaan, kun taas jälkimmäiset voidaan pitää erillään muusta koodista tai toteuttaa vasta käännöksen jälkeen. Mikäli tekniikka toteutetaan niin, että nämä oletukset eivät ole voimassa, myös vaikutus koodin ylläpidettävyyteen on erilainen.

Toteuttamisen helppous saa arvosanan ”+” molemmissa datan suojaamiseksi esitellyissä tekniikoissa. Kaikki takaisinmallintajaan kohdistuvat keinot puolestaan saavat keskimmäisen arvosanan ”±”, koska ovat datan suojaamistekniikoita vaikeampia toteuttaa, mutta kuitenkin vaatavuudeltaan keskitasoa. Suoritettavan tiedoston suojaamisen teknisissä keinoissa toteuttamisen helppous vaihtelee laidasta laitaan. Suoritettavan tiedoston pakkaaminen on erittäin helppoa valmiilla työkaluilla, debuggauksen estäminen vaatii perehtymistä käytettävään alustaan ja kryptaus on selvästi vaativin toteuttaa kaikista esitellyistä tekniikoista.

Takaisinmallintajan näkökulmasta datan suojaamistekniikat ovat helpoimmat havaita. Suoritettavan tiedoston teknisen suojaaminen keinot ovat havaitsemisen vaatavuuden kannalta keskitasoa. Laajimman kirjon havaitsemisen vaikeudessa tarjoavat takaisinmallintajaan kohdistuvat tekniikat. Tiheästi toistuvat hyppykäskyt huomaa välittömästi. Muistipaikkojen sisällön vaihtelemisen toteamiseen menee hieman pidempään, mutta ylimääräisen koodin olemassaolon takaisinmallintaja saattaa havaita vasta tuhlattuaan hyvin paljon aikaa sellaisen koodin analysointiin, joka ei teekään mitään ohjelman kannalta merkittävää.

Datan suojaamistekniikoiden poistaminen on melko helppoa. Kaikki takaisinmallintajaan vaikuttavat tekniikat ovat puolestaan vaikeita poistaa. Tämä johtuu osittain samoista syistä kuin koodin ylläpidettävyyden heikkeneminen: tekniikat ovat niin tiiviisti alkuperäiseen lähdekoodiin sidottuja, että poistaminen olisi hyvin vaativaa. Toisaalta näiden tekniikoiden osalta havaitseminen on usein poistamista tärkeämpää: kun takaisinmallintaja oivaltaa, että koodin takaisinmallinnusta on tarkoituksella vaikeutettu, hän voi lähestyä ohjelmaa keinoin, jotka kärsivät vähiten valitusta takaisinmallinnuksen estotekniikasta. Suoritettavaan tiedostoon kohdistuvien teknisten keinojen poistaminen puolestaan vaikuttaisi olevan sitä vaikeampaa, mitä vaativampi tekniikan toteutus on.

Jos kaikki kriteerit ovat yhtä tärkeitä, parhaimmiksi menetelmiksi osoittautuvat ylimääräisen koodin lisääminen ja kryptaus. Ylimääräisen koodin lisääminen saa arvosanan ”-” ainoastaan koodin ylläpidettävyyden merkittävästä heikkenemisestä, kryptaus puolestaan erittäin vaativasta toteutuksesta. Sopivimman menetelmän valinta riippuu kuitenkin käyttökohteesta. Kaikki kriteerit eivät ole olennaisia kaikissa

sovelluksissa. Alustariippuvuus ei haittaa, jos kohdejärjestelmiä on vain yksi. Jos sovellukselle ei julkaisun jälkeen aiota suorittaa minkäänlaista ylläpitoa, on kriteeri ylläpidettävyydestä turha. Mikäli estotekniikan toteuttajan tietämys riittää minkä tahansa tekniikan toteuttamiseksi, ei toteuttamisen vaativuudella ole niin suurta merkitystä — ammattitaitoinen ja motivoitunut kehittäjä saattaa jopa mieluummin valita estotekniikan, jonka toteuttaminen tarjoaa hänelle enemmän haastetta.

Mikäli ohjelmaan sovellettavia takaisinmallinnuksen estotekniikoita on enemmän kuin yksi, taulukko 1 ei kerro koko totuutta. Tässä työssä käytetyillä kriteereillä kaksi tehokkaimmaksi osoittautunutta tekniikkaa eivät välttämättä ole tehokkain pari. Yhdistämällä esimerkiksi kryptaus ja debuggauksen esto on mahdollista vaikeuttaa takaisinmallinnusta huomattavasti, koska tämä luo eräänlaisen muna-kana-ongelman: ohjelmaa ei voi analysoida staattisesti, koska koodi on kryptattu; live-analyysi puolestaan ei onnistu, koska debuggaus on estetty. Samantyyppinen asetelma voidaan luoda myös käyttämällä suoritettavan tiedoston pakkausta debuggauksen eston kanssa.

5. YHTEENVETO

Tässä työssä esiteltiin erilaisia takaisinmallinnuksen estotekniikoita sekä yleisellä tasolla että symbolisella konekielellä esitettyjen esimerkkien avulla. Jokainen estotekniikka arvioitiin alustariippumattomuuden, havaitsemisen vaativuuden, poistamisen vaikeuden, koodin ylläpidettävyyden ja toteuttamisen helppouden suhteen. Esitellyistä takaisinmallinnuksen estotekniikoista tehokkaimmiksi osoittautuivat ylimääräisen koodin lisääminen ja kryptaus, kun kaikkien kriteerien (alustariippumattomuus, havaitsemisen vaativuus, poistamisen vaikeus, koodin ylläpidettävyyden ja toteuttamisen helppous) painoarvot olivat yhtä suuret. Kunkin kriteerin tärkeys ja täten myös sopivin estotekniikka ovat kuitenkin sovelluskohtaisia.

Työn tulokset osoittivat, että tehokkaita estotekniikoiden yhdistelmiä ei voi päätellä sen perusteella, kuinka hyvin yksittäiset tekniikat täyttävät tähän työhön valitut kriteerit. Ongelman käsittelyä voisikin tulevaisuudessa jatkaa kehittämällä uusia vertailukriteereitä, jotka mahdollistaisivat myös tehokkaiden estotekniikoiden yhdistelmien päättelemisen. Kriteerien suunnittelussa tulisi ottaa tarkemmin huomioon erilaisten takaisinmallinnustekniikoiden näkökulma. Jokaisen estotekniikan osalta tulisi selvittää, minkä takaisinmallinnustekniikan käyttöä se vaikeuttaa ja millä tekniikoilla se on ohitettavissa. Näin pystyttäisiin tehokkaasti yhdistelemään sellaisia estotekniikoita, joista jokaisen ohittaminen edellyttäisi takaisinmallinnustekniikan käyttöä, jonka jokin toinen yhdistelmässä käytetty tekniikka puolestaan estää.

LÄHTEET

Chikofsky, E.J. & Cross, J.H. Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software 7(1990)1, s. 13–17.

Debugging Tools for Windows. 2008. WinDbg [tietokoneohjelma]. [viitattu 13.1.2008] Saatavissa: <http://www.microsoft.com/whdc/devtools/debugging/default.aspx>

Dumpbin. 2008. Description of the DUMPBIN utility. [viitattu 13.1.2008] Saatavissa: <http://support.microsoft.com/kb/177429>

Eilam, E. 2005a. Reversing: Secrets of Reverse Engineering. Indianapolis, Wiley. 617 s.

Eilam, E. 2005b. Defender [tietokoneohjelma]. [viitattu 9.12.2007] Saatavissa: http://crackmes.de/users/eldad_eilam/defender.exe/download

Kruegel, C., Robertson, W., Vauler, F. & Vigna G. 2003. Static Disassembly of Obfuscated Binaries. Proceedings of the 13th USENIX Security Symposium, San Diego, California, USA, August 9–13, 2004.

Lee, T. B. 2006. Circumventing Competition. [viitattu 5.1.2008] Saatavissa: <http://www.cato.org/pubs/pas/pa564.pdf>

Leoniak, K. 2007a. F-Secure's Reverse Engineering Challenge for ASSEMBLY Level 1 [tietokoneohjelma]. [viitattu 9.12.2007] Saatavissa: http://www.f-secure.com/export/sites/fs_global_site/khallenge/assembly_2007/FSC_Level1.zip

Leoniak, K. 2007b. F-Secure's Reverse Engineering Challenge for ASSEMBLY Level 2 [tietokoneohjelma]. [viitattu 9.12.2007] Saatavissa: http://www.f-secure.com/export/sites/fs_global_site/khallenge/assembly_2007/FSC_Level2.zip

Leoniak, K. 2007c. F-Secure's Reverse Engineering Challenge for ASSEMBLY Level 3 [tietokoneohjelma]. [viitattu 9.12.2007] Saatavissa: http://www.f-secure.com/export/sites/fs_global_site/khallenge/assembly_2007/FSC_Level3.zip

MSDN Library. 2008a. Implementing Callback Functions. [viitattu 21.1.2008] Saatavissa: [http://msdn2.microsoft.com/en-us/library/d186xcf0\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/d186xcf0(VS.71).aspx)

MSDN Library. 2008b. Run-Time Library Reference: sprintf, swprintf. [viitattu 13.1.2008] Saatavissa: [http://msdn2.microsoft.com/en-us/library/ybk95axf\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/ybk95axf(VS.71).aspx)

MSDN Library. 2008c. Run-Time Library Reference: `_stricmp`, `_wcsicmp`, `_mbsicmp`. [viitattu 13.1.2008] Saatavissa: [http://msdn2.microsoft.com/en-us/library/k59z8dwe\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/k59z8dwe(VS.71).aspx)

Mueller, S. 2006. Upgrading and Repairing PCs. 17th edition. United States, Que. 1608 s.

Oberhumer, M., Molnár, L., Reiser J. F. 2007. UPX [tietokoneohjelma]. [viitattu 6.1.2008] Saatavissa: <http://upx.sourceforge.net/>

Peikari, C. & Chuvakin, A. 2004. Security Warrior. Sebastopol, California, USA, O'Reilly. 531 s.

Pietrek, M. Under The Hood. Microsoft Systems Journal 14(1999)9. [viitattu 6.1.2008] Saatavissa: <http://www.microsoft.com/msj/0999/hood/hood0999.aspx>

Schneier, B. 1996. Applied Cryptography: Protocols, Algorithms and Source Code in C. 2nd edition. New York, John Wiley & Sons. 758 s.

Schwartz, M. 2001. Reverse-Engineering. [viitattu 5.1.2008] Saatavissa: <http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,65532,00.html>

Yuschuk, O. 2007. OllyDbg [tietokoneohjelma]. [viitattu 9.12.2007] Saatavissa: <http://ollydbg.de>

LIITE 1: ESIMERKKIOHJELMA (LEONIAK 2007C)

Alla on esitetty symbolisella konekielellä jäljitys (englanniksi trace) yhdestä kierroksesta silmukassa, jonka tarkoitus on laskea merkkijonon pituus etsimällä merkkijonon päättävä nollatavu. Listauksen koodin takaisinmallintamista on pyritty vaikeuttamaan lisäämällä ylimääräistä koodia, vaihtelemalla muistipaikkojen sisältöä keskenään ja lisäämällä ylimääräisiä hyppykäskyjä. Listauksen ymmärtämisen helpottamiseksi ainoastaan olennaiset käskyt on kommentoitu ja alleviivattu. Kommenteissa on käytetty seuraavia merkintöjä:

i_n Kierroksella n tutkittavan merkin indeksi merkkijonossa
 i_{n+1} Seuraavalla kierroksella ($n+1$) tutkittavan merkin indeksi merkkijonossa
 $\&i_n$ i_n :n osoite
 m_jono Merkkijono, jonka pituutta määritetään
 $m_jono[i_n]$ Merkkijonon alkio i_n
 $==$ Yhtäsuuruusvertailu

```

00411115 Main MOV DWORD PTR DS:[EBX+78],204
0041111F Main JMP FSC_Leve.00413B52
00413B52 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00413B58 Main ADD DWORD PTR DS:[EBX+78],EAX
00413B5E Main JMP FSC_Leve.00414811
00414811 Main MOV EAX,DWORD PTR DS:[EBX+78] EAX=00163174 ; &i_n
00414817 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000000 ; i_n
00414819 Main MOV DWORD PTR DS:[EBX+64],EAX
0041481F Main JMP FSC_Leve.0041394C
0041394C Main MOV DWORD PTR DS:[EBX+78],0
00413956 Main JMP FSC_Leve.004131D6
004131D6 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
004131DC Main ADD DWORD PTR DS:[EBX+78],EAX
004131E2 Main JMP FSC_Leve.00414382
00414382 Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000000
00414388 Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00162F70
0041438E Main MOV DWORD PTR DS:[ECX],EAX ; i_n
00414390 Main JMP FSC_Leve.004147A9
004147A9 Main MOV DWORD PTR DS:[EBX+78],0
004147B3 Main JMP FSC_Leve.0040F7D6
0040F7D6 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040F7DC Main ADD DWORD PTR DS:[EBX+78],EAX
0040F7E2 Main JMP FSC_Leve.004118AB
004118AB Main MOV EAX,DWORD PTR DS:[EBX+78]
004118B1 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000000 ; i_n
004118B3 Main MOV DWORD PTR DS:[EBX+64],EAX
004118B9 Main JMP FSC_Leve.00412336
00412336 Main MOV DWORD PTR DS:[EBX+78],-4
00412340 Main JMP FSC_Leve.00413A7B
00413A7B Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70

```

```

00413A81  Main  ADD  DWORD PTR DS:[EBX+78],EAX
00413A87  Main  JMP  FSC_Leve.00411164
00411164  Main  MOV  EAX,DWORD PTR DS:[EBX+78]  EAX=00162F6C
0041116A  Main  MOV  EAX,DWORD PTR DS:[EAX]      EAX=0012FB70 ; mjono
-----
0041116C  Main  MOV  DWORD PTR DS:[EBX+68],EAX
00411172  Main  JMP  FSC_Leve.00413626
00413626  Main  MOV  EAX,DWORD PTR DS:[EBX+68]
0041362C  Main  ADD  DWORD PTR DS:[EBX+64],EAX ; mjono[i_n]
-----
00413632  Main  JMP  FSC_Leve.00412842
00412842  Main  MOV  EAX,DWORD PTR DS:[EBX+64]
00412848  Main  MOV  AL,BYTE PTR DS:[EAX]        EAX=0012FB30
0041284A  Main  MOV  BYTE PTR DS:[EBX+64],AL ; mjono[i_n]
-----
00412850  Main  JMP  FSC_Leve.0040F504
0040F504  Main  MOV  DWORD PTR DS:[EBX+78],0
0040F50E  Main  JMP  FSC_Leve.00410A59
00410A59  Main  MOV  EAX,DWORD PTR DS:[EBX+8]   EAX=00162F70
00410A5F  Main  ADD  DWORD PTR DS:[EBX+78],EAX
00410A65  Main  JMP  FSC_Leve.0040F4D0
0040F4D0  Main  MOV  AL,BYTE PTR DS:[EBX+64]    EAX=00162F30
-----
0040F4D6  Main  MOV  ECX,DWORD PTR DS:[EBX+78]
0040F4DC  Main  MOV  BYTE PTR DS:[ECX],AL ; mjono[i_n]
-----
0040F4DE  Main  JMP  FSC_Leve.0040FD55
0040FD55  Main  MOV  DWORD PTR DS:[EBX+64],0
0040FD5F  Main  JMP  FSC_Leve.0040E612
0040E612  Main  MOV  DWORD PTR DS:[EBX+78],4
0040E61C  Main  JMP  FSC_Leve.00411BB9
00411BB9  Main  MOV  EAX,DWORD PTR DS:[EBX+8]   EAX=00162F70
00411BBF  Main  ADD  DWORD PTR DS:[EBX+78],EAX
00411BC5  Main  JMP  FSC_Leve.00414783
00414783  Main  MOV  EAX,DWORD PTR DS:[EBX+64]  EAX=00000000
00414789  Main  MOV  ECX,DWORD PTR DS:[EBX+78]  ECX=00162F74
0041478F  Main  MOV  DWORD PTR DS:[ECX],EAX
00414791  Main  JMP  FSC_Leve.0040EB11
0040EB11  Main  MOV  DWORD PTR DS:[EBX+78],0
0040EB1B  Main  JMP  FSC_Leve.00413DE3
00413DE3  Main  MOV  EAX,DWORD PTR DS:[EBX+8]   EAX=00162F70
-----
00413DE9  Main  ADD  DWORD PTR DS:[EBX+78],EAX
00413DEF  Main  JMP  FSC_Leve.00410AD9
00410AD9  Main  MOV  EAX,DWORD PTR DS:[EBX+78]
00410ADF  Main  MOV  AL,BYTE PTR DS:[EAX]        EAX=00162F30
-----
00410AE1  Main  MOV  BYTE PTR DS:[EBX+64],AL ; mjono[i_n]
-----
00410AE7  Main  JMP  FSC_Leve.004127C7
004127C7  Main  MOV  DWORD PTR DS:[EBX+78],4
004127D1  Main  JMP  FSC_Leve.00411FDE
00411FDE  Main  MOV  EAX,DWORD PTR DS:[EBX+8]   EAX=00162F70
00411FE4  Main  ADD  DWORD PTR DS:[EBX+78],EAX
00411FEA  Main  JMP  FSC_Leve.00412209
00412209  Main  MOV  EAX,DWORD PTR DS:[EBX+78]  EAX=00162F74
0041220F  Main  MOV  AL,BYTE PTR DS:[EAX]        EAX=00162F00
00412211  Main  MOV  BYTE PTR DS:[EBX+68],AL
00412217  Main  JMP  FSC_Leve.00413E3B
00413E3B  Main  XOR  EAX,EAX  EAX=00000000

```

```

00413E3D Main MOV CL,BYTE PTR DS:[EBX+68] ECX=00162F00
00413E43 Main CMP BYTE PTR DS:[EBX+64],CL ; mjono[i_n] == 0?
00413E49 Main JNZ SHORT FSC_Leve.00413E4C
00413E4C Main MOV BYTE PTR DS:[EBX+64],AL
00413E52 Main JMP FSC_Leve.0040ECD2
0040ECD2 Main AND BYTE PTR DS:[EBX+64],1
0040ECD9 Main XOR BYTE PTR DS:[EBX+64],1
0040ECE0 Main JMP FSC_Leve.0040EF95
0040EF95 Main MOV DWORD PTR DS:[EBX+78],0
0040EF9F Main JMP FSC_Leve.0041417B
0041417B Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00414181 Main ADD DWORD PTR DS:[EBX+78],EAX
00414187 Main JMP FSC_Leve.0040FCF9
0040FCF9 Main MOV AL,BYTE PTR DS:[EBX+64] EAX=00162F01
0040FCFF Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00162F70
0040FD05 Main MOV BYTE PTR DS:[ECX],AL
0040FD07 Main JMP FSC_Leve.00414717
00414717 Main MOV DWORD PTR DS:[EBX+78],0
00414721 Main JMP FSC_Leve.00414771
00414771 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00414777 Main ADD DWORD PTR DS:[EBX+78],EAX
0041477D Main JMP FSC_Leve.0040F049
0040F049 Main MOV EAX,DWORD PTR DS:[EBX+78]
0040F04F Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000001
0040F051 Main MOV DWORD PTR DS:[EBX+64],EAX
0040F057 Main JMP FSC_Leve.0040ECBB
0040ECBB Main CMP DWORD PTR DS:[EBX+64],0
0040ECC5 Main JE SHORT FSC_Leve.0040ECCC
0040ECC7 Main JMP FSC_Leve.00414711
00414711 Main JMP FSC_Leve.00412474
00412474 Main MOV DWORD PTR DS:[EBX+78],200
0041247E Main JMP FSC_Leve.0040E9B2
0040E9B2 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040E9B8 Main ADD DWORD PTR DS:[EBX+78],EAX
0040E9BE Main JMP FSC_Leve.00410BE2
00410BE2 Main MOV EAX,DWORD PTR DS:[EBX+78] EAX=00163170
00410BE8 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000000
00410BEA Main MOV DWORD PTR DS:[EBX+64],EAX
00410BF0 Main JMP FSC_Leve.004117C6
004117C6 Main MOV DWORD PTR DS:[EBX+78],0
004117D0 Main JMP FSC_Leve.0040FFB1
0040FFB1 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040FFB7 Main ADD DWORD PTR DS:[EBX+78],EAX
0040FFBD Main JMP FSC_Leve.0041395C
0041395C Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000000
00413962 Main MOV ECX,DWORD PTR DS:[EBX+78]
00413968 Main MOV DWORD PTR DS:[ECX],EAX
0041396A Main JMP FSC_Leve.00410F8F
00410F8F Main MOV DWORD PTR DS:[EBX+64],1
00410F99 Main JMP FSC_Leve.00410E71
00410E71 Main MOV DWORD PTR DS:[EBX+78],4

```

```
00410E7B Main JMP FSC_Leve.00411F30
00411F30 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00411F36 Main ADD DWORD PTR DS:[EBX+78],EAX
00411F3C Main JMP FSC_Leve.004111AC
004111AC Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000001
004111B2 Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00162F74
004111B8 Main MOV DWORD PTR DS:[ECX],EAX
004111BA Main JMP FSC_Leve.004138ED
004138ED Main MOV DWORD PTR DS:[EBX+78],0
004138F7 Main JMP FSC_Leve.0040ED5C
0040ED5C Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040ED62 Main ADD DWORD PTR DS:[EBX+78],EAX
0040ED68 Main JMP FSC_Leve.0041178C
0041178C Main MOV EAX,DWORD PTR DS:[EBX+78]
00411792 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000000
00411794 Main MOV DWORD PTR DS:[EBX+64],EAX
0041179A Main JMP FSC_Leve.0040F85C
0040F85C Main MOV DWORD PTR DS:[EBX+78],4
0040F866 Main JMP FSC_Leve.00413F08
00413F08 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00413F0E Main ADD DWORD PTR DS:[EBX+78],EAX
00413F14 Main JMP FSC_Leve.00413CDA
00413CDA Main MOV EAX,DWORD PTR DS:[EBX+78] EAX=00162F74
00413CE0 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000001
00413CE2 Main MOV DWORD PTR DS:[EBX+68],EAX
00413CE8 Main JMP FSC_Leve.0040F037
0040F037 Main MOV EAX,DWORD PTR DS:[EBX+68]
0040F03D Main ADD DWORD PTR DS:[EBX+64],EAX
0040F043 Main JMP FSC_Leve.0041166E
0041166E Main MOV DWORD PTR DS:[EBX+78],0
00411678 Main JMP FSC_Leve.00412624
00412624 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0041262A Main ADD DWORD PTR DS:[EBX+78],EAX
00412630 Main JMP FSC_Leve.00410A21
00410A21 Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000001
00410A27 Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00162F70
00410A2D Main MOV DWORD PTR DS:[ECX],EAX
00410A2F Main JMP FSC_Leve.00410DE5
00410DE5 Main MOV DWORD PTR DS:[EBX+78],0
00410DEF Main JMP FSC_Leve.00411D38
00411D38 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00411D3E Main ADD DWORD PTR DS:[EBX+78],EAX
00411D44 Main JMP FSC_Leve.00412F1A
00412F1A Main MOV EAX,DWORD PTR DS:[EBX+78]
00412F20 Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000001
00412F22 Main MOV DWORD PTR DS:[EBX+64],EAX
00412F28 Main JMP FSC_Leve.0040E5AE
0040E5AE Main MOV DWORD PTR DS:[EBX+78],200
0040E5B8 Main JMP FSC_Leve.00414A69
00414A69 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00414A6F Main ADD DWORD PTR DS:[EBX+78],EAX
```

```

00414A75   Main   JMP   FSC_Leve.00411411
00411411   Main   MOV   EAX,DWORD PTR DS:[EBX+64] EAX=00000001
00411417   Main   MOV   ECX,DWORD PTR DS:[EBX+78] ECX=00163170
0041141D   Main   MOV   DWORD PTR DS:[ECX],EAX
0041141F   Main   JMP   FSC_Leve.0041430E
0041430E   Main   JMP   FSC_Leve.0040EA0F
0040EA0F   Main   MOV   DWORD PTR DS:[EBX+78],204
0040EA19   Main   JMP   FSC_Leve.00413322
00413322   Main   MOV   EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00413328   Main   ADD   DWORD PTR DS:[EBX+78],EAX
0041332E   Main   JMP   FSC_Leve.004129A7
004129A7   Main   MOV   EAX,DWORD PTR DS:[EBX+78] EAX=00163174
004129AD   Main   MOV   EAX,DWORD PTR DS:[EAX] EAX=00000000 ; i_n
004129AF   Main   MOV   DWORD PTR DS:[EBX+64],EAX
004129B5   Main   JMP   FSC_Leve.0040F2CD
0040F2CD   Main   MOV   DWORD PTR DS:[EBX+78],0
0040F2D7   Main   JMP   FSC_Leve.0040FCD5
0040FCD5   Main   MOV   EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040FCDB   Main   ADD   DWORD PTR DS:[EBX+78],EAX
0040FCE1   Main   JMP   FSC_Leve.004115D4
004115D4   Main   MOV   EAX,DWORD PTR DS:[EBX+64] EAX=00000000 ; i_n
004115DA   Main   MOV   ECX,DWORD PTR DS:[EBX+78] ECX=00162F70
004115E0   Main   MOV   DWORD PTR DS:[ECX],EAX
004115E2   Main   JMP   FSC_Leve.0040F3BC
0040F3BC   Main   MOV   DWORD PTR DS:[EBX+64],1 ; vakio 1
0040F3C6   Main   JMP   FSC_Leve.0040F944
0040F944   Main   MOV   DWORD PTR DS:[EBX+78],4
0040F94E   Main   JMP   FSC_Leve.00410F23
00410F23   Main   MOV   EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00410F29   Main   ADD   DWORD PTR DS:[EBX+78],EAX
00410F2F   Main   JMP   FSC_Leve.0040E576
0040E576   Main   MOV   EAX,DWORD PTR DS:[EBX+64] EAX=00000001 ; vakio 1
0040E57C   Main   MOV   ECX,DWORD PTR DS:[EBX+78] ECX=00162F74
0040E582   Main   MOV   DWORD PTR DS:[ECX],EAX
0040E584   Main   JMP   FSC_Leve.00410E19
00410E19   Main   MOV   DWORD PTR DS:[EBX+78],0
00410E23   Main   JMP   FSC_Leve.004138A7
004138A7   Main   MOV   EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
004138AD   Main   ADD   DWORD PTR DS:[EBX+78],EAX
004138B3   Main   JMP   FSC_Leve.0040FB13
0040FB13   Main   MOV   EAX,DWORD PTR DS:[EBX+78]
0040FB19   Main   MOV   EAX,DWORD PTR DS:[EAX] EAX=00000000 ; i_n
0040FB1B   Main   MOV   DWORD PTR DS:[EBX+64],EAX
0040FB21   Main   JMP   FSC_Leve.00412791
00412791   Main   MOV   DWORD PTR DS:[EBX+78],4
0041279B   Main   JMP   FSC_Leve.00411ED0
00411ED0   Main   MOV   EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00411ED6   Main   ADD   DWORD PTR DS:[EBX+78],EAX
00411EDC   Main   JMP   FSC_Leve.0040ED7E
0040ED7E   Main   MOV   EAX,DWORD PTR DS:[EBX+78] EAX=00162F74
0040ED84   Main   MOV   EAX,DWORD PTR DS:[EAX] EAX=00000001 ; vakio 1

```

```

0040ED86 Main MOV DWORD PTR DS:[EBX+68],EAX
0040ED8C Main JMP FSC_Leve.0040FB27
0040FB27 Main MOV EAX,DWORD PTR DS:[EBX+68]
0040FB2D Main ADD DWORD PTR DS:[EBX+64],EAX ;  $i_n + 1 = i_{n+1}$ 
0040FB33 Main JMP FSC_Leve.00412CEA
00412CEA Main MOV DWORD PTR DS:[EBX+78],0
00412CF4 Main JMP FSC_Leve.004145C2
004145C2 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
004145C8 Main ADD DWORD PTR DS:[EBX+78],EAX
004145CE Main JMP FSC_Leve.004113ED
004113ED Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000001 ;  $i_{n+1}$ 
004113F3 Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00162F70
004113F9 Main MOV DWORD PTR DS:[ECX],EAX
004113FB Main JMP FSC_Leve.00410199
00410199 Main MOV DWORD PTR DS:[EBX+78],0
004101A3 Main JMP FSC_Leve.0041350E
0041350E Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
00413514 Main ADD DWORD PTR DS:[EBX+78],EAX
0041351A Main JMP FSC_Leve.00410AC5
00410AC5 Main MOV EAX,DWORD PTR DS:[EBX+78]
00410ACB Main MOV EAX,DWORD PTR DS:[EAX] EAX=00000001 ;  $i_{n+1}$ 
00410ACD Main MOV DWORD PTR DS:[EBX+64],EAX
00410AD3 Main JMP FSC_Leve.0040FD97
0040FD97 Main MOV DWORD PTR DS:[EBX+78],204
0040FDA1 Main JMP FSC_Leve.0040E2B4
0040E2B4 Main MOV EAX,DWORD PTR DS:[EBX+8] EAX=00162F70
0040E2BA Main ADD DWORD PTR DS:[EBX+78],EAX
0040E2C0 Main JMP FSC_Leve.0040EEC8
0040EEC8 Main MOV EAX,DWORD PTR DS:[EBX+64] EAX=00000001 ;  $i_{n+1}$ 
0040EECE Main MOV ECX,DWORD PTR DS:[EBX+78] ECX=00163174 ;  $\&i_n$ 
0040EED4 Main MOV DWORD PTR DS:[ECX],EAX ;  $i_n = i_{n+1}$ 
0040EED6 Main JMP FSC_Leve.0041230C
0041230C Main JMP FSC_Leve.00411115

```

LIITE 2: KRYPTAUS (EILAM 2005B)

Alla on esitetty symbolisella konekielellä esimerkkiohjelmasta (Eilam 2005b) osa, joka ensin tutkii, onko koodi kryptattu vai ei. Edellisessä tapauksessa koodi puretaan, jälkimmäisessä kryptataan. Kryptauksen päätteeksi ohjelmaa tallettaa kryptatusta koodista lasketun tarkastussumman, jota liitteessä 3 käytetään purkuavaimena.

```

00403401  CMP DWORD PTR SS:[EBP-8],0      ; onko koodi kryptaamatonta?
00403405  JE SHORT Defender.0040346D     ; jos on, kryptaa
00403407  MOV EAX,DWORD PTR SS:[EBP-18] ; purkusilmukan alku
0040340A  SUB EAX,DWORD PTR SS:[EBP-20]
0040340D  MOV DWORD PTR SS:[EBP-30],EAX
00403410  MOV EAX,DWORD PTR SS:[EBP-20]
00403413  MOV DWORD PTR SS:[EBP-34],EAX
00403416  AND DWORD PTR SS:[EBP-24],0
0040341A  AND DWORD PTR SS:[EBP-28],0
0040341E  CMP DWORD PTR SS:[EBP-30],3
00403422  JBE SHORT Defender.0040346B
00403424  MOV EAX,DWORD PTR SS:[EBP-34]
00403427  MOV EAX,DWORD PTR DS:[EAX]
00403429  MOV DWORD PTR SS:[EBP-2C],EAX
0040342C  MOV EAX,DWORD PTR SS:[EBP-34]
0040342F  MOV EAX,DWORD PTR DS:[EAX]
00403431  XOR EAX,2BCA6179              ; symmetrinen salausavain
00403436  MOV ECX,DWORD PTR SS:[EBP-34]
00403439  MOV DWORD PTR DS:[ECX],EAX
0040343B  MOV EAX,DWORD PTR SS:[EBP-34]
0040343E  MOV EAX,DWORD PTR DS:[EAX]
00403440  XOR EAX,DWORD PTR SS:[EBP-28]
00403443  MOV ECX,DWORD PTR SS:[EBP-34]
00403446  MOV DWORD PTR DS:[ECX],EAX
00403448  MOV EAX,DWORD PTR SS:[EBP-2C]
0040344B  MOV DWORD PTR SS:[EBP-28],EAX
0040344E  MOV EAX,DWORD PTR SS:[EBP-24]
00403451  XOR EAX,DWORD PTR SS:[EBP-2C]
00403454  MOV DWORD PTR SS:[EBP-24],EAX
00403457  MOV EAX,DWORD PTR SS:[EBP-34]
0040345A  ADD EAX,4
0040345D  MOV DWORD PTR SS:[EBP-34],EAX
00403460  MOV EAX,DWORD PTR SS:[EBP-30]
00403463  SUB EAX,4
00403466  MOV DWORD PTR SS:[EBP-30],EAX
00403469  JMP SHORT Defender.0040341E     ; purkusilmukan loppu
0040346B  JMP SHORT Defender.004034D5
0040346D  MOV EAX,DWORD PTR SS:[EBP-18] ; kryptaussilmukan alku
00403470  SUB EAX,DWORD PTR SS:[EBP-20]
00403473  MOV DWORD PTR SS:[EBP-40],EAX
00403476  MOV EAX,DWORD PTR SS:[EBP-20]
00403479  MOV DWORD PTR SS:[EBP-44],EAX
0040347C  AND DWORD PTR SS:[EBP-38],0

```

```

00403480 AND DWORD PTR SS:[EBP-3C],0
00403484 CMP DWORD PTR SS:[EBP-40],3
00403488 JBE SHORT Defender.004034CD
0040348A MOV EAX,DWORD PTR SS:[EBP-44]
0040348D MOV EAX,DWORD PTR DS:[EAX]
0040348F XOR EAX,DWORD PTR SS:[EBP-3C]
00403492 MOV ECX,DWORD PTR SS:[EBP-44]
00403495 MOV DWORD PTR DS:[ECX],EAX
00403497 MOV EAX,DWORD PTR SS:[EBP-44]
0040349A MOV EAX,DWORD PTR DS:[EAX]
0040349C XOR EAX,2BCA6179 ; symmetrinen salausavain
004034A1 MOV ECX,DWORD PTR SS:[EBP-44]
004034A4 MOV DWORD PTR DS:[ECX],EAX
004034A6 MOV EAX,DWORD PTR SS:[EBP-44]
004034A9 MOV EAX,DWORD PTR DS:[EAX]
004034AB MOV DWORD PTR SS:[EBP-3C],EAX
004034AE MOV EAX,DWORD PTR SS:[EBP-44]
004034B1 MOV ECX,DWORD PTR SS:[EBP-38]
004034B4 XOR ECX,DWORD PTR DS:[EAX]
004034B6 MOV DWORD PTR SS:[EBP-38],ECX
004034B9 MOV EAX,DWORD PTR SS:[EBP-44]
004034BC ADD EAX,4
004034BF MOV DWORD PTR SS:[EBP-44],EAX
004034C2 MOV EAX,DWORD PTR SS:[EBP-40]
004034C5 SUB EAX,4
004034C8 MOV DWORD PTR SS:[EBP-40],EAX
004034CB JMP SHORT Defender.00403484 ; kryptaussilmukan loppu
004034CD MOV EAX,DWORD PTR SS:[EBP-38] ; laitetaan kryptauksen
004034D0 MOV DWORD PTR DS:[406008],EAX ; tarkastussumma talteen
004034D5 PUSH Defender.004041FD
004034DA POP EBX
004034DB JMP EBX

```

LIITE 3: SALAUKSEN PURKU (EILAM 2005B)

Alla on esitetty symbolisella konekielellä salauksen purkurutiini esimerkkiohjelmasta (Eilam 2005b). Listauksesta 2 ja tavallisuudesta poiketen purkuavaimena käytetään ajonaikana laskettua arvoa, joka tässä tapauksessa on listauksessa 2 esitetty kryptauksen tarkastussumma.

```

00401774  MOV DWORD PTR SS:[EBP-14],1
0040177B  CMP DWORD PTR SS:[EBP-14],0
0040177F  JE Defender.00401845          ; hypätään salauksen purkuun
00401785  MOV EAX,DWORD PTR DS:[406008] ; tarkistussumma
0040178A  MOV DWORD PTR SS:[EBP-9C0],EAX
00401790  MOV EAX,DWORD PTR SS:[EBP-98C]
00401796  SUB EAX,DWORD PTR SS:[EBP-9AC]
0040179C  MOV DWORD PTR SS:[EBP-9C4],EAX
004017A2  MOV EAX,DWORD PTR SS:[EBP-9AC]
004017A8  MOV DWORD PTR SS:[EBP-9C8],EAX
004017AE  AND DWORD PTR SS:[EBP-9B4],0
004017B5  AND DWORD PTR SS:[EBP-9B8],0
004017BC  CMP DWORD PTR SS:[EBP-9C4],3 ; purkusilmukan alku
004017C3  JBE SHORT Defender.00401840
004017C5  MOV EAX,DWORD PTR SS:[EBP-9C8]
004017CB  MOV EAX,DWORD PTR DS:[EAX]
004017CD  MOV DWORD PTR SS:[EBP-9BC],EAX
004017D3  MOV EAX,DWORD PTR SS:[EBP-9C8]
004017D9  MOV EAX,DWORD PTR DS:[EAX]
004017DB  XOR EAX,DWORD PTR SS:[EBP-9C0] ; tarkistussumma avaimena
004017E1  MOV ECX,DWORD PTR SS:[EBP-9C8]
004017E7  MOV DWORD PTR DS:[ECX],EAX
004017E9  MOV EAX,DWORD PTR SS:[EBP-9C8]
004017EF  MOV EAX,DWORD PTR DS:[EAX]
004017F1  XOR EAX,DWORD PTR SS:[EBP-9B8]
004017F7  MOV ECX,DWORD PTR SS:[EBP-9C8]
004017FD  MOV DWORD PTR DS:[ECX],EAX
004017FF  MOV EAX,DWORD PTR SS:[EBP-9BC]
00401805  MOV DWORD PTR SS:[EBP-9B8],EAX
0040180B  MOV EAX,DWORD PTR SS:[EBP-9B4]
00401811  XOR EAX,DWORD PTR SS:[EBP-9BC]
00401817  MOV DWORD PTR SS:[EBP-9B4],EAX
0040181D  MOV EAX,DWORD PTR SS:[EBP-9C8]
00401823  ADD EAX,4
00401826  MOV DWORD PTR SS:[EBP-9C8],EAX
0040182C  MOV EAX,DWORD PTR SS:[EBP-9C4]
00401832  SUB EAX,4
00401835  MOV DWORD PTR SS:[EBP-9C4],EAX
0040183B  JMP Defender.004017BC        ; purkusilmukan loppu
00401840  JMP Defender.004018FE        ; hypätään purettuun koodiin
00401845  MOV EAX,DWORD PTR DS:[406008]
0040184A  MOV DWORD PTR SS:[EBP-9D4],EAX
00401850  MOV EAX,DWORD PTR SS:[EBP-98C]
00401856  SUB EAX,DWORD PTR SS:[EBP-9AC]
0040185C  MOV DWORD PTR SS:[EBP-9D8],EAX
00401862  MOV EAX,DWORD PTR SS:[EBP-9AC]
00401868  MOV DWORD PTR SS:[EBP-9DC],EAX
0040186E  AND DWORD PTR SS:[EBP-9CC],0
00401875  AND DWORD PTR SS:[EBP-9D0],0
0040187C  CMP DWORD PTR SS:[EBP-9D8],3 ; kryptaussilmukan alku
00401883  JBE SHORT Defender.004018F3

```

```
00401885 MOV EAX,DWORD PTR SS:[EBP-9DC]
0040188B MOV EAX,DWORD PTR DS:[EAX]
0040188D XOR EAX,DWORD PTR SS:[EBP-9D0]
00401893 MOV ECX,DWORD PTR SS:[EBP-9DC]
00401899 MOV DWORD PTR DS:[ECX],EAX
0040189B MOV EAX,DWORD PTR SS:[EBP-9DC]
004018A1 MOV EAX,DWORD PTR DS:[EAX]
004018A3 XOR EAX,DWORD PTR SS:[EBP-9D4]
004018A9 MOV ECX,DWORD PTR SS:[EBP-9DC]
004018AF MOV DWORD PTR DS:[ECX],EAX
004018B1 MOV EAX,DWORD PTR SS:[EBP-9DC]
004018B7 MOV EAX,DWORD PTR DS:[EAX]
004018B9 MOV DWORD PTR SS:[EBP-9D0],EAX
004018BF MOV EAX,DWORD PTR SS:[EBP-9DC]
004018C5 MOV ECX,DWORD PTR SS:[EBP-9CC]
004018CB XOR ECX,DWORD PTR DS:[EAX]
004018CD MOV DWORD PTR SS:[EBP-9CC],ECX
004018D3 MOV EAX,DWORD PTR SS:[EBP-9DC]
004018D9 ADD EAX,4
004018DC MOV DWORD PTR SS:[EBP-9DC],EAX
004018E2 MOV EAX,DWORD PTR SS:[EBP-9D8]
004018E8 SUB EAX,4
004018EB MOV DWORD PTR SS:[EBP-9D8],EAX
004018F1 JMP SHORT Defender.0040187C ; kryptaussilmukan loppu
004018F3 MOV EAX,DWORD PTR SS:[EBP-9CC] ; laitetaan kryptauksen
004018F9 MOV DWORD PTR DS:[406008],EAX ; tarkastussumma talteen
004018FE PUSH Defender.0040190E
00401903 POP EBX
00401904 JMP EBX
```