LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Diplomityö

Janne Kautiainen

JÄRJESTELMIEN SIIRTÄMINEN UUTEEN KÄYTTÖJÄRJESTELMÄYMPÄRISTÖÖN

Työn tarkastajat: Prof. TkT Heikki Kälviäinen, LUT
DI Juho Mähönen, Honeywell Oy

Työn ohjaaja: DI Juho Mähönen, Honeywell Oy

Tiivistelmä

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Janne Kautiainen

Järjestelmien siirtäminen uuteen käyttöjärjestelmäympäristöön

Diplomityö

2010

77 sivua, 4 kuvaa, 3 taulukkoa ja 1 liite

Työn tarkastajat: Prof. TkT Heikki Kälviäinen, LUT
DI Juho Mähönen, Honeywell Oy

Hakusanat: käyttöjärjestelmä, ohjelmisto, siirtäminen, tietoturva, Windows
Keywords: information security, operating system, porting, software, Windows

Käyttöjärjestelmän uuden version myötä vanhat ohjelmat eivät välttämättä toimi uudessa ympäristössä. Windows-käyttöjärjestelmässä sovellusten yhteensopivuus on aiemmin säilytetty melko hyvin. Uusimpiin Windows Vista ja Windows 7 -versioihin on tehty paljon tietoturvauudistuksia. Niistä johtuen vanhojen ohjelmien yhteensopivuutta on karsittu. Tässä työssä kuvataan automaatiojärjestelmän ohjelmakomponenttien siirtoa uuteen Windows-ympäristöön. Tavoitteena on saada tehtyä ohjeita muille automaatiojärjestelmän kehittäjille. Myös Windowsin tietoturvaominaisuuksiin tehdään katsaus, erityisesti pääsynhallintaan.

Abstract

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Janne Kautiainen

Porting systems to a new operating system environment

Master's Thesis

2010

77 pages, 4 figures, 3 tables and 1 appendix

Examiners: Prof. Dr. Tech. Heikki Kälviäinen
M.Sc.Tech. Juho Mähönen

Keywords: information security, operating system, porting, software, Windows

Legacy programs do not necessarily operate properly in a new environment of an operating system. Previously, application compatibility has been retained on Windows operating systems rather well. Large amounts of data security enhancements have been done to the newest Windows Vista and Windows 7 versions. Because of that, legacy application compatibility has been reduced. In this thesis, porting the legacy software of an automation system to the new Windows environment is described. The objective is to author porting instructions to the members of the automation system development team. Also, the Windows security, especially access control, is studied.

Alkusanat

Työ tehtiin Honeywell Oy:n Kuopion toimipisteessä vuosina 2009-2010. Kiitän työn ensimmäistä tarkastajaa, professori Heikki Kälviäistä, kannustamisesta opintojeni loppuunsaattamiseen, sekä kärsivällisyydestä odottaa diplomityön valmistumista. Kiitokset myös Juho Mähöselle hänen kallisarvoisesta ajastaan työn tarkastamisessa muista työkiireistään huolimatta.

Leppävirta, 30. huhtikuuta 2010

Janne Kautiainen

Sisältö

Lyhenneluettelo

ACE	Access Control Entry
ACL	Access Control List
AM	Access Mask
API	Application Programming Interface
ASLR	Address Space Layout Randomization
ATL	Active Template Library
CC	Common Criteria
CIA	Confidentiality, Integrity, Availability
COM	Component Object Model
DAC	Discretionary Access Control
DACL	Discretionary Access Control List
DCOM	Distributed Component Object Model
DEP	Data Execution Prevention
DLL	Dynamic Load Library
DP	Degree of Portability
HTML	Hypertext Markup Language
IIS	Internet Information Server
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
MAC	Mandatory Access Control
MIC	Mandatory Integrity Control
MSDN	Microsoft Developer Network
.NET	Microsoft ajoympäristö
NX	No Execute
OCX	OLE Control Extension
OLE	Object Linking and Embedding
ORCON	Originator-Controlled Access Control
PC	Personal Computer
PMIE	Protected Mode Internet Explorer
RBAC	Role Based Access Control
RPC	Remote Procedure Call
SACL	System Access Control List
SD	Security Descriptor
SID	Secyrity ID

SQL	Structured Query Language
SRM	Security Reference Monitor
STL	Standard Template Library
UAC	User Account Control
UIPI	User Interface Privilege Isolation
VB	Visual Basic
WEB	World Wide Web
WFP	Windows File Protection
WRP	Windows Resouce Protection
XML	eXtensible Markup Language
XP	Windows XP

1 Johdanto

Tässä kappaleessa kerrotaan työn taustoista, tavoitteista ja rakenteesta.

1.1 Tausta

Ohjelmiston siirtäminen uuteen käyttöjärjestelmäympäristöön aiheuttaa usein haasteita ohjelmien kehittäjille. Jo ohjelmiston siirtäminen tietyn käyttöjärjestelmän uuteen versioon saattaa aiheuttaa ohjelmien yhteensopimattomuutta, saatikka sitten siirtäminen kokonaan erilaiseen käyttöjärjestelmäympäristöön.

Tässä diplomityössä tutkitaan ohjelmistojen siirtämistä Microsoft Windows XP ja Server 2003 -käyttöjärjestelmistä uuteen Windows 7 ja Server 2008 -ympäristöön. XP:n virallisen tuen vähitellen loppuessa käyttöjärjestelmäalustaa täytyy päivittää. Näihin käyttöjärjestelmiin on tullut paljon uusia muutoksia, joiden seurauksena vanhat ohjelmat eivät välttämättä ole enää suoraan yhteensopivia, vaikka Microsoft onkin yrittänyt yhteensopivuutta säilyttää.

Työ tehdään Honeywell Oy:n Kuopion Control Systems -tuotekehitysosaston tarpeisiin. Osastolla kehitetään ohjelmisto- ja laitekomponentteja hajautettuun automaatiojärjestelmään. Näitä tuotteita käytetään monenlaisissa kohteissa prosessiteollisuudessa, erityisesti sellu-, paperi- ja graafisessa teollisuudessa. Järjestelmän ohjelmistot jakaantuvat laiteläheisiin, kontrollereilla toimiviin ohjelmistoihin sekä PC -koneissa (Personal Computer) toimiviin ylemmän tason säätö-, ohjaus- ja tiedonkeruuohjelmiin.

1.2 Tavoitteet ja rajaukset

Työn tavoitteena on

- tutkia, mitä muutoksia olemassa oleviin automaatiojärjestelmän ohjelmiin joudutaan tekemään, jotta ne toimisivat uudessa Windows-käyttöjärjestelmäversioissa.
- tehdä ohjeita ja dokumentteja, joiden avulla ohjelmistotuotteet voidaan muokata yhteensopiviksi uusimman Windows-käyttöjärjestelmän kanssa. Ohjeet on tarkoitettu yhtiön tuotekehitysosaston ohjelmistokehittäjien käyttöön.
- tutkia samalla Windows-käyttöjärjestelmän tietoturvaominaisuuksien toimintaa.

Työn tuloksena on tarkoitus saada muille kehitysyksikön kehittäjille käytännön tietoa, kuinka järjestelmän ohjelmakomponenttien siirto uuteen ympäristöön tehdään.

Osa PC:n ohjelmista on 16-bittiseltä aikakaudelta. Uuden sukupolven 32-bittiset Windows versiot pystyvät ajamaan 16-bittistä koodia, mutta 64-bittiset versiot eivät. 16-bittisten ohjelmien siirtäminen uuteen kohdeympäristöön rajataan tämän työn ulkopuolelle. Ohjelmistolle tehtävään asennusohjelmaan tarvittavia uuden toimintaympäristön vaatimia muutoksia ei myöskään käsitellä tässä työssä.

1.3 Työn rakenne

Kappaleessa 2 kerrotaan taustatietoja aiheesta, esittelemällä ensin automaatiojärjestelmän periaatteellista rakennetta. Sitten luodaan katsaus ohjelmistojen siirrettävyyden ja tietoturvan käsitteisiin. Kappale 3 kertoo uuden Windows-toimintaympäristön, eli kohdekäyttöjärjestelmän uusia ominaisuuksia. Kappaleen loppupuolella tutustutaan Windowsin tietoturvamalliin.

Käytännön osuudessa, kappaleessa 4, kerrotaan käytännön työn eri vaiheista, sekä kerrotaan, minkälaisia ratkaisuja tehtiin ohjelmien siirron mahdollistamiseksi.

Kappaleessa 5 pohditaan työn onnistumista ja mahdollisia jatkotoimenpiteitä. Opinnäytetyön lopussa on yhteenveto.

2 Taustaa

Tässä kappaleessa esitellään tarkemmin hajautettua automaatiojärjestelmää. Sitten tutustutaan ohjelmiston siirrettävyyden käsitteisiin. Lopussa pohditaan ohjelmiston laatua ja tietoturvaa yleisellä tasolla.

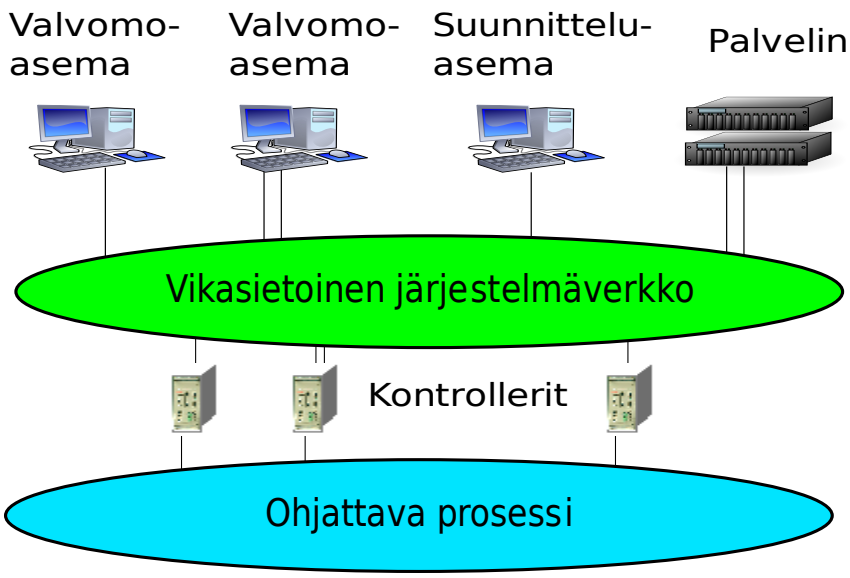
2.1 Automaatiojärjestelmä

Honeywell on maailmanlaajuinen yhtiö, jonka päätoimialueina ovat automaatioteollisuus, ilmailuteollisuus, erikoismateriaalit ja kuljetusteollisuus. Automaatioratkaisujen ja -järjestelmien toimittajana yhtiöllä on tuotekehitysyksiköitä eri puolilla maailmaa. Yksi kehitysyksiköistä sijaitsee Kuopiossa. Aiemmin Suomessa kehitettiin kokonaisia automaatiojärjestelmiä alusta loppuun asti itse. Nykyisin toiminnat on jaettu siten, että järjestelmän perusosa tuotetaan muualla. Paikalliset tuotekehitysyksiköt lisäävät perusosaan oman osaamisalueensa mukaisia osakokonaisuuksia. Näin perusjärjestelmää laajentamalla se saadaan sopeutumaan hyvin erityyppisiin automaatiotarpeisiin. Perusjärjestelmän vahvuus on ollut öljynjalostusteollisuudessa. Kuopion osajärjestelmä on suunnattu nopeisiin koneenohjauksiin, esimerkiksi puunjalostus- ja painoteollisuuteen (pulp, paper and printing).

2.1.1 Rakenne

Hajautetun automaatiojärjestelmän laitteisto voidaan jakaa palvelimiin, asiakaskoneisiin, verkkokomponentteihin, kuten reitittimiin, sekä ohjattavaa prosessia lähellä sijaitseviin kontrollereihin. Käsite 'hajautettu' viittaa siihen, että prosessin ohjaus on jaettu usealle itsenäisesti toimivalle laitteelle. Palvelin- ja asiakaskoneet ovat tyypiltään PC -koneita.

Kuvassa 1 on esitetty järjestelmän periaatteellinen rakenne. Todellisessa järjestelmässä kuvassa nähtävien laitteiden määrä voi vaihdella, riippuen järjestelmän



Kuva 1: Hajautetun automaatiojärjestelmän rakenne

koosta. Laitteet on kytketty toisiinsa Ethernet-pohjaisen vikasietoisen lähiverkon avulla. Palvelin sisältää mm. tietokannan, jossa automaatiojärjestelmän rakennemäärittelyt ja ohjaussovellusten määrittelyt sijaitsevat. Muita palvelimen tehtäviä on prosessidatan ja -hälytysten reititys sekä historiadatan keruu. Vikasietoisuuden lisäämiseksi palvelimet voivat toimia pareittain. Toinen palvelimista on ajovuorossa ja toinen odottaa valmiina omaa ajovuoroaan, jos toiselle palvelimelle tulee häiriö.

Asiakaskoneita ovat operoijien prosessin ohjaukseen ja valvotaan käyttämät valvomo- ja suunnitteluasemat. Prosessin osista on piirretty prosessiohjausnäyttöjä, joiden avulla operoija pystyy näkemään prosessin tilan, säätämään sen parametreja ja antamaan ohjauskomentoja prosessille. Näytöt on toteutettu dynaamisella HTML:llä. Valvomoasemalla on ohjelmisto, jolla operoija voi pyytää tietyn prosessin osakokonaisuuden (näytön) tarkasteltavaksi. Näyttöön voi sisältyä tarkennekuvia, joilla saadaan lisätietoa esimerkiksi toimilaitteen tilasta. Käyttöliittymän näytölle tuleva data voidaan kytkeä suoraan kontrollerille. Näin datan päivitys näyttöön saadaan nopeammaksi, palvelimen kautta kiertävään dataan verrattuna.

Näytöt, prosessisäätöjen toiminta, hälytykset ja muu järjestelmän rakenne suunnitellaan suunnitteluasemalla, josta ne ladataan palvelimen tietokantaan sekä kontrollereille. Suunnitteluasemien tyypillisiä käyttäjiä ovat systeemisuunnittelijat ja prosessi-insinöörit.

Kontrollerit ovat laitteita, joka suorittaa varsinaiset mittaus-, säätö- ja ohjaustoimenpiteet. Ne sijaitsevat lähellä ohjattavaa prosessia. Kontrollerit kytkeytyvät prosessiin I/O- tai kenttäväylän kautta. Kontrolleri sisältää oman prosessorin ja muistia, joten se pystyy suorittamaan sille annettuja toimintoja itsenäisesti. Kontrollerin käyttöjärjestelmä on kehitetty Honeywell:llä.

Tässä työssä tutkittava ohjelmistojen siirtäminen koskee palvelin- ja asiakaskoneilla sijaitsevia ohjelmia. Kontrollerin käyttöjärjestelmä ei muutu, joten sen ohjelmistoja ei tutkita.

2.2 Käsitteiden määrittely

Edellä olevassa tekstissä 'prosessi' tarkoitti automaatiojärjestelmän ohjaamaa ja mittaamaa teollista prosessia, esimerkiksi selluloosan valmistusta, tai paperikoneen ohjausta.

Tästä eteenpäin tässä tekstissä mainittu käsite 'prosessi' tarkoittaa käyttöjärjestelmässä suoritettavaa, käynnissä olevaa tietokoneohjelmaa. 'Ohjelma' on joukko käskyjä, joita prosessin aikana suoritetaan. 'Ohjelmisto' koostuu taas useista eri ohjelmista. Myös käsitettä 'komponentti' käytetään 'ohjelman' kanssa rinnakkain.

Tässä työssä käytetään käsitettä 'perusjärjestelmä', kun tarkoitetaan Kuopion kehitysyksikön ulkopuolella tuotettua automaatiojärjestelmän osaa. Kuopiossa tuotettua järjestelmän osuutta kutsutaan nimellä 'osajärjestelmä'. Kuopion tuottama automaatiojärjestelmä koostuu siis perusjärjestelmästä, johon on lisätty Kuopion tuotekehityksen tuottama osajärjestelmä.

2.3 Ohjelmiston laatu

Ohjelmiston laatu voidaan määritellä ohjelmistolle asetettujen vaatimusten toteutumisena. Laadukkuutta voidaan mitata monin tavoin. Laadun mittarina voidaan käyttää esimerkiksi suorituskykyä, koodivirheiden vähäisyyttä, käytettävyyttä, luotettavuutta, vikasietoisuutta, turvallisuutta ja ylläpidettävyyttä. On huomattava, että ohjelmiston laatu on riippuvainen myös näkökulmasta, josta ohjelmistoa katsotaan. Esimerkiksi loppukäyttäjän näkemys tietyn ohjelmiston laadukkuudesta voi olla hyvinkin erilainen kuin ohjelmiston ylläpitäjän [1]. Yksi laadun mittari on ohjelmiston siirrettävyys eri kohdeympäristöön. Myös tietoturva on osa ohjelmiston laatua. Näistä kahdesta asiasta kerrotaan seuraavaksi.

2.4 Ohjelmiston siirrettävyys

Ohjelmiston siirrettävyys, engl. portability, kuvaa olemassa olevan ohjelmiston saattamista toimimaan uudenlaisessa toimintaympäristössä [2], [3]. Esimerkkejä toimintaympäristöistä ovat vaikkapa käyttöjärjestelmä, prosessoriarkkitehtuuri, ohjelmointikieli, ohjelmointikirjastot sekä kehitystyökalut kuten kääntäjä, linkkeri ja debuggeri [4].

Kirjallisuudesta löytyy siirrettävyyteen läheisesti liittyviä muita käsitteitä. Seuraavassa käsitellään ylläpitoa, evoluutiota, takaisinmallinnusta, uudelleensuunnittelua ja migraatiota.

2.4.1 Ylläpito ja evoluutio

Godfrey ja German [5] vertailevat ylläpidon (maintenance) ja ohjelmiston evoluution eroja. Ylläpito käsitetään ohjelmiston muokkaamiseksi sen julkaisun jälkeen. Se jaotellaan korjaavaan (corrective), sopeuttavaan (adaptive) ja parantavaan (perfective) ylläpitoon. Näistä sopeuttava ylläpito tarkoittaa sellaista ylläpitoa, jossa ohjelma sopeutetaan toimimaan uudessa ympäristössä, mutta ohjelman

toiminnallisuus ei muutu [6]. Ylläpidossa ohjelman alkuperäistä toteutussuunnitelmaa ei ole tarkoitus muuttaa, kun taas evoluutiossa ohjelmistoa parannetaan entisen ohjelman pohjalta sen eliniän aikana. Siihen keksitään lisäominaisuuksia ja sen alkuperäistä suunnitelmaakin saatetaan muuttaa [5].

2.4.2 Takaisinmallinnus, uudelleensuunnittelu ja migraatio

Normaalissa ohjelmistonkehityksessä edetään vaatimusmäärittelystä suunnittelun kautta toteutukseen. Takaisinmallinnuksessa (reverse engineering) edetään päinvastoin [6]. Valmiista tuotteesta yritetään ymmärtää, kuinka se on toteutettu. Edelleen taaksepäin mentäessä yritetään selvittää ohjelman rakenne ja suunnitteluperiaatteet. Jos tiettyjen takasinmallinnusaskelten jälkeen käännetään taas suuntaa, ja muokataan ohjelmaa, puhutaan uudelleensuunnittelusta (re-engineering). Uudistamisessa (restructuring), tehdään vaihtoehtoinen toteutus ohjelman tietylle abstraktiotasolle, esimerkiksi kirjoitetaan vaikeaselkoinen ohjelmakoodi uudelleen. Ohjelman toiminnallisuus ei muutu. Jos uudistamisen yhteydessä muutetaan myös ohjelman kohdetoimintaympäristöä, puhutaan migraatiosta (migration) [6].

2.4.3 Siirrettävyyden lajit

Kirjallisuudessa puhutaan kahdenlaisesta siirrettävyydestä: binääri- ja lähdekoodisiirrettävyydestä [2] [7]. Binääritasolla siirrettävissä oleva ohjelma toimii kohdeympäristössä suoraan, ilman ohjelman uudelleenkääntämistä. Tällöin lähde- ja kohdeympäristö ei voi olla kovin erilainen [8].

Lähdekooditasolla siirrettävä ohjelma toimii ideaalitapauksessa kohdeympäristössään heti kun se on käännetty ja linkitetty uudelleen kohdeympäristön kehitystyökaluilla. Usein lähdekoodia joudutaan kuitenkin muokkaamaan, ennenkuin ohjelma saadaan toimimaan kohdeympäristössä.

Binääri- ja lähdekoodisiirrettävyyden lisäksi on nykyisin olemassa niiden välimuoto, virtuaalinen ajoympäristö. Tällaisessa ympäristössä lähdekoodi käännetään ns. välikoodiksi, jota virtuaalinen ajoympäristö ajaa [9]. Lähdekooditaso on mahdollisimman laitteistoriippumaton. Ajoympäristö on puolestaan riippumaton lähdekoodin kielestä. Tällöin välikoodiksi käännetty ohjelma pitäisi olla ajettavissa missä tahansa ympäristössä, jolle virtuaalinen ajoympäristö on implementoitu. Tällaisia virtuaalisia ympäristöjä ovat mm. Sunin Java ja Microsoftin .NET.

2.4.4 Siirrettävän ohjelman kehittäminen

Siirrettävyyteen kannattaa kiinnittää huomiota jo kehitysvaiheessa [3]. Näin ohjelmiston siirtäminen uuteen ympäristöön helpottuu huomattavasti. Kuitenkin uuden ohjelmiston kehitysvaiheessa joudutaan usein keskittymään toiminnallisuuden toteuttamiseen senhetkiseen kohdeympäristöön, mm. kustannussyistä. Siirrettävyyden pohtiminen jää usein vähemmälle huomiolle. Tämä saattaa kuitenkin kostautua myöhemmin mm. suurempina ylläpitokustannuksina [7].

Siirrettävän koodin kehittäminen ei ole helppo työ. Kun tavoitellaan korkeaa siirrettävyyttä, ohjelmiston muut ominaisuudet saattavat kärsiä. Suorituskyky saattaa jäädä alhaisemmaksi ja joitain toimintoja ei voida toteuttaa ollenkaan. Tämän vuoksi ohjelmistoa kehitettäessä kannattaisi pyrkiä suunnittelemaan ohjelmiston siirrettävyys käytännölliselle tasolle, ei liian siirrettäväksi [3].

Ohjelmakoodin siirrettävyys saadaan paremmaksi, kun pyritään minimoimaan riippuvuudet toimintaympäristöön. Tähän on olemassa seuraavia keinoja [2], [3] :

- Eriytetään ympäristöstä riippuvat koodin osat omaksi osuudeksi, selkeän rajapinnan taakse. Tällöin ympäristön muuttuessa tarvitsee vain muuttaa rajapinnan takainen osuus. Jos mahdollista, parametroidaan ympäristöriippuvuudet, jolloin ne on helppo muuttaa [8].

- Pyritään käyttämään standardinmukaista koodausta. Ei sidota koodia käyttämällä tietyn kehitysympäristön tarjoamia mahdollisesti helppokäyttöisiä, mutta epästandardeja metodeja.

- Käytetään tunnettuja ohjelmointikirjastoja, jotka tukevat useaa toimintaympäristöä. Suhtaudutaan varauksella aivan uusiin kirjastoihin, jotka eivät ole vielä vakiintuneet. Jos mahdollista, tuetaan vaihtoehtoisia kirjastoja, esimerkiksi OpenGL ja DirectX, jotka molemmat ovat grafiikkaohjelmointikirjastoja. OpenGL tukee monia laitealustoja, mutta DirectX on optimoitu Windows-ympäristöön.

- Pidetään siirrettävyys koko ajan mielessä. Asioiden abstraktointi helpottaa siirrettävyyttä. Monet siirrettävyyttä hyvin tukevat ohjelmointikirjastot perustuvat abstraktointiin. Ne tarjoavat ohjelmoijalle joukon luokkia ja rajapintoja, joita vasten ohjelma koodataan. Ohjelman siirtäminen toiseen ympäristöön pitäisi sitten olla vaivatonta, koska ohjelmointikirjasto huolehtii sisäisesti riippuvuudet kohdeympäristöön kohdilleen. Tämä tietenkin edellyttää, että ohjelmointikirjasto tukee kohdejärjestelmää.

2.4.5 Siirrettävyyden kustannukset

Ideaalitapauksessa ohjelmiston siirtämisvaiheessa ei tule kustannuksia ollenkaan. Tällöin ohjelmiston on täydellisesti siirrettävissä. Käytännössä näin ei kuitenkaan ole. Siirrettävyyden vaihtoehtona on ohjelmiston tai ohjelman kokonaan uudelleenkehittäminen. Kustannuksia voidaan pitää siirrettävyyden mittarina, kuten Mooney [2] ehdottaa:

$$DP = 1 - \frac{CP}{CRD} \ , \tag{1}$$

missä

DP = siirrettävyyssuhde (degree of portability)

CP = ohjelmiston siirtämiskustannukset (cost of porting)

CRD = ohjelmiston uudelleenkehityskustannukset (cost of redevelopment)

Ohjelmiston siirrettävyys on huono, jos siirtokustannukset ylittävät kokonaan uudelleenkehittämiskustannukset. Kaavan 1 antaessa negatiivisen tuloksen, siirrettävyys on huono, ja sitä ei kannata tehdä. Lähellä ykköstä olevalla tuloksella ohjelmiston siirtäminen on kannattavaa verrattuna sen uudelleenkehittämiseen.

Siirrettävyyskustannuksiin vaikuttavat mm. nykyisen ja uuden toimintaympäristön erot, sekä se, onko ohjelmisto alun perin suunniteltu siirrettäväksi. Hakuta ja Ohminami [10] ovat ehdottamassaan ohjelmiston siirtokustannuksien arviointimallissa löytäneet kustannuksiin vaikuttavia tekijöitä ja jakaneet ne eri luokkiin: kohdeympäristöstä, ohjelmistosta, kehittäjästä ja kehitysympäristöstä johtuviin kustannuksiin. Kohdeympäristöstä johtuvia tekijöitä ovat mm. prosessoriarkkitehtuurin ja käyttöjärjestelmien erot, sekä ohjelmiston siirrettävyyden taso. Kehittäjästä johtuvia kustannustekijöitä ovat kehittäjän

- tietämys siirrettävästä ohjelmistosta
- tietämys kohdeympäristöstä
- tietämys ohjelmien siirtämisestä yleensä
- ohjelmointikielen osaamistaso
- kehitystyökalujen osaamistaso

Lisäksi ovat vielä kehitys- ja testausympäristöstä johtuvat tekijät, kuten esimerkiksi, onko testitapauksia valmiina, ja onko virheenjäljitys mahdollista.

2.4.6 Olemassaolevan ohjelman siirtäminen

Olemassaolevan ohjelmiston siirrettävyyttä ei voi jälkikäteen suunnitella. Kuinka tällaista koodikantaa kannattaisi sitten alkaa siirtää uuteen ympäristöön? Hook [3] listaa muutaman pääkohdan:

- Ei oleteta, että koodi on siirrettävissä, ennenkuin se on siirretty. Monesti siirtäminen ei olekaan niin helppoa, kuin aluksi kuvitellaan.

- Muutetaan koodia vain niiltä osin, kuin se on välttämätöntä. Liiallinen koodin optimointi tuottaa helposti uusia ohjelmavirheitä aikaisemmin toimivaan koodiin.

- Tehdään hyvä suunnitelma, kuinka siirtäminen tehdään. Erityisesti suurissa ohjelmistoissa, jos lähdetään suin päin korjaamaan koodia, voidaan helposti tehdä sellaisia muutoksia, jotka vaikeuttavat siirtämistä myöhemmässä vaiheessa.

- Käytetään versionhallintajärjestelmää koko siirtoprosessin aikana, ja dokumentoidaan muutokset. Erityisesti suurissa projekteissa, ilman versiohallintajärjestelmää on lähes mahdotonta palata aiempaan, jos tulee tilanne, jossa koodin muutos aiheuttaakin ongelmia jossain toisessa kohdassa ohjelmistoa.

Ohjelmiston siirtoprosessi sisältää hyvin paljon samanlaisia toimenpiteitä kuin normaali ohjelmistonkehityskin [10]: Kohdejärjestelmän, siirrettävän ohjelmiston, ohjelmointiympäristön ja testiympäristön esitutkimus, siirrettävän ohjelmiston muokkaaminen tarvittaessa, kääntäminen ja linkitys, eli ohjelmiston rakentaminen, ja testaus. Lisäksi tarvitaan dokumentointia.

Myös testauksella ohjattavaa ohjelmiston siirtoa (test driven porting) on kokeiltu [11]. Tässä tapauksessa siirrettiin Smalltalk:lla koodattu ohjelmisto Java:lle. Ohjelman toiminnallisuus siirrettiin asteittain. Siirrettävän sovelluksen käyttäytyminen saatiin selville tekemällä sille testitapauksia. Testien tulokset kirjattiin ylös. Koodin tietyn toiminnallisuuden siirron jälkeen testitapaukset ajettiin uudelleen siirretyllä koodilla kohdeympäristössä. Jos testin tulos oli sama kuin alkuperäisessä sovelluksessa, todettiin kyseisen kohdan siirron onnistuneen, ja siirryttiin seuraavan toiminnallisuuden siirtämiseen. Testitapauksia pystyttiin generoimaan

ja ajamaan automaattisesti. Tämän menetelmän etuna pidettiin sitä, että alkuperäisessä sovelluksessa ollut käyttämätön koodi jäi siirretystä ohjelmasta pois, helpottaen tulevaa ylläpitoa. Jos testeillä ei saatu aktivoitua tiettyä koodin osaa, se jätettiin pois ja todettiin tarpeettomaksi.

2.5 Tietoturva

ISO/IEC 27000:n määritelmän [12] mukaan tietoturvalla (information security) käsitetään tiedon suojaamista siten, että tiedon

- luottamuksellisuus (confidentiality): tietoon pääsevät käsiksi vain siihen oikeutetut tahot
- eheys (integrity): tieto ei ole muuttunut tahallisesti eikä tahattomasti
- saatavuus ja käytettävyys (availability): tieto on saatavilla, kun sitä tarvitaan

on taattu. Englanninkielisistä termeistä muodostuu termi CIA, joka on helppo muistaa, kun puhutaan tietoturvasta.

Muita tietoturvaan liittyviä ominaisuuksia ovat ISO/IEC 27000:n mukaan:

- kiistämättömyys (non-repudiation): tiedon muokkaaja ei voi jälkeenpäin kiistää muokanneensa tietoa. Tietoon tehdyistä muutoksista jää jäljet. Tämä on tärkeää esimerkiksi sähköisessä kaupankäynnissä.
- paikkansapitävyys (reliability): tieto on luotettavaa ja uskottavaa
- todentaminen (authenticity): Tietoa käyttävien tahojen luotettava tunnistaminen. Luottamuksellisuus ja kiistämättömyys edellyttävät todentamista.

Tietoturvaan liittyy toinenkin usean eri maan käyttämä standardi ISO/IEC 15408 [13], tunnetaan myös nimellä Common Criteria (CC). Se määrittelee joukon tietoturvaan liittyviä toiminnallisia vaatimuksia. Lisäksi siinä on määritelty tavat, joilla tuotteen tietoturvan tasoa voidaan mitata ja arvioida. Common criteria määrittelee tietoturvaan liittyvät osat seuraavasti:

- Omaisuus (asset). Tämä on se asia/ominaisuus, jota halutaan turvata. Esimerkiksi WEB-palvelin sisältöineen. Sisältö pitää pystyä turvaamaan ja palvelimen on oltava käytettävissä.
- Omistaja. Taho, jolle omaisuudella on arvoa. Omaisuuden menettäminen vahingoittaa omistajaa. Esimerkkinä omistajasta on verkkokauppa.
- Riskit. Omaisuus vaarantuu riskien kautta. Riski voi olla suuri esimerkiksi verkkokaupan käyttämän WEB-palvelimen ohjelmiston vanhentumisen johdosta.
- Uhkat kohdistuvat myös omaisuuteen. Ne kasvattavat riskiä. Uhkana mainitaan vaikkapa tietokonevirukset, jotka saattavat vaarantaa WEB-palvelimen toiminnan, tai sähkökatko, joka ajaa palvelimen alas.
- Vastatoimet. Vastatoimien tarkoitus on pienentää omaisuuteen kohdistuvia riskejä. WEB-palvelimen tapauksessa, vastatoimia ovat esimerkiksi virusohjelmiston asentaminen, palomuurin asetusten tarkistaminen ja virransyötön takaaminen palvelinkoneelle.

2.5.1 Tietoturva käyttöjärjestelmissä

Käyttöjärjestelmän tulee pystyä luotettavasti kontrolloimaan pääsyä järjestelmään ja suojaamaan tietoa joka on sille talletettu. Yleisiä tekniikoita, joita käyttöjärjestelmissä käytetään tietoturvan saavuttamiseksi ovat [14]:

- Muistin suojaaminen. Prosessit eivät saa kirjoittaa toistensa tai käyttöjärjestelmän muistialueille. Tämän estämiseksi käyttöjärjestelmissä käytetään yleisesti virtuaalimuistia, jossa prosessi näkee virtuaalisen osoiteavaruuden fyysisen muistin sijasta. Käyttöjärjestelmä huolehtii virtuaaliosoitteen muuttamisesta fyysiseksi osoitteeksi. Myöskään käytetyn muistialueen uudelleenallokointi ei saa paljastaa vanhaa dataa (object reuse protection). [15]

- Käyttäjäsuuntautunut pääsynhallinta eli käyttäjän todentaminen . Päästäkseen käyttämään käyttöjärjestelmän palveluja, käyttäjän pitää tunnistautua järjestelmään. Todentamiseen kuuluu identiteetin esittäminen (identification) ja identiteetin todistaminen (authentication) Tyypillisesti tämä tapahtuu käyttäjätunnuksen ja salasanan avulla. Käyttäjätunnus kertoo identiteetin, ja käyttäjätunnus on todisteena siitä, että oikea identiteetti on kyseessä.

- Datasuuntautunut pääsynhallinta. Kun käyttäjä on kirjautunut onnistuneesti järjestelmään, hänellä ei kuitenkaan ole automaattisesti pääsyä kaikkiin järjestelmän resursseihin. Järjestelmässä olevan suojattavan resurssin (esimerkiksi hakemisto, tiedosto, prosessi) yhteyteen määritellään tieto siitä, ketkä saavat oikeuden kyseiseen resurssiin. Kirjallisuudessa puhutaan access control list:sta (ACL). Oikeuksia on erilaisia, kuten vaikkapa oikeus lukea, kirjoittaa tai vaikkapa tuhota kyseinen suojattu kohde.

- Suojaus käyttöjärjestelmän eri tiloilla. Prosessoria voidaan ajaa eri toimintamoodeissa. Eri moodeissa on erilaiset pääsyoikeudet koneen resursseihin. Käyttöjärjestelmän käyttämä moodi (kernel mode) takaa pääsyn kaikkiin koneen resursseihin, kuten muisti, I/O, sekä prosessorin hallintarekistereihin. Sovellusohjelmat puolestaan toimivat vähemmän oikeuksia omaavassa tilassa (user mode). Kun prosessoria ajetaan käyttöjärjestelmämoodissa, se pystyy hallitsemaan prosesseja, muistia, oheislaitteita ja keskeytyksiä ilman pelkoa, että käyttäjämoodissa ajettavat prosessit pääsisivät sotkemaan

käyttöjärjestelmän omia tietorakenteita. Sovellusohjelmat pääsevät kiinni esimerkiksi I/O -laitteisiin vain käyttöjärjestelmän tarjoamien palvelujen kautta.

2.5.2 Erilaisia pääsynhallintamalleja

Seuraavassa on katsaus muutamaan yleiseen pääsynhallintamalliin [14], [16], [17].

- Pakotettu pääsynhallinta (Mandatory Access Control, MAC) mallissa suojatut resurssit on luokiteltu eri tietoturvatasoille. Tiedon käyttäjälle on annettu pääsy tietylle tasolle. Käyttäjä pystyy pääsemään käsiksi hänelle määrätyn tason mukaisiin ja sitä alemmalla tasolla oleviin resursseihin. Käyttäjä itse ei pysty määräämään pääsyoikeuksia, vaan ne on annettu ylemmältä taholta.

- Valinnainen pääsynhallinta (Discretionary Access Control, DAC) käyttäjä pystyy itse määrittelemään pääsyoikeuksia omistamilleen resursseille. Tämä on yleisin pääsynhallintamalli tietokoneissa [14].

- Alkuperän mukaan määräytyvä pääsynhallinta (Originator-Controlled Access Control, ORCON), sisältää ominaisuuksia sekä MAC että DAC -mallista. Resurssin alkuperäinen tekijä voi määrätä kuka saa pääsyoikeuden resurssiin.

- Roolipohjainen pääsynhallinta (Role Based Access Control, RBAC). Tässä mallissa käyttäjän rooli, ei käyttäjän identiteetti, määrää pääsyoikeuden resursseihin.

MAC, DAC ja ORCON mallit ovat datakeskeisiä, joissa pääsynhallinta perustuu datan luonteeseen. RBAC -mallissa puolestaan pääsy resurssiin määräytyy roolin tarpeitten perusteella [14].

3 Kohdekäyttöjärjestelmä

Tässä kappaleessa tutustutaan Windows -käyttöjärjestelmään ja sen uusiin ominaisuuksiin lähinnä ohjelmoijan näkökulmasta katsottuna. Nämä uudet ominaisuudethan vaikuttavat ohjelmiston siirtoprosessin onnistumiseen. Kappaleen lopussa selvitetään Windowsin tietoturvamekanismeja.

3.1 Tietoturvan merkitys on kasvanut

Windows on kehittynyt alunperin yhden käyttäjän käyttöjärjestelmästä monen käyttäjän järjestelmäksi. Windows:n alkuaikoina PC oli tyypillisesti yhden käyttäjän käyttämä oma itsenäinen yksikkönsä, jota ei oltu kytketty verkkoon. Tällaisessa eristetyssä ympäristössä tietoturvaa ei katsottu kovin merkittäväksi asiaksi. Nykyisin on toisin. Lähestulkoon mikä tahansa tietotekninen laite on kytkettävissä verkkoon. Samalla tietoturvan merkitys on kasvanut. Myös Microsoft on tunnustanut tämän tosiasian, ja uusimmissa Windows-versioissa on tietoturvalle annettu suuri painoarvo [18]. Tästä syystä suurin osa uusimpiin Windows-versioihin tulleista muutoksista koskee juuri tietoturvaa.

3.2 Käyttöjärjestelmäversiot

Ensimmäinen versio Windows XP:stä julkaistiin 2001. Palvelinkoneisiin suunnattu käyttöjärjestelmä XP:n 'pariksi' oli Windows Server 2003. XP:n jälkeen seuraava suurempi käyttöjärjestelmäpäivitys oli vuoden 2006 lopulla julkaistu Vista, ja sitä vastaava palvelinkäyttöjärjestelmä Windows Server 2008. Äskettäin on julkaistu Vistan seuraaja nimeltään Windows 7. Palvelimissa samaa käyttöjärjestelmäsukupolvea edustaa Windows Server 2008 R2. Näistä uusimmista käyttöjärjestelmistä on olemassa versiot sekä 32- että 64-bittisille prosessoreille. Tässä työssä käsitellään vain 32- bittisiä versioita.

3.3 Windowsin uudet ominaisuudet

Tässä kappaleessa esitellään lyhyesti XP ja Windows Server 2003:n jälkeen tulleet muutokset ohjelmistokehittäjän kannalta katsottuna. Muut muutokset, kuten käyttöliittymä visuaalinen ilme, uudet ja poistuneet sovellukset, ja niin edelleen, jätetään käsittelyn ulkopuolelle.

Microsoftin dokumentaatio [19] ja [20] listaavat ohjelmien yhteensopivuuteen mahdollisesti vaikuttavia ominaisuuksia verrattuna Windows XP:hen, kuten taulukossa 1 nähdään.

Taulukko 1: Ohjelmien yhteensopivuuteen vaikuttavat muutokset käyttöjärjestelmäversioissa

Käyttöjärjestelmä	Muutokset(kpl)
Vista	31
Windows Server 2008	6
Windows 7	11
Windows Server 2008 R2	8

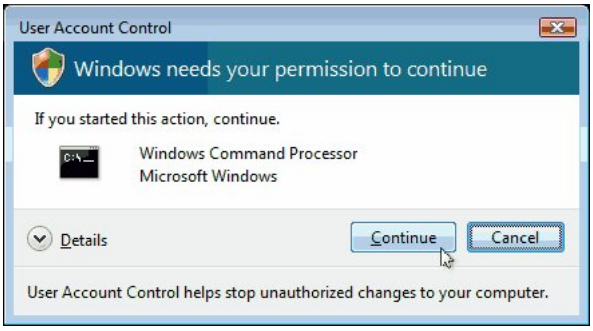
Seuraavassa on listattu noista muutoksista muutamia:

- User Account Control (UAC). Varmasti käyttäjälle näkyvin muutos on User Account Control. Se on mekanismi, jonka avulla yritetään estää (haitta)ohjelmien ajaminen järjestelmävalvojan oikeuksilla. Kaikki käyttäjän käynnistämät ohjelmat käynnistyvät peruskäyttäjän oikeuksilla, myös silloin, kun järjestelmään kirjautunut käyttäjä kuuluu järjestelmävalvojiin. Jos ohjelma oikeasti tarvitsee järjestelmävalvojan oikeuksia, käyttöjärjestelmässä on mekanismi, joka pyytää ne käyttäjältä selvästi näkyvällä tavalla. Näin mikään haittaohjelma ei pysty käyttäjän tietämättä tekemään vain järjestelmävalvojalle sallittuja asioita.

  Kuvassa 2 nähdään esimerkki UAC:n pyytämästä varmistuksesta. Esimerkissä järjestelmävalvojiin kuuluva käyttäjä on käynnistämässä komentotulk-

kia siten, että se saa täydet järjestelmävalvojan oikeudet käyttöönsä. Järjestelmävalvojiin kuulumattomalle käyttäjälle näytetään tuossa tilanteessa dialogi, jossa pyydetään järjestelmänvalvojan tunnusta ja salasanaa.

UAC vaikuttaa asennusohjelmiin ja sellaisiin käyttöliittymällisiin ohjelmiin, jotka vaativat järjestelmävalvojan oikeuksia toimiakseen. Palveluprosesseihin (service) UAC:llä ei ole vaikutusta.



Kuva 2: UAC:n varmennus. Komentotulkkia ollaan avaamassa järjestelmävalvojan oikeuksilla

- Versionumerointi. Käyttöjärjestelmän palauttama versionumero on muuttunut. Suositellaan, että ohjelmat eivät käyttäisi käyttöjärjestelmän versionumeroa ohjelman yhteensopivuuden tarkistamiseen. Sen sijaan ohjelman tulisi tarkistaa, onko tietty käyttöjärjestelmän toiminto tuettu, ja toimia sen mukaan.

- Windows Resource Protection (WRP) suojaa käyttöjärjestelmälle kuuluvia kriittisiä resursseja ylikirjoitukselta ja muokkaamiselta.

- Windowsin internet selain, Intenet Explorer, toimii suojatussa tilassa. Tämä tarkoittaa, että Internet Exploreria käyttävät sovellukset eivät voi kirjoittaa tiedostoja kuin tarkoin rajatulle alueelle hakemistorakenteessa. Tällä pyritään estämään haittaohjelmien pääsy järjestelmään internetiä selattaessa.

- Käyttöjärjestelmä on saatavana myös 64-bittisille prosessoreille. 64-bittisessä ympäristössä pystytään ajamaan 32-bittisiä sovelluksia, mutta ei

enää 16-bittisiä sovelluksia.

- Palvelut (services) ajetaan omassa istunnossaan. Aikaisemmin palvelut ja ensimmäiseksi järjestelmään sisäänkirjautuneen käyttäjän ohjelmat ajettiin samassa istunnossa. Tämä oli tietoturvariski. Nyt palvelut muista ohjelmista erillään ei-interaktiivisessa istunnossa (session 0).

- Web palvelin (Internet Information Server, IIS). Kaikki toiminnallisuus ei ole oletuksena päällä asennuksen jälkeen, kuten aiemmin. Lisäksi tiettyjä komponentteja on poistettu.

- Verkkotoiminnot on kirjoitettu kokonaan uudelleen. Uutena ohjelmointirajapintana on Windows Filtering Platform (WFP), jonka avulla voidaan toteuttaa mm. palomuuritoimintoja. IPv6 on oletuksena käytössä.

- Tuki perinteisille Windowsin help-tiedostoille on poistettu. Ohjetiedostot pitää muuttaa tuettuun formaattiin, joita ovat esimerkiksi .chm ja .h1s

- Käyttäjän henkilökohtaisten tiedostojen hakemistopolut ovat muuttuneet. Sovellusten ei tule käyttää kovakoodattuja polkuja, vaan sen sijaan käyttöjärjestelmän tarjoamaa known folders -mekanismia.

3.4 Windowsin tietoturvamekanismi

Perinteisesti, johtuen osittain myös Windowsin oletusasetuksista, käyttäjät ajavat Windows-ohjelmia suuremmilla käyttöoikeuksilla kuin olisi tarpeen. Windows XP:ssä käyttäjälle asennuksen aikana määriteltiin oletuksena järjestelmänvalvojan (administrator) oikeudet. Jokainen prosessi, jonka käyttäjä käynnisti, peri nämä samat oikeudet. Tietoturvan kannalta tämä oli huono asia. Esimerkiksi Microsoftin nettiselain, Internet Explorer, kärsi tästä syystä monista tietoturvahaavoittuvuuksista. Hyökkäyssivuston kautta oli melko helppoa ujuttaa käyttäjän koneelle erilaisia haittaohjelmia, joiden avulla hyökkääjä saattoi saada koko koneen hallintaansa.

Edellisten kahden vuosikymmenen aikana PC -ohjelmistojen kehittäjät oppivat 'pahoille' tavoille, osin siksi, kun Windows-käyttöjärjestelmä salli sen. Aluksi kehittäjät pääsivät ohjelmoimaan suoraan laitteistotasolle. Kun laitteistotasolle pääsy estettiin myöhemmissä Windows-käyttöjärjestelmissä, niin ohjelmia kehitettiin yhden käyttäjän näkökulmasta [21]. Lisäksi ohjelmat usein kehitettiin käyttäen järjestelmävalvojan oikeuksia, koska kyseiset oikeudet olivat oletuksena käytössä. Ohjelmien toimivuutta peruskäyttäjän oikeuksilla ei vaivauduttu kokeilemaan. Aikataulupaineiden ja kustannusten vuoksi usein oli tärkeintä saada ohjelma mahdollisimman nopeasti valmiiksi.

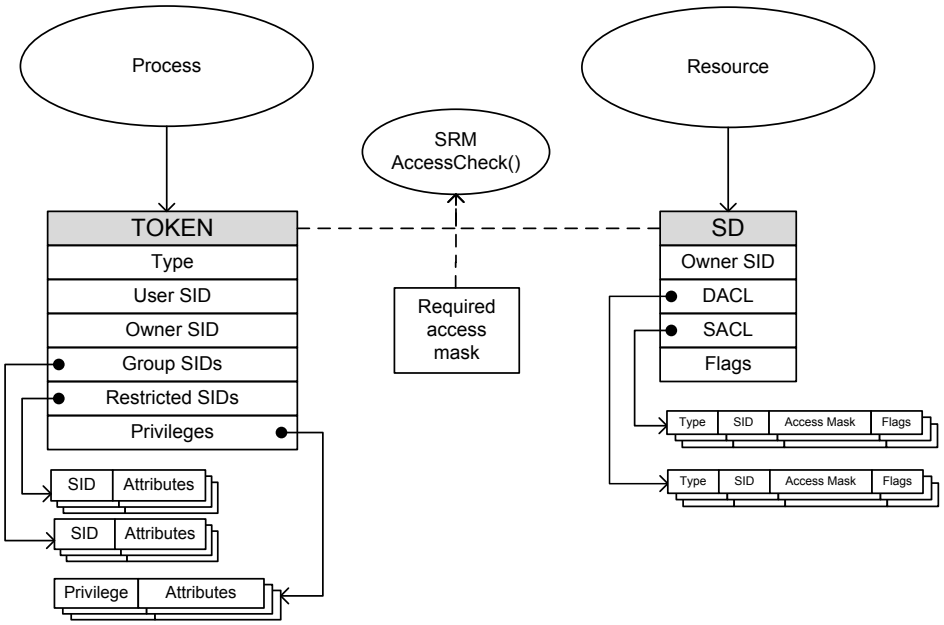
Windows 7 -ympäristö yrittää ohjata ja muuttaa tätä käytäntöä mm. siten, että kehittäjät osaisivat ottaa huomioon ohjelman käyttäytymisen eri käyttöoikeuksilla. Lisäksi käyttäjän käynnistämät ohjelmaprosessit toimivat normaalikäyttäjän oikeuksilla, vaikka järjestelmään olisi kirjautunut järjestelmänvalvojan oikeudet omaava käyttäjä. Täydet järjestelmävalvojan oikeudet prosessi saa vasta niitä erikseen pyydettäessä. Kehittäjiä ohjataan tekemään ohjelmistaan mahdollisimman vähän oikeuksia tarvitsevia.

3.5 Resurssien tietoturva Windowsissa

Seuraavassa kerrotaan Windowsin tietoturvamekanismista resurssien suojauksen näkökulmasta Windowsissa. Vaikka aiheesta on paljon kirjallisuutta, niin tietoturvamallia esitellään tässäkin yhteydessä. Työssä törmättiin usein tilanteeseen, jossa tietty asia ei toiminut oikein, koska prosessilla ei ollut riittävästi oikeuksia haluamaansa resurssiin. Ymmärtämystä aiheesta piti saada kasvatettua. Seuraava selvitys on koottu lähteistä [22], [23], [24], [25], [26] ja [15]. Kuten nähdään, pääsynhallinta Windowsissa ei ole kovin yksinkertainen asia[26]. Aluksi hahmotellaan, kuinka tietoturva toimii XP versiossa, ja sitten katsotaan Vistan ja Windows 7:n myötä tulleet muutokset.

3.5.1 Perinteinen malli

Tietoturvamalliin kuuluvat seuraavassa luetellut asiat (katso kuvaa 3):



Kuva 3: Windows resurssien suojauksen tärkeimmät tietorakenteet

- Käyttöjärjestelmän pääsynhallintaprosessi (security reference monitor, SRM), jota ajetaan käyttöjärjestelmän suojatussa tilassa (kernel mode). Tämä prosessi tarkistaa resurssiin pyydetyt käyttöoikeudet.

- SID. Jokaisella käyttäjällä ja käyttäjäryhmällä on oma identiteetti (security id, SID). SID on vaihtelevanmittainen numeerinen arvo, jonka järjestelmä luo uutta käyttäjää tai ryhmää luotaessa. On myös olemassa yleisiä SID:ejä, kuten esimerkiksi 'järjestelmävalvojat' -ryhmä, jonka SID on S-1-5-32-544.

- Pääsyoikeus. Resursseille (tiedostot, hakemistot, prosessit, jne) voidaan määritellä oikeus, kuka niitä saa käyttää ja miten.

- Etuoikeus (privilege). Termi etuoikeus tarkoittaa oikeutta suorittaa käyttöjärjestelmään kohdistuvia toimintoja. Sitä ei pidä sekoittaa pääsyoikeuteen.

Esimerkkeinä etuoikeudesta mainittakoon oikeus sammuttaa tietokone, tai oikeus muuttaa järjestelmän kellonaikaa.

- Token, eli valtuutus. Kun käyttäjä kirjautuu järjestelmään, hänelle luodaan valtuutus. Käyttäjän käynnistämät prosessit assosioidaan tähän valtuutukseen. Valtuutus sisältää mm. käyttäjän SID:n, käyttäjän jäsenyydet käyttäjäryhmissä SID-listana, etuoikeudet(privilege), valtuutuksen tyypin ja ns. impersonation -tason. Impersonointi on mekanismi, jolla prosessi voi esiintyä toisena käyttäjänä. Tätä käytetään esimerkiksi asiakas/palvelinprosessien välillä siten, että palvelin voi joissain tilanteessa ajaa koodiaan matkimalla asiakkaan oikeuksia. Impersonation -taso kertoo onko toiminto käytössä. Valtuutuksen käyttäjäryhmäjäsenyyslistalla jokaisen SID:n ohessa on lisäattribuutti, joka kertoo kyseisen SID:n tilan. Se voi olla *'enabled'*, *'restricted'* tai *'deny_only'*. Tätä tietoa tarvitaan pääsyoikeustarkastelussa.

- Pääsynhallintalista (access control list, ACL). Tämä lista on tietorakenne, jossa on N kappaletta pääsynhallintamäärityksiä (access control entry, ACE). ACL:iä on kahta tyyppiä. Tavallisella ACL (DACL):llä määrätään pääsyoikeus suojattavaan objektiin. Järjestelmän ACL (SACL) on puolestaan tarkoitettu käytettäväksi, kun halutaan seurata käyttäjän pääsyä suojattuihin objekteihin (auditing). Kukin ACE sisältää

  - tiedon, minkä tyyppinen pääsyhallinta on voimassa. Yleisimmät tyypit ovat *'allow'* ja *'deny'*
  - tietoa, kuinka tämä pääsynhallinta periytyy objektihierarkiassa
  - 4-tavuisen bittimaskin (access mask, AM), joka määrittelee pääsyoikeuden. Näitä ovat esimerkiksi luku, kirjoitus, suoritus, omistajan muutos ja tuhoamisoikeudet.
  - käyttäjän/käyttäjäryhmän SID:n, jota kyseinen pääsyoikeus koskee

- Tietoturvan kuvaaja, eli Security Descriptor, (SD). Tämä tietorakenne on jokaisella suojattavalla objektilla. Se sisältää mm. objektin omistajan SID:n, Järjestelmän ACL:n ja normaalin ACL:n, sekä erinäisiä tilalippuja.

3.5.1.1 Pääsyoikeuden tarkistaminen

Kun käyttäjän käynnistämä prosessi haluaa esimerkiksi kirjoittaa tiedostoon, SRM testaa pääsyoikeuden kyseiseen resurssiin. Pyytäessään pääsyoikeutta, SRM:lle välitetään parametreina resurssin SD, pääsyoikeutta pyytävän prosessin valtuutus ja haluttu pääsyoikeusmaski (AM). SRM palauttaa joko pääsy sallittu tai pääsy estetty -tiedon seuraavassa esitettyjen algoritmien mukaan. Algoritmissa 1 tehdään varsinainen DACL listan läpikäynti. Pääsyntarkastus alkaa kuitenkin algoritmista 2, josta algoritmiä 1 kutsutaan.

Algoritmissa 2 tapahtuu seuraavaa: Jos DACL on tyhjä, resurssia ei ole suojattu Tällöin paluuarvona palautetaan *allow*. Muussa tapauksessa seuraavaksi tarkistetaan onko valtuutuksella etuoikeus ottaa resurssin omistajuus haltuunsa. Jos on, ja *am* ei sisällä muita pääsyoikeuspyyntöjä *allow* palautetaan, ilman että DACL listaa käydään ollenkaan läpi. Seuraavaksi, jos kutsuja on resurssin omistaja, sallitaan oikeus lukea SD(ReadControl) ja oikeus muokata(WriteDaCL) DACL:ia. Myös tässä tilanteessa, jos muita oikeuksia ei oltu pyydetty *am*:ssä, *allow* palautetaan ilman DACL:n läpikäyntiä.

Tämän jälkeen aletaan käymään DACL listaa järjestyksessä läpi (algoritmi 1) . Listalla on siis *allow*- tai *deny* tyyppisiä pääsyoikeuksia. Jos yksikin pyydetty pääsyoikeus löytyy *deny* -tyyppiseltä ACE:lta, pääsy evätään. *Allow* -tyyppisen ACE:n sisältämät oikeudet sallitaan. Jos on päästy listan loppuun, ja joku pyydetyistä pääsyoikeuksista on vielä sallimatta, pääsy evätään. Jos on pääsy on sallittu, ja tokenista löytyy yksikin SID, jonka tyyppi on *restricted*, tehdään vielä toinen kierros pääsyoikeuslistalla, mutta tällä kertaa tarkastellaan vain niitä ACE:tä, joita vastaava SID tokenissa on *restricted*. Molempien kierrosten pitää palauttaa *allow*, jotta pääsyoikeus sallitaan.

Algoritmi 1: Käsittele DACL, lähdettä [15] mukaillen

input : Security descriptor *sd*
Käyttäjän valtuutus *token*
Pyydetty pääsyoikeus *am*
Tähän mennessä sallitut *allowmask*
Testikierros *pass*
output: Pääsyoikeus *allow* tai *deny*

```
testMask = allowmask;
for ace ← sd.DACL[FIRST] to sd.DACL[LAST] do
    if pass == secondpass then
        test = ace.SID ∈ sd.RestrictedSIDs;
    else
        test = ace.SID ∈ token.EnabledSIDs;
    end
    if test == true then
        switch ace.Type do
            case Access Allowed
                if ace.SID ∉ token.deny_onlySIDs then
                    testmask.add(ace.Mask);
                end
            endsw
            case Access Denied
                if am ⊆ ace.Mask then
                    return deny;
                end
            endsw
        endsw
    end
    if am ⊆ testmask then
        return allow;
    end
end
return deny;
```

Algoritmi 2: Pääsyoikeuden tarkastaminen, lähdettä [15] mukaillen

input : Security descriptor *sd*
Käyttäjän valtuutus *token*
Pyydetty pääsyoikeus *am*
output: Pääsyoikeus *allow* tai *deny*

if $sd.DACL == \emptyset$ **then** `// resurssia ei ole suojattu`
  return *allow*
end
$allowmask = \emptyset$;
if $TakeOwnership \in token.Privileges$ **then**
  $allowmask.add(WriteOwner)$;
  if $am \subseteq allowmask$ **then**
    return *allow*;
  end
end
if $sd.OwnerSID == token.UserSID$ **then**
  $allowmask.add(ReadControl)$;
  $allowmask.add(WriteDACL)$;
end
if $am \subseteq allowmask$ **then**
  return *allow*;
end
$access1$ = KäsitteleDACL($sd, token, am, allowmask$,firstpass);
if $access1 ==$ *allow and* $Restricted \in token.SIDs$ **then**
  $access2$ = KäsitteleDACL($sd, token, am, allowmask$,secondpass);
  return $access1$ && $access2$;
else
  return $access1$;
end

Pääsyoikeuksien järjestyksellä DACL listalla on merkitystä. Algoritmin mukaan kun pääsyoikeus on saatu selville, niin listan läpikäynti keskeytetään.

3.5.2 Windows 7:n malli

Vistassa ja Windows 7:ssa on tullut pääsynhallintaan liittyvinä uusina asioina User Account Control (UAC), User Interface Privilege Isolation (UIPI) ja Protected Mode Internet Explorer (PMIE). UIPI:n tarkoituksena on estää alemmalla suojaustasolla toimivan prosessin yritystä lähettää Windows-sanomia ylemmän suojaustason prosessille. PMIE tarkoittaa Internet Explorerin ajamista alemmalla suojaustasolla. Nämä toteutetaan Mandatory Integrity control:n (MIC) avulla. MIC tuo lisäturvaa perinteisen Windows-pääsynhallinnan oheen. Tarkemmin MIC -mekanismista kerrotaan hiukan myöhemmin kohdassa 3.5.2.2.

3.5.2.1 Uusittu valtuutus

Alkaen Vistasta, mihin tahansa 'järjestelmävalvojat' -ryhmään kuuluvan käyttäjän kirjautuessa sisään järjestelmään, hänelle luodaankin aikaisemmasta poiketen kaksi valtuutusta, joista toinen sisältää alkuperäisen valtuutuksen ominaisuudet vain rajoitetusti. Valtuutuksessa rajoitetaan joitain mahdollisesti vaarallisia etuoikeuksia, sekä asetetaan valtuutuksen käyttäjäryhmälistalta 'järjestelmänvalvojat' SID tilaan *deny_only*. Tämä rajoitettu valtuutus assosioidaan sitten käyttäjän käynnistämiin prosesseihin. Lopputuloksena kaikki käyttäjän käynnistämät prosessit saavat rajoitetummat oikeudet. Seuraavassa kerrotaan, kuinka tämä on toteutettu.

3.5.2.2 MIC

Pakotetun pääsynhallinnan oikeustasoja on viisi, lueteltuna kaikkein rajoittavimmasta kaikkein sallivimpaan: 0:Untrusted, 1:Low, 2:Medium, 3:High ja 4:System. Normaalisti prosessit ovat tasolla 2, paitsi suojatun tilan Internet Explorer

(PMIE), joka käyttää tasoa 1. Tasot suojaavat järjestelmää siten, että alemmalla tasolla toimiva prosessi ei pysty kirjoittamaan ylemmällä tasolla oleviin resursseihin. MIC on samankaltainen mekanismi kuin kappaleessa 2.5.2 mainittu MAC.

Tasot on toteutettu määräämällä jokaista tasoa vastaava oma SID. Esimerkiksi 2:Medium tason SID on S-1-16-8192. Oikeustaso on jokaisessa valtuutuksessa ja suojattavan resurssin SD:ssä. Valtuutuksessa sen paikka on käyttäjän ryhmäjäsenyys -listalla. Resurssin SD:ssä se on System ACL -listalla. Jos resurssille ei ole määrätty oikeustasoa, käyttöjärjestelmä käyttää oletusarvoa 2:Medium.

3.5.2.3 Parannettu pääsyoikeuden tarkistaminen

Pääsoikeutta tarkastettaessa tehdään ensin oikeustason (integrity level) tarkastelu. Jos se sallii pääsyn resurssiin, niin sen jälkeen tehdään 'perinteinen' pääsyoikeustarkastelu (DACL), joka kuvattiin algoritmissa 2. Parannettu pääsyoikeuden tarkastus on kuvattu algoritmissa 3.

Algoritmi 3: Pääsynhallinta Vistassa [15]
input : Security descriptor *sd*
Käyttäjän valtuutus *token*
Pyydetty pääsyoikeus *am*
output: Pääsyoikeus *allow* / *deny*

if $token.Integrity < sd.Integrity$ **then**
  return *deny*;
end
return AccessCheck($sd, token, am$);

3.5.2.4 Suojaus käytännössä

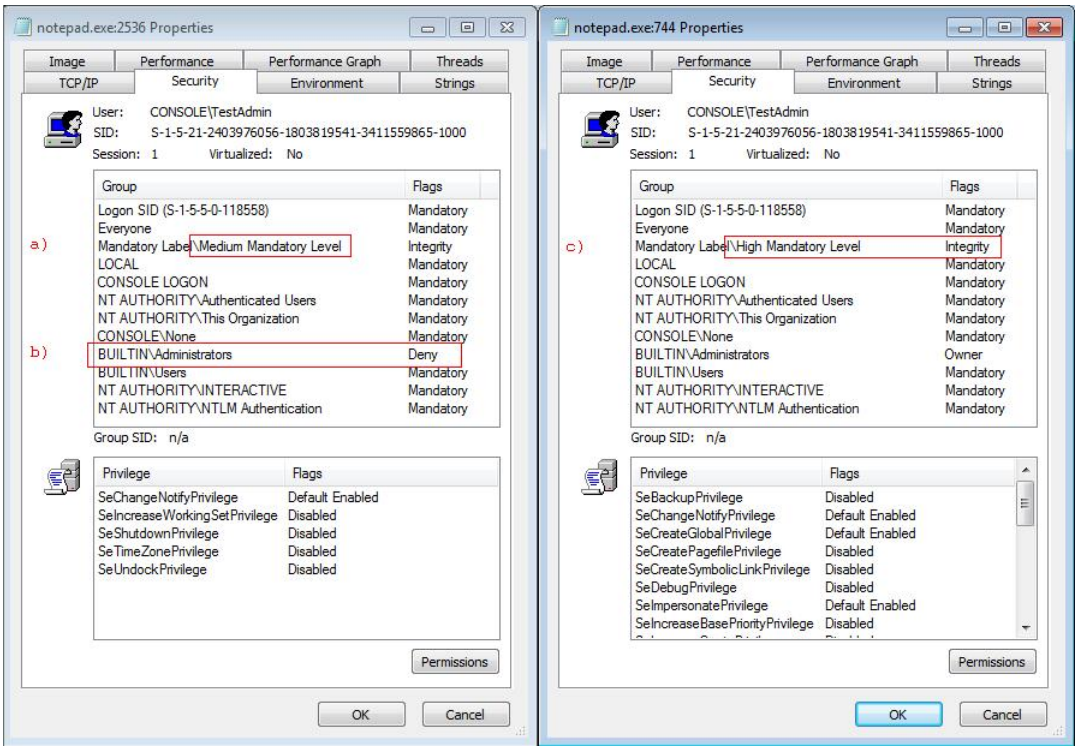
Windowsissa MIC -tasoa ei aseteta tiedostoihin tai hakemistoihin. Tällöin taso 2 määräytyy implisiittisesti. Tämä tarkoittaa, että 2:Medium -tason omaavat prosessit saavat MIC:n kannalta katsottuna kirjoitusoikeuden Windows:n kaikkiin hakemistoihin. Ainoastaan Internet Explorer -ohjelmaa ajetaan 1:Low -tasolla. Sille on varattu tietyt hakemistot, jonne se voi kirjoittaa.

Järjestelmälle kriittisten resurssien, kuten ohjelmatiedostohakemiston, resurssien suojaus toimii normaalikäyttäjien tapauksessa DACL:n kautta, koska MIC ei sitä estä. Kuten aiemmin mainittiin, järjestelmänvalvojan prosessit saavat oletuksena rajoitetun tokenin, eli tokenin, jossa 'järjestelmävalvojat' -ryhmä on $'deny\_only'$ -tilassa. Tällöin, vaikka resurssin DACL sallisikin järjestelmänvalvojille pääsyn resurssiin, niin pääsyoikeutta tarkastettaessa DACL:ssa oleva ACE, joka sallisi pääsyn järjestelmävalvojille, jätetään huomiotta. Näin ollen pääsy järjestelmävalvojat identiteetillä resurssiin estyy.

3.5.2.5 Oikeustason nosto

Tästä päästäänkin UAC:hen ja käsitteeseen 'elevation'. Mitä tehdään järjestelmävalvojalla, joka ei voikaan kirjoittaa järjestelmän kaikkiin resursseihin? Esimerkiksi uuden ohjelman asennuksessa pitää päästä kirjoittamaan ohjelmatiedostot -hakemistoon. Ratkaisuna on saada prosessi käyntiin riittävän korkealla oikeustasolla, jolloin pääsyoikeusongelma katoaa. Prosessin oikeustason nostoa kutsutaan elevoinniksi. Elevointi ja UAC liittyvät toisiinsa. UAC on mekanismi, jolla elevointi toteutetaan. Kun prosessi elevoidaan, sille annetaan sellainen valtuutus, jonka SACL listalla on riittävän suuren oikeustason omaava SID. Järjestelmävalvojat -ryhmän käyttäjän tapauksessa käytetäänkin alkuperäistä valtuutusta, rajoitetun valtuutuksen sijaan. Jos käyttäjä ei kuulu järjestelmänvalvojiin, käyttöjärjestelmä pyytää käyttäjältä järjestelmävalvojat -ryhmään kuuluvan käyttäjän käyttäjätunnuksen ja salasanan. Jos oikea salasana annetaan, käyttäjän käynnistämälle prosessille annetaankin järjestelmävalvojan rajoittamaton valtuutus käyttäjän oman valtuutuksen sijaan. Kuvassa 4 nähdään valtuutusten erot. Vasemmalla puolella on järjestelmävalvojan käynnistämä tekstieditori (notepad.exe) normaalitilassa ja oikealla puolestaan elevoidussa tilassa.

Nähdään, että normaalitilassa tokenin käyttäjäryhmät -listalla oleva järjestelmänvalvojat -ryhmän oikeustasot (MIC) ovat erilaiset. Kuvassa vasemmalla kohdassa a) oikeustaso on $'MediumMandatoryLevel'$ ja kuvassa oikealla kohdassa



Kuva 4: Valtuutus normaalisti ja elevoituna

c) oikeustaso on *'HighMandatoryLevel'* . Samoin nähdään, että normaalitilassa järjestelmänvalvojat -ryhmän tilalippu on *'deny'*, eli estetty. (kuvassa kohdassa b)). Tämä vaikuttaa kuvan vasemmanpuoleisen tekstieditorin pääsyntarkastusta tehtäessä siten, että vaikka resurssin DACL:ssa olisikin järjestelmänvalvojilla oikeudet päästä käsiksi resurssiin, niin pääsy on silti estetty.

Lisäksi havaitaan valtuutusten sisältämien etuoikeuksien olevan erilaiset. Vasemmanpuoleisessa kuvassa olevalla valtuutuksella ei ole kuin yksi etuoikeus sallittuna. Kuvassa 4 etuoikeudet on listattu privilege -osiossa.

3.6 Yhteensopivuus vanhojen Windows-ohjelmien kanssa

Uusi pääsynvalvontamekanismi vaikuttaa vanhojen ohjelmien toimintaan, jos ne eivät ole varautuneet toimimaan normaalikäyttäjän oikeuksilla.

Vuosien saatossa Windows XP:lle on tehty lukematon määrä sovelluksia. Vanhojen ohjelmien yhteensopivuus uusien käyttöjärjestelmäversioiden kanssa on siksi välttämätöntä, XP:lle tehty ohjelmistokanta on aivan liian suuri hylättäväksi vain sen takia, että uusi käyttöjärjestelmäversio ei niitä tue.

Pyrkiessään takaamaan mahdollisimman laajan yhteensopivuuden vanhojen ohjelmien kanssa, Microsoft on tehnyt joitain mekanismeja, joiden avulla vanhoja ohjelmia voidaan ajaa uudessa järjestelmässä ilman uudelleenkääntämistä.

3.6.1 Virtuaalinen XP -tila

Vanhoja, XP:ssä toimivia ohjelmia voidaan ajaa virtuaalisessa XP -tilassa (Virtual XP Mode). Se on toteutettu siten, että Windows 7:ssä ajetaan lisensioitua virtuaalikonetta, jonka käyttöjärjestelmänä on Windows XP. Vanhentuneen ohjelman käynnistäminen käynnistää ohjelmaprosessin virtuaalikoneeseen automaattisesti, joten käyttäjä ei välttämättä edes huomaa virtuaalikoneen olemassaoloa. Tämän mekanismin haittana mainitaan hitaus [27]. Virtuaalinen XP tila on saatavilla vain Windows 7:n tietyille versioille.

3.6.2 Virtuaaliset hakemistopolut

Vanhoissa Windows-ohjelmissa oli usein tapana kirjoittaa esimerkiksi konfiguraatiotiedostoja samaan paikkaan, kuin mihin ohjelma oli asennettu. Tyypillisesti tämä hakemisto sijaitsi hakemistossa c:\program files\. Vistassa ja Windows 7:ssa ohjelmatiedostot sisältävään hakemistoon kirjoittaminen on estetty normaalikäyttäjän oikeuksilla ajettavilta ohjelmilta, osana WRP:tä. Virtuaaliset hakemistopolut on mekanismi, joka sallii kyseisentyyppisen ohjelman kuitenkin toimivan: Kun ohjelma yrittää kirjoittaa suojattuun hakemistoon, käyttöjärjestelmä ohjaakin operaation käyttäjän omaan hakemistoon. Tämä mekanismi on tarkoitettu vain ylimenokauden ajaksi, ja tulevissa versioissa se saatetaan poistaa [22].

Virtualisointi on käytössä vain sellaisissa ohjelmissa, joista puuttuu ns. manifesti. Manifestista kerrotaan lisää kappaleessa 4.4.3.1. Manifesti lisätään ohjelmaan automaattisesti, kun se käännetään Visual Studio 2008:lla.

3.6.3 Ohjelman ajaminen korkeammalla oikeustasolla

UAC:n aiheuttamat rajoitteet voidaan tietenkin kiertää käynnistämällä ohjelma elevoituun tilaan. Se tapahtuu Windowsin käyttöliittymästä klikkaamalla oikealla hiiren näppäimellä käynnistettävän ohjelma ikonia ja valitsemalla esille tulevasta valikosta 'Run as Administrator'. Normaalikäyttäjä ei tietenkään voi ohjelmaa käyttää, ellei tiedä järjestelmävalvojan käyttäjätunnusta ja salasanaa.

4 Käytännön työ

Tässä kappaleessa kerrotaan työn käytännön osuudesta. Alussa kuvaillaan suunnitelmaa, sitten työn eri tekovaiheita. Työn yhtenä tavoitteenahan oli saada tehdyksi komponenttien siirtämisohjeita muille kehittäjille. Ohjeiden teosta ja niiden sisällöstä kerrotaan myöskin tässä kappaleessa.

Ohjelmakomponenttien siirto tapahtui siis Windows -käyttöjärjestelmäversiosta uudempaan versioon. Oli oletettavissa, että olemassa olevien komponenttien siirto olisi paljon helpompaa, kuin jos kohdeympäristö olisi ollut kokonaan eri käyttöjärjestelmä. Jos peilataan tämän projektin luonnetta kappaleessa 2.4 kerrottuihin siirrettävyyden käsitteisiin, niin projekti oli luonteeltaan sopeuttavaa ylläpitoa. Toisaalta projektin aikana tehtiin myös jonkun verran uusia ominaisuuksia, jolloin voidaan puhua myöskin evoluutiosta.

Työn kulku oli pääpiirteissään seuraava:

1. Tutustuminen kohdekäyttöjärjestelmään
2. Tutustuminen siirrettävään ohjelmistoon
3. Tutustuminen kehitystyökaluihin
4. Erilaisia testejä ohjeiden kirjoittamien pohjaksi
5. Käännösohjeen teko
6. Ohjelmakomponenttien sopeuttaminen uuteen ympäristöön
    - Siirto-ohjeen teko
    - Komponenttien siirto
7. Testaus

Edellä kuvatut työvaiheet eivät menneet peräkkäisesti, vaan pikemminkin iteratiivisesti. Erityisesti ohjeita koostettaessa, palattiin ajoittain muutama vaihe taaksepäin tekemään lisää testejä, jotka auttoivat ohjeiden tekoa.

Työ alkoi tutustumalla Windows 7 ja Windows Server 2008:n ominaisuuksiin. Työn alkuvaiheessa oli vielä epäselvää, onko kohdekäyttöjärjestelmänä Windows Vista vai Windows 7. Siksi alkuvaiheessa dokumentit, joihin tutustuttiin, koskivat Windows Vistaa. Dokumenteissa kerrotuista käyttöjärjestelmän ominaisuuksista on kerrottu aiemmin kappaleessa 3.

4.1 Siirrettävään ohjelmistoon tutustuminen

Seuraavaksi tutustuttiin siirrettävään ohjelmistoon. Ohjelmiston laajuudesta johtuen kaikki osat eivät olleet ennestään tuttuja. Koko järjestelmän perusrakennetta on esitetty aiemmin kappaleessa 2.1.1.

Kuten aiemmin kappaleessa 2.2 mainittiin, tässä työssä käsitelty automaatiojärjestelmä koostuu muualla tuotettavasta perusjärjestelmästä, johon lisätään Kuopion tuottama osajärjestelmä. Siten myös uuden, Windows 7 ja Windows Server 2008:ssa toimivan, järjestelmän ohjelmistoasennus tehdään asentamalla ensin perusjärjestelmä, jonka päälle asennetaan osajärjestelmä. Tästä johtuen, myös perusjärjestelmän uusiin ominaisuuksiin tutustuttiin. Samoin, etsittäessä ratkaisuja osajärjestelmän komponenttien mahdollisiin toimintaongelmiin, ratkaisujen piti olla yhdenmukaisia perusjärjestelmän vastaavien ratkaisujen kanssa.

4.1.1 Perusjärjestelmä

Perusjärjestelmän asennusohjelma asentaa omien komponenttiensa lisäksi mm. SQL Server -tietokannan palvelinkoneelle, jota myös osajärjestelmä käyttää. Lisäksi, perusjärjestelmään kuuluu erillinen tietoturvan asennuspaketti, joka

- luo muutamia käyttäjäryhmiä normaalien Windows-ryhmien lisäksi. Automaatiojärjestelmän käyttöön tarkoitetut käyttäjätunnukset liitetään näiden käyttäjäryhmien jäseniksi. Tällä tavalla roolitetaan käyttäjät esimerkiksi operaattoreihin, järjestelmäinsinööreihin tai -valvojiin.
- asettaa ja muuttaa tiettyjä Windows:n tietoturvakäytäntöjä (policy) oletusarvoja tiukemmiksi.
- asettaa tiettyjä Windows-oikeuksia (privilege) oletusarvoista poikkeaviksi.
- määrittelee automaatiojärjestelmälle varatut hakemistot ja rekisteripolut siten, että käyttäjärooleilla on eritasoisia pääsyoikeuksia kyseisiin sijainteihin. Normaaleiden Windows-käyttäjäryhmien pääsyä automaatiojärjestelmälle varattuihin tiedostojärjestelmän osiin rajoitetaan.

4.1.2 Siirrettävät ohjelmistokomponentit

Seuraavaksi selvitettiin osajärjestelmän ohjelmakomponenttien määrät ja tyypit. Taulukossa 2 nähdään eri ohjelmointikielellä toteutettujen komponenttien prosentuaalinen jakauma jaoteltuna käännöksen lopputuloksen mukaan, eli ovatko ne itsenäisesti ajettavia komponentteja (exe), vai kirjastoja (dll ja ocx). Kaikkiaan komponentteja oli vajaa kolmesataa. Taulukko 3 näyttää puolestaan komponenttityyppien suhteelliset osuudet käyttöliittymän perusteella jaoteltuna. Käyttöliittymä, jonka tyyppinä on 'Windows', tarkoittaa exe -tyyppistä ohjelmaa, joka luo ajon aikana ikkunoita, eli on siis perinteinen Windows-ohjelma. 'Konsoli' -tyypin ohjelmien käyttöliittymä on tekstipohjainen ja ne toimivat Windows -komentotulkin alaisuudessa.

Todettiin, että ohjelmointikielenä oli enimmäkseen C++, jonkin verran Visual Basic:ia oli käytetty. Suuri osa ilman käyttöliittymää olevista komponenteista oli COM -komponentteja. Ne oli toteutettu C/C++ -kielellä ATL-kirjastoa hyödyntäen. COM-komponentit voidaan edelleen jaotella normaaleihin

Taulukko 2: Komponenttien suhteellinen jakauma (%)

Kieli	Exe	DLL	OCX
C/C++	15	33	0
C# (.NET)	2	41	0
VB	0	0	9

Taulukko 3: Komponenttien suhteelliset osuudet käyttöliittymän mukaan

Käyttöliittymä	%-Osuus
Ei käyttöliittymää	47
Windows	35
Konsoli	18

COM-komponentteihin ja Windows -palveluprosessin (service) sisällä toimiviin COM-komponentteihin. Komponentteja oli sekä out-of-prosess tyyppisinä exe -komponentteina että in-process tyyppisinä dll -komponentteina.

Windows käyttöliittymälliset ohjelmat oli toteutettu sekalaisesti C/C++, VB ja .NET -tekniikoilla. Konsoliohjelmat oli tehty C/C++:lla.

VB:llä tehdyt komponentit olivat automaatiojärjestelmän käyttöliittymään kuuluvia näyttökomponentteja, tyypiltään ocx. Myös uudemman sukupolven .NET -komponentteja oli joukossa, joista suurin osa dll-kirjastoina. Ne oli kehitetty C# -ohjelmointikielellä.

4.2 Työkaluihin tutustuminen

4.2.1 Virtuaaliympäristöt

Tutustuminen uusiin käyttöjärjestelmäversioihin tehtiin aluksi virtuaalikoneiden avulla. Käytössä oli VMWare -ohjelmisto. Virtuaalikoneiden avulla päästiin kokeilemaan uutta käyttöjärjestelmää ja siitä sai paremman käsityksen, kuin pelkästään dokumentaatiota lukemalla. Virtuaalikoneiden hyviin puoliin kuului myös

helppo ympäristön vaihtaminen: Koneesta voitiin ottaa tietyn hetken tila talteen, sitten voitiin tehdä lisää kokeiluja, ja jos mentiin huonompaan suuntaan, oli alkutilanne helposti palautettavissa. Samoin tietyn ympäristön levittäminen toisten kehittäjien käyttöön onnistui helposti kopioimalla virtuaalikoneen imagetiedostot.

Virtuaalikoneiden käyttämisen huonona puolena on niiden suuri levytilan ja muistintarve sekä hitaus. Yhdelle virtuaalikoneelle joutui varaamaan levytilaa vähintään n. 20GB. Isäntäkoneen työmuistia kului vähintään 1GB / virtuaalikone. Niinpä esimerkiksi yhden palvelin- ja yhden asiakasvirtuaalikoneen ajaminen yhdessä 4 GB:n muistilla varustetussa isäntäkoneessa onnistui juuri ja juuri. Käyttöliittymän vaste oli kyllä ajoittain melko huono.

4.2.2 Kehitystyökalut

Visual Studio 2008:sta on olemassa useampi eritasoinen versio erityyppisille käyttäjille. Perusversion lisäksi on kehittäjille, testaajille, tietokantakehittäjille ja ohjelmistoarkkitehdeille saatavilla enemmän ominaisuuksia sisältävät versiot. Lisäksi on vielä versio, jossa on kaikki ominaisuudet yhdistettynä. Projektissa käytettiin sekä perus- että kehittäjäversiota.

Kehittäjille suunnatussa Development editionissa on perusversion ominaisuuksien lisäksi seuraavat ominaisuudet [28]:

- Työkalu automaattiseen yksikkötestien tekoon. Koodaaja voi tehdä yksikkötestejä kehittäessään uusia ominaisuuksia, siten parantaen koodin laatua. Testit pystytään ajamaan tarvittaessa automaattisesti. Testit tulee suunnitella melko itsenäisesti ajettaviksi, eli ne eivät saa riippua toisistaan. Lisäksi työkalussa on ominaisuus, jolla analysoidaan, kuinka suuri osa koodista saadaan katettua testeillä.

- Profiloija, jolla mitataan koodin suorituskykyä. Profiloijaa käytetään ohjelman ajon aikana, jolloin se kerää monenlaista mittausdataa: esimerkiksi funktioiden kutsukertojen määriä, funktiossa käytettyä aikaa, muistin allokointia, jne. Profiloinnin tuloksia voidaan käyttää apuna etsittäessä suorituskyvyn pullonkauloja.

- Koodin analysoija: Analysoija tutkii käännösvaiheen aikana koodin rakennetta ennalta määriteltyjä sääntöjä vastaan, ja ilmoittaa käyttäjälle, jos se löytää epäjohdonmukaisuuksia. Analysoija yrittää ymmärtää koodin semantiikkaa, kun taas kääntäjä keskittyy koodin syntaksiin.

- 'Code metrics', joka mittaa ja laskee koodista viisi erilaista mittaria: ylläpidettävyys, monimutkaisuus, luokkien periytyvyyden syvyys, luokkien riippuvuus toisistaan ja koodirivien määrän.

Valitettavasti nämä työkalut on suunnattu enemmän .NET -tekniikoille. Toki natiivillekin C/C++ koodille työkaluja löytyi. Koodin analysointityökalu oli integroituna kehitysympäristön käyttöliittymään, mistä se oli helposti käynnistettävissä, mutta esimerkiksi profiloija toimi ainoastaan komentoriviltä käsin. Komponenteista löytyi analysoijan avulla monia potentiaalisia ongelmakohtia, kuten esimerkiksi epäilyttäviä tietotyyppien muunnoksia, alustamattomien muuttujien arvojen käyttöä, puskurin ylivuotoja, mahdollisia muistivuotoja, natiivien API:n funktioiden paluuarvojen huomiotta jättämisiä ja API -funktioiden kutsuvirheitä.

4.2.2.1 Kääntäjän parantunut tietoturva

Kääntäjässä on myös panostettu tietoturvaan. Kääntäjä pyrkii löytämään jo käännösvaiheessa mahdollisia tietoturvaan liittyviä heikkouksia. Löytämänsä epäilyttävät kohdat se ilmoittaa käännösaikaisina varoituksina tai virheinä. Tämä havaittiin ensimmäisiä koodin käännöskokeiluja tehtäessä. Koodi, joka oli aiemmin kääntynyt ilman virheitä, tuottikin uudella kääntäjällä useita virheitä ja vielä useampia varoituksia.

Lisäksi kääntäjässä ja linkkerissä on oletuksena käytössä toimintoja, jotka yrittävät estää mm. erityyppisiä puskurin ylivuodosta aiheutuvia ongelmia. Howard [22] listaa seuraavat kytkimet, jotka pitäisi olla aina käytössä haavoittuvuuksien vähentämiseksi:

- /dynamicbase. Tämä kytkin aiheuttaa käyttöjärjestelmän lataamaan ajettavan koodin, pinon ja keon ennalta tuntemattomaan muistiosoitteeseen. Tekniikka tunnetaan nimellä Address Space Layout Randomization (ASLR)
- /NXCOMPAT. NX, no execute tai Data Execution Prevention (DEP), on mekanismi joka yrittää estää dataa sisältävän muistialueen ajamisen ohjelmakoodina. Kun tämä kytkin laitetaan päälle, linkkeri merkitsee ajettavan koodin ja datan eri tavalla, jolloin käyttöjärjestelmä pystyy estämään datan ajamisen koodina.
- /GS. GS pyrkii estämään pinossa tapahtuva ylivuodon, jonka avulla hyökkääjä saattaisi pystyä ajamaan haluamaansa koodia.
- /SAFESEH. Tämä kytkin liittyy poikkeustilanteen käsittelyyn. Poikkeuskäsittely saadaan turvallisemmaksi.

4.2.3 Aputyökalut

Projektin aikana käytettiin ja analysoitiin aputyökaluja. Analyysissä olivat Microsoftin sovellusten yhteensopivuutta testaavat työkalut, sekä erilaiset ohjelman ajonaikaisten ongelmien selvittämiseen soveltuvat työkalut (muut kuin debuggeri).

4.2.3.1 Application verifier

Microsoftin kehittämä Application Verifier on työkalu ajonaikaisten virheiden löytämiseen testattavasta ohjelmasta. Työkalu toimii siten, että testattava ohjelma

käynnistetään työkalusta käsin. Ohjelmaa ajetaan normaalisti ja käyttäen kaikkia mahdollisia ohjelman ominaisuuksia. Työkalu kerää taustalla virhetilanteita. Täyden hyödyn saadakseen, testattavaa ohjelmaa pitää ajaa debuggerissa. Työkalu osoittautui testien perusteella vaikeakäyttöiseksi. Sen tuottama tieto oli kryptistä, ja sisälsi paljon epäolennaista tietoa. Oleellinen tieto hävisi tietomassaan. Työkalun tulosteen analysointiin ja tulkintaan soveltuvaa ohjetta ei löydetty.

4.2.3.2 Standard User Analyzer

Tämä työkalu on myös Microsoftin kehittämä ja tarkoitettu löytämään UAC:hen liittyviä ongelmia. Se tukeutuu edellä mainittuun Application Verifieriin. Kantavana ajatuksena on löytää testattavasta ohjelmasta ne kohdat, jotka vaativat suurempia oikeuksia kuin mitä normaalikäyttäjällä on. Jos sellaisia löytyy, testattava ohjelma ei toimi kunnolla UAC:n alaisuudessa ilman koodin muokkausta.

4.2.3.3 Process Explorer

Sysinternals:n kehittämä Process Explorer on samantapainen ohjelma kuin Windowsin Task Manager, mutta paljon monipuolisempi. Normaalin järjestelmän prosessilistauksen lisäksi se osaa näyttää yksittäisestä prosessista Task Manageria paljon enemmän tietoa. Projektissa paljon käytettyjä työkalun ominaisuuksia olivat mm. prosessin oikeustason (integrity level) ja valtuutuksen tietojen katseleminen. Sivulla 34 olevassa kuvassa 4 nähdään Process Explorerin esimerkkinäytö. Kyseisessä kuvassa Process Explorer näyttää notepad -prosessin valtuutuksen tietoja.

4.2.3.4 Process monitor

Sysinternals:n kehittämällä Process Monitor:lla pystytään seuraamaan mm. järjestelmän prosessien tiedostojärjestelmän ja rekisterin käyttöä reaaliaikaisesti. Ohjelma listaa jokaisen yrityksen käyttää tiettyä resurssia. Myös tieto resurssin

käytön onnistumisesta näytetään. Koska monitorointi tapahtuu järjestelmänlaajuisesti, sen huomattiin joissain tilanteissa kuormittavan aika paljon järjestelmää, mikä näkyi hitautena. Samoin ohjelman palauttamaa tietomäärää piti suodattaa, jotta oleellinen tieto saatiin esille. Ohjelman avulla saatiin löydettin tilanteita, joissa ohjelma yritti lukea tai kirjoittaa tietoa väärästä paikasta.

4.3 Uuden kehitysympäristön käyttöönotto

Kehittäjille tehtiin kääntämisohjeita sisältävä dokumentti ennen komponenttien kääntämistä ensimmäistä kertaa uudella kääntäjällä. Ohjeiden sisältö saatiin suorittamalla testikäännöksiä sopivasti valituille komponenteille. Ratkaisuja käännösongelmiin etsittiin Microsoftin dokumentaatiosta. Useissa tapauksissa ratkaisu löytyi melko helposti MSDN:n kirjastosta hakemalla dokumentaatiota käännösvirheen numerolla. Toinen tapa oli etsiä tietoa internetistä. Kääntämisohjeen sisällöstä kerrotaan seuraavassa tiivistetysti.

4.3.1 Komponenttien kääntämisohje

Yhtenäisyyden vuoksi kaikille kehittäjille tarvittiin samanlainen kehitysympäristö. Siksi ohjeen ensimmäinen aihe olikin Visual Studion ja tarvittavien ohjelmointikirjastojen asennusohje.

Seuraavaksi ohjeessa neuvottiin käännettävien komponenttien haku versionhallinnasta työhakemistoon. Sen jälkeen ohjeessa oli listattu mahdollisista käännöksen aikaisista ongelmista ja niiden ratkaisuista.

Testien perusteella havaittiin, että vanhat, .NET 2.0 kirjastoon tukeutuvat C# komponentit kääntyivät helposti uuteen .NET versioon 3.5, joten niiden osalta ei tullut suuria ongelmia. Tästä johtuen komponenttien ohjedokumentin teossa keskityttiin enimmäkseen C/C++ kielellä kirjoitettujen komponenttien kääntämisohjeisiin.

Seuraavassa on listattuna ohjeessa käsiteltyjä kääntämisen aikaisia ongelmia.

- Projektin konvertointi. Kun Visual Studio:n vanha projektitiedosto, joka siis sisältää käännöksen määrittelyt (vrt. makefile), avattiin Visual Studio 2008:ssa, Visual studio siirsi projektin uuteen formaattiin.
- Varoitusten ja virheiden huomioimatta jättäminen väliaikaisesti. Projektia käännettäessä virheitä ja varoituksia tuli runsaasti. Jotta käännöstyössä pääsisi eteenpäin, joskus oli aiheellista poistaa joku tietty varoitus tai virheilmoitus väliaikaisesti pois käytöstä.
- Actice Template Library (ATL) -pohjaisten projektien käännösvirheet
- Vanhentuneet metodit. Kääntäjä varoittaa, jos käytetty funktio on vanhentunut, ja suosittelee sen korvaamista uudella funktiolla.
- Kääntäjän paremmasta C++ standardin noudattamisesta johtuvat virheet. Edellisessä kääntäjässä oli paljon ei-standardeja ominaisuuksia, joista johtuen vanha kääntäjä hyväksyi koodin, joka ei enää käänny uudessa versiossa.
- XML kirjaston uuden version käyttöönotto.
- Standard Template Library (STL) muutoksiin liittyvät ongelmat
- Versionumerointi.
- COM -tyyppisten komponenttien käännöksenjälkeinen automaattinen rekisteröinti.

Yllä luetelluista ohjelman käännös- ja linkitysvirheistä ja niiden ratkaisuista on kerrottu tarkemmin liitteessä 1. Se on ote tässä työssä tehdystä kääntämisohjeesta.

Lopuksi ohjeessa neuvottiin, kuinka käännetyt komponentit siirretään takaisin versionhallintaan.

Tämän ohjeen perusteella komponentit pystyttiin kääntämään kohdejärjestelmän kehitysympäristöllä, mutta ohjelmien ajaminen ei ollut vielä mahdollista, koska varsinaista komponenttien siirtoa ei ohjeessa neuvottu.

4.4 Ohjelmien sopeuttaminen uuteen ympäristöön

Seuraavassa vaiheessa lähdettiin kokoamaan tietoa muutoksista, jotka komponentteihin jouduttaisiin tekemään niiden toiminnallisuuden saamiseksi kuntoon kohdeympäristössä. Tämän vaiheen tuloksena syntyi toinen kehittäjille suunnattu ohje. Sisältö kyseiseen dokumenttiin saatiin lukemalla Microsoftin ja perusjärjestelmän dokumentaatio ja yhdistämällä niihin omien testien tuloksista saatu tieto. Dokumentin kokoa rajoitettiin kirjaamalla siihen vain tässä siirtoprojektissa olennaiset asiat. Seuraavassa käydään läpi kyseiseen ohjeeseen tulleita asioita.

Microsoftin mukaan eniten yhteensopivuusongelmia aiheuttavat

- Sovellukset, jotka vaativat järjestelmävalvojan oikeuksia
- Kovakoodatut polut
- Käyttöjärjestelmän versiotarkastus
- Poistetut komponentit käyttöjärjestelmästä
- Palveluprosessien eristäminen

4.4.1 Hakemistopolut

Monet ohjelmat kirjoittivat ajonaikaisia työtiedostojaan ohjelman asennushakemistoon. Kuten aiemmin kappaleessa 3.6.2 on mainittu, tätä ei enää sallita. Ohjelmille varattiin oma hakemisto työtiedostoja varten.

Windowsin ja perusjärjestelmän ohjeistuksissa suositeltiin mahdollisten kovakoodattujen hakemisto- ja rekisteripolkujen korvaamista dynaamisemmalla ns. known folders -mekanismilla: Tunnetut hakemistot, esimerkiksi ohjelmatiedostot, haetaan käyttöjärjestelmän tarjoamalla palvelulla. Palvelulle annetaan parametrina halutun hakemiston tunniste. Myös omia hakemistopolkuja voidaan määritellä.

Windowsin toteutus oli kuitenkin siinä mielessä puutteellinen, että omia rekisteripolkuja ei sen avulla pystytty määrittämään. Tästä syystä kehitettiin apukomponentti, joka osaa myös rekisteripolkujen määrittämisen. Hakemisto- ja rekisteripolkuja tarvitsevat komponentit siirrettiin käyttämään tätä komponenttia. Komponentille tehtiin rajapinnat C/C++, COM .NET ja Visual Basic käyttöön. Tämän mekanismin avulla osajärjestelmän asennusohjelma voi määrittää polkujen fyysisen sijainnin asennuksen aikana. Komponentit kysyvät sijainnin tiettyjä tunnettuja avaimia käyttämällä. Uusia avaimia määriteltiin ohjelmatiedostoja, ohjelmien työtiedostoja ja rekisterikäyttöä varten. Lisäksi kaikki käyttöjärjestelmän määrittämät known folders -avaimet olivat käytettävissä.

4.4.2 Roolit ja ohjelmien oikeudet

Ohjeessa suositeltiin ohjelmien käyttävän pienimmän käyttöoikeuden mallia, tietoturvan parantamiseksi. Neuvottiin karsimaan ohjelmista turhan suuria käyttöoikeuksia vaativat toiminnot pois, mikäli mahdollista. Esimerkkinä rekisteristä lukeminen: ATL-kirjastossa on apuluokka rekisterin avaimen käsittelyyn, nimeltään CRegKey. Rekisteriavaimen avaamiseen on määritelty seuraava funktio:

```
LONG CRegKey:: Open(
   HKEY hKeyParent,
   LPCTSTR lpszKeyName,
   REGSAM samDesired = KEY_READ | KEY_WRITE
)
```

Kuten nähdään, jos funktiota kutsuttaessa viimeinen parametri jätetään pois, se saa saa oletusarvon $KEY\_READ|KEY\_WRITE$. Jos kutsuva koodi, jossa tarvitsee vain lukea rekisterin arvo, on koodattu,

```
LONG ret = key.Open(HKEY_LOCAL_MACHINE, "MyKey");
```

niin kutsuvaa ohjelmaa normaalikäyttäjän oikeuksilla ajettaessa, rekisterin arvon lukeminen epäonnistuu, koska koodissa pyydetään luku- ja kirjoitusoikeutta ja $HKEY\_LOCAL\_MACHINE$ on suojattu normaalikäyttäjän kirjoittamiselta.

Perusjärjestelmä vaatii automaatiojärjestelmän komponenttien käyttävän perusjärjestelmän määrittelemiä rooleja. Roolithan tarkoittavat käytännössä eri käyttäjäryhmiä ja niille määriteltyjä pääsyoikeuksia. Koska projektin yhtenä tavoitteena oli integraation parantaminen perus- ja osajärjestelmän välillä, niin osajärjestelmän ohjelmienkin tuli käyttää samoja rooleja.

Osajärjestelmässä oli jo aiemmin käytetty roolitusta, mutta siinä oli pieniä eroavaisuuksia perusjärjestelmän malliin. Ohjeistettiin osajärjestelmän roolien sopeuttaminen perusjärjestelmän rooleihin.

4.4.3 UAC ja käyttöliittymälliset komponentit

Roolien ja hakemistopolkumuutosten myötä suuri osa UAC:stä aiheutuneista pääsyoikeusongelmista saatiin katoamaan: Ohjelmat eivät enää tarvinneet järjestelmävalvojan oikeuksia tiedostojensa levylle kirjoitukseen, kun roolit takasivat pääsyoikeuden roolin mukaisiin työhakemistoihin.

Jos kaikesta huolimatta ohjelma vaatii järjestelmävalvojan oikeuksia niin neuvottiin muuttamaan ohjelman toiminta seuraavanlaiseksi:

4.4.3.1 Manifestointi

Korkeamman oikeustason vaativan ohjelman pitää pystyä kertomaan käyttöjärjestelmälle, että se vaatii oikeustason nostamista (elevointi). Ohjelma ei voi käynnistymisensä jälkeen enää muuttaa tasoaan. Tämä haluttu oikeustaso asetetaan ohjelman linkityksen yhteydessä ns. manifestissa. Oletusarvona käytetään *asInvoker*. Käyttöjärjestelmä käynnistää prosessin samalle oikeustasolle (integrity level), kuin mikä on käynnistävän prosessin tokenissa. Korkeampaa oikeustasoa vaativille ohjelmille manifestiin määritellään *requireAdministrator*. Tällöin käyttöjärjestelmä osaa pyytää käyttäjältä luvan ohjelman käynnistämiseen elevoituna, eli käyttäjälle näytetään UAC vahvistusdialogi. Tämän jälkeen prosessia ajetaan elevoituna koko sen eliniän ajan. Normaalikäyttäjä ei pysty käynnistämään tällaista prosessia, ellei hän tiedä järjestelmävalvojan käyttäjätunnusta ja salasanaa.

4.4.3.2 Ohjelman jakaminen osiin

Jos ohjelmassa on vain pieni toiminnallinen osuus, joka tarvitsee korkeampaa oikeustasoa, suositellaan että ohjelma jaetaan kahdeksi erilliseksi ohjelmaksi. Perusohjelmalle annetaan manifestissa *asInvoker*, jolloin normaalikäyttäjät voivat sitä käyttää. Suurempia oikeuksia vaativalle ohjelmalle määritellään *requireAdministrator*. Sitten toinen, korkeampaa oikeustasoa vaativa prosessi käynnistetään tarvittaessa perusprosessista ShellExecute() -funktiolla, jolloin Windows pyytää luvan korkeampaa oikeustasoa vaativan prosessin käynnistämiseen. Tämän mekanismin heikkoutena on prosessien välinen kommunikointi.

Toinen mahdollisuus on siirtää korkeampaa oikeustasoa vaativa osuus palveluprosessiin ja kommunikoida sovellusten välillä esimerkiksi RPC:llä.

4.4.3.3 Konsolisovellukset

Jos konsolisovellus vaatii korkeampaa oikeustasoa toimiakseen oikein, sen manifestiin tulee määritellä *asInvoker*. Tällöin normaalikäyttäjä pystyy käynnistä-

mään ohjelman. Sitten ohjelman tulee tarkistaa, onko se käynnistetty elevoituna. Jos on, jatketaan ohjelman suoritusta normaalisti. Jos ei, annetaan virheilmoitus, jossa pyydetään käyttäjää käynnistämään ohjelma elevoituna ja pysäytetään prosessi käyttäen tarkoitukseen sopivaa paluukoodia.

4.4.4 Palveluprosessikomponentit

Osa siirrettävistä komponenteista oli etupäässä palvelinkoneella ajettavia palvelueli service -prosesseja. Näille prosesseille on tyypillistä, että ne ovat käynnissä pitkiä aikoja, usein koko järjestelmän käynnissä oloajan, joten ne ovat houkuttelevia hyökkäyskohteita. Tästä syystä palveluprosessien ajoympäristö on muuttunut uudessa käyttöjärjestelmässä siten, että kaikki palveluprosessit ajetaan omassa istunnossaan, erillään normaaleista ohjelmaprosesseista. Tästä johtuen palvelu ei voi luoda käyttöliittymää, eikä se voi kommunikoida sovelluksen kanssa Windowsin viestien välitysmekanismilla (SendMessage()). Tällaisessa tilanteessa neuvottiin käyttämään jotain muuta kommunikointimekanismia, esimerkiksi RPC, nimetyt putket tai COM.

Serviceprosesseja varten on käyttöjärjestelmään varattu kolme omaa erikoiskäyttäjätunnusta. Niillä ei ole salasanaa. *Localsystem* -tunnuksella on kaikkein suurimmat oikeudet. Siksi sen käyttöä tulee välttää. Sen sijaan suositellaan käytettävän vähemmän voimakkaita *localservice* tai *networkservice* -tunnuksia. Myös normaalia käyttäjätunnusta voi käyttää, mutta salasanojen hallinta voi olla ongelma.

4.4.5 Winhelp-ohjelman poistuminen käytöstä

Uusimman Windows-version myötä Windowsin ensimmäinen aputiedostomuoto (.hlp -tiedostot) ei ole enää tuettu. Aputiedostot pitää muuttaa johonkin toiseen käyttöjärjestelmän tukemaan formaattiin, esimerkiksi .chm.

4.4.6 Windows-sanomien välitys prosessien välillä

Windowsin viestienvälistyksellä (SendMessage() ja vastaavat funktiot) tapahtuva kommunikointi ei enää välttämättä toimi. Sen estää Windows 7:n UIPI -mekanismi: Alemmalla suojaustasolla oleva prosessi ei voi lähettää viestejä ylemmän tason prosessille.

4.5 Komponenttien siirtäminen

Kun dokumentti oli valmis, komponenttien siirto voitiin aloittaa. Mikäli mahdollista, komponentin alkuperäinen kehittäjä teki siirron kehittämilleen komponenteille. Tiettyjen komponenttien osalle tämä ei ollut kuitenkaan mahdollista. Vieraan koodin siirtäminen oli jonkin verran työläämpää, koska siirtäjän piti ensin perehtyä ohjelman toimintaan.

Havaittiin, että eniten koodimuutoksia aiheutti hakemisto- ja rekisteripolkuihin liittyvät muutokset.

4.6 Testaus

Komponenttien riippuvuus toisistaan vaikeutti modulitestausta: Ei ollut helppoa ottaa vain yhtä komponenttia kokeiltavaksi, koska sen toiminnallisuus riippui useista muista komponenteista. Piti saada kerralla sopiva komponenttijoukko käännetyksi ja asennetuksi kohdeympäristöön. Tätä varten tehtiin asennusohjelma, jolla Windows 7 ja Windows Server 2008 -ympäristöön siirretyt ohjelmat pystyttiin asentamaan kohdeympäristöön. Asennusohjelma tehtiin myös uusia työkaluja käyttäen, mutta sen toteutukseen ei oteta tässä työssä kantaa. Projektin aikana testibuildeja tehtiin useita kappaleita. Buildi tehtiin aina, kun sopiva ohjelmien osakokonaisuus oli saatu siirrettyä. Testaus tehtiin siis iteratiivisesti. Testeissä löytyneet virheet korjattiin mahdollisimman aikaisessa vaiheessa.

5 Työn tulokset ja pohdintaa

Tässä kappaleessa kerrotaan työn tulokset, pohditaan työtä yleisesti sekä työn jatkon kannalta.

5.1 Työn tulokset

Kuopion toimipisteen ohjelmistonkehittäjille tehtiin kaksi ohjetta, joissa ohjeistettiin Kuopion tuottaman automaatiojärjestelmän osajärjestelmän ohjelmistokomponenttien ohjelmien siirtämisestä uuteen Windows 7 ja Windows Server 2008 -käyttöympäristöön. Ensimmäisessä ohjeessa keskityttiin ohjelmien kääntämisaikaisiin ongelmiin ja annettiin niihin ratkaisumalleja. Toisessa ohjeessa annettiin käytännön neuvoja, kuinka ohjelmat saadaan muokataan uuden toimintaympäristön sovellusohjelmille asettamia vaatimuksia vastaaviksi. Komponenttien siirto uuteen ympäristöön tehtiin ohjeiden opastamana.

Microsoftin ja muiden osapuolten Windows 7 ja Server 2008 -ohjeistusta oli paljon saatavilla. Tietotulvasta vain projektin kannalta olennaisen tiedon kerääminen kootusti ohjedokumentteihin helpotti muiden kehittäjien työtä. Komponenttien yhdenmukaiset ratkaisumallit helpottavat ohjelmien ylläpitoa tulevaisuudessa.

Komponentti automaatiojärjestelmän työhakemistojen hallintaan kehitettiin. Sen avulla automaatiojärjestelmän ohjelmat ohjataan käyttämään sallittuja hakemisto- ja rekisterisijainteja. Hakemistojen sijainnin muuttaminen tarvittaessa voidaan tehdä helposti osajärjestelmän asennusohjelmassa, koska komponenteilla ei ole enää kovakoodattuja polkuja.

Komponenttien tietoturva parantui muun työn ohessa, koska iso osa uuden käyttöjärjestelmän vaatimista muutoksista koski tietoturvaa. Myöskin uudet kehitystyökalut tuottavat jo oletusasetuksilla aikaisempaa turvallisempaa koodia.

Työn tekemisessä tutustuttiin Windows-tietoturvan toimintaan käytännön tasolla. Erityisesti pääsynhallintamekanismin toiminnan ymmärtäminen auttoi siirtoprojektin dokumenttien teossa ja uuden ympäristön aiheuttamien ohjelmien ajonaikaisten ongelmien ratkonnassa. Kaksi hyvää, yleiskäyttöistä analysointityökalua, Process Monitor ja Process Explorer, otettiin käyttöön.

Ajattelutapa tietoturvaa kohtaan muuttui. Aiemmin ohjelmia kehitettäessä tietoturvan ajateltiin usein olevan jotain, joka 'tulee tielle ja estää ohjelman toiminnan'. Tietoturva-ajattelu tulee olla koko ajan mielessä, kun ohjelmia kehitetään ja ylläpidetään.

5.2 Pohdintaa

Jo projektia aloitettaessa työ tuntui haasteelliselta. Uusia opeteltavia asioita oli paljon: Uusi kohdeympäristö, uudet kehitystyökalut, Windows:n pääsynhallintamalli ja virtuaalikoneet. Myöskään kaikkien siirrettävän järjestelmän ohjelmakomponenttien toteutusteknologia ei ollut tuttua.

Opinnäytteestä tuli melko tietoturvapainotteinen. Painotusta voi perustella sillä, että kohdejärjestelmän uudet muutokset olivat nekin suurimmalta osin tietoturvaan liittyviä, ainakin ohjelmankehittäjän näkökulmasta katsottuna.

Projektin aikana kunnollisten dokumenttien merkitys huomattiin taas kerran. Kehittäjälle vieraan komponentin siirtäminen oli huomattavasti helpompaa, jos siitä oli saatavilla hyvät dokumentit.

5.3 Jatkokehitys

Siirtoprojektissa tuli esille myös sellaisia ongelmia, joita ei oltu mainittu projektille tehdyssä ohjeistuksessa. Nämä olivat kuitenkin yksittäistapauksia, ja koska projekti oli kertaluonteinen, niin ohjeiden jatkokehitys ei ole kannattavaa.

Kuitenkin tässä työssä käytettyä mallia voitaisiin käyttää myös tulevaisuudessa käyttöympäristön muuttuessa. Ennen ohjelmien siirtoa tapahtuvaan ohjeistuksen tuottamiseen kannattaa panostaa. Tällöin vältytään varsinaisessa siirtotilanteessa siltä, että kukin kehittäjä etsii ratkaisuja samoihin ongelmiin kukin omalla tahollaan.

Nyt ohjelmiin saavutettu tietoturvan paraneminen on tullut etupäässä sopeutumisesta käyttöjärjestelmän vaatimuksiin. Ohjelmien tietoturvaa voisi edelleen parantaa. Hyviä ja käytännöllisiä vinkkejä Windows käyttöjärjestelmälle löytyy kirjallisuudesta, esimerkiksi [18] ja [22]. Yleisemmällä tasolla asiaa käsitteleviä kirjoja ovat esimerkiksi [14] ja [17].

Ohjelmien edelleen kehittämistä aina vain turvallisemmiksi voisi jatkaa loputtomiin, jos vain aikaa ja rahaa riittäisi. Kuitenkin, reaalimaailmassa joudutaan arvioimaan jatkokehityksen hyötyjä suhteessa siihen kuluviin resursseihin.

6 Yhteenveto

Tämä työ perustui tarpeeseen saada automaatiojärjestelmän Windows XP ja Windows Server 2003 -käyttöjärjestelmissä toiminut ohjelmisto siirretyksi uudemman sukupolven Windows 7 ja Windows Server 2008 -käyttöjärjestelmiin. Uuden käyttöjärjestelmäversion käyttöönotto aiheutti aikaisemmin toimineiden ohjelmien toimimattomuutta. Työssä tutkittiin, millä tavalla ohjelmia tulisi muuttaa, että ne olisivat yhteensopivia uuden käyttöjärjestelmän ohjelmille asettamien vaatimusten kanssa. Suurin osa tarvittavista muutoksista liittyi uuden käyttöjärjestelmän parantuneeseen tietoturvaan. Windows -käyttöjärjestelmän tietoturvan toimintaperiaatetta tutkittiin, erityisesti pääsynhallinnan osalta.

Työn tuloksena saatiin dokumentoitua kehittäjille suunnattuja ohjeita, joita hyväksi käyttäen ohjelmien siirto saatiin tehdyksi. Koottujen ohjeiden ansiosta yksittäisten kehittäjien työ nopeutui. Yhtenäiset muutokset komponentteihin helpottavat ylläpitoa. Siirron aikana tehtyjen muutosten johdosta ohjelmien laatu tietoturvan kannalta katsottuna saatiin paremmaksi. Myös ymmärrys Windows -käyttöjärjestelmän tietoturvamekanismeista kasvoi.

Viitteet

[1] Sagar Naik and Piyu Tripathy. *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, 2008.

[2] James Mooney. Bringing portability to the software process. `http://www.cs.wvu.edu/~jdm/research/portability/reports/TR_97-1.pdf`, 1997. Viitattu 4.3.2010.

[3] Brian Hook. *Write Portable Code: An Introduction to Developing Software for Multiple Platforms*. No Starch Press, 2005.

[4] Judith Bishop and Nigel Horspool. Cross-platform development: Software that lasts. *Computer*, 39(10):26–35, 2006.

[5] D.M. Godfrey, M.W. German. The past, present, and future of software evolution. In *Frontiers of Software Maintenance, 2008*, pages 129 – 138. IEEE, 2008.

[6] Hans van Vliet. *Software Engineering: Principles and Practice*, chapter 14. John Wiley & Sons, 2008.

[7] Piia Ruokonen. Portable application user interface development in symbian os. Master's thesis, Lappeenranta University of Technology, 2002.

[8] James Mooney. Portability and reusability: common issues and differences. In *CSC '95: Proceedings of the 1995 ACM 23rd annual conference on Computer science*, pages 150–156. ACM, 1995.

[9] K John Gough. Stacking them up: a comparison of virtual machines. In *Proceedings of the 6th Australasian conference on Computer systems architecture*, ACM International Conference Proceeding Series, pages 55–61. IEEE Computer Society, 2001.

[10] Mitsuari Hakuta and Masato Ohminami. A study of software portability evaluation. *Journal of Systems and Software*, 38(2):145–154, 8 1997.

[11] Ralph Bohnet and Gerard Meszaros. Test-driven porting. In *Proceedings, Agile 2005*, pages 259 – 266. IEEE, 2005.

[12] Information technology - security techniques - information security management systems - overview and vocabulary. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c041933_ISO_IEC_27000_2009.zip`, 2009. Viitattu 31.1.2010.

[13] Information technology - security techniques - evaluation criteria for it security - part 1: Introduction and general mode. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c050341_ISO_IEC_15408-1_2009.zip`, 2009. Viitattu 31.1.2010.

[14] S Borsworth et al. *Computer Security Handbook*. John Wiley & Sons, fifth edition, 2009.

[15] Mark Russinowitch et al. *Windows Internals*. Microsoft Press, fifth edition, 2009.

[16] Ronald Krutz and RUssell Vines. *The CISSP Prep Guide*. John Wiley & Sons, gold edition, 2003.

[17] Jatinder Gupta and Sushil Sharma. *Handbook of Reseach on Information Security and Assurance*. IGI Global, 2009.

[18] Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, 2 edition, 2008.

[19] Anon. Application compatibility. `http://msdn.microsoft.com/en-us/library/bb757005.aspx`. Viitattu 31.1.2010.

[20] Anon. Windows 7 and windows server 2008 r2 application quality cookbook. http://code.msdn.microsoft.com/Release/ProjectReleases.aspx?ProjectName=Windows7AppQuality&ReleaseId=1734. viitattu 31.1.2010.

[21] Peter Coffee. Playing by vista's rules. *eWeek*, 23(48):65, 2006.

[22] Michael Howard and David LeBlanc. *Writing Secure Code for Windows Vista*. Microsoft Press, 2007.

[23] M Heward and D Pravat. *Advanced Windows Debugging*. Addison-Wesley, 2007.

[24] Msdn libary, security aliotsikko. http://msdn.microsoft.com/fi-fi/library/ee663293(en-us,VS.85).aspx. Viitattu 25.3.2010.

[25] J.R. Michener. Common permissions in microsoft windows server 2008 and windows vista. *Security & Privacy, IEEE*, 6:63 – 67, 2008.

[26] Prasad Naldurg et al. Netra:: seeing through access control. In *Workshop on Formal Methods in Security Engineering*, Proceedings of the fourth ACM workshop on Formal methods in security, pages 55–66. ACM, 2006.

[27] Michael Scalisi. What you need to know about xp mode. *PC World*, 28(1):28, 1 2010.

[28] Lars Powers and Mike Snell. *Microsoft Visual Studio 2008 Unleashed*, chapter 26. Sams, 2008.

Liite 1.

Common problems in project compiling

Unfortunately, only a few C++ projects compile without errors (and warnings). Below are listed some common problems, and how they could be avoided.

General note of eliminating compiler warnings

There are some compiler switches which could be used to turn off the specific warning. Even some errors can be avoided by compiler switch. Although eliminating compiler warnings is not strictly necessary to build project in VS2008, it is strongly recommended. Fixing warnings will increase code quality. Although there may be hundreds of errors listed, or more, the actual amount of effort required to eliminate all warnings may be less than expected, as errors may be repeated multiple times. For instance, if a certain header file generates 10 compiler warnings, and is included in 10 different projects, it will result in 100 warnings when the entire solution is compiled, but only require 10 changes to eliminate all errors. The majority of compiler warnings will be generated as a result of using older API calls which have been replaced by new secure versions.

If some warning is easily fixed, it should be done. On the other hand, if the error fixing requires large amount of work, and the code has been working earlier, you could 'hide' the warning message with proper compiler switch. But when writing completely new code, all the warning messages shall be taken seriously.

ATL based projects:

<x>.cpp is obsolete

statreg.cpp is obsolete. Please remove it from your project.
atlimpl.cpp is obsolete. Please remove it from your project.

(jatkuu)

-those were typically #included in stdafx.cpp. Remove both lines in stdafx.cpp

CRegKey

(Warning C4996.) The compiler encountered a function that was marked with deprecated. The function may no longer be supported in a future release. ATL CRegKey class has some method changes. E.g. the QueryValue() –method has been superseded with QueryStringValue() or QueryMultiStringValue().

Fix: Change all CRegKey::QueryValue() –calls to either of the new methods as appropriate. Note that the order of parameters is different!

_ATL_MIN_CRT is no longer supported.

Cause: Excerpt from MSDN: "ATL cannot be built without a dependency on CRT. In earlier versions of Visual Studio, you could use #define ATL_MIN_CRT to make an ATL project minimally dependent on CRT. In Visual C++ 2008, all ATL projects are minimally dependent on CRT regardless of whether ATL_MIN_CRT is defined."

Fix: remove ATL_MIN_CRT definition to get rid of the warning.

DEFINE_COMMAND macro

warning C4995: 'DEFINE_COMMAND': name was marked as #pragma deprecated. Use DEFINE_COMMAND_EX instead.

Strings and string functions

Wchar_t

By default, wchar_t is handled as built-in type That means that wchar_t* != WORD*, and wchar_t* != USHORT*. This causes a lot of error messages.

(jatkuu)

Temporary fix: (in order to continue compiling: project settings/Config properties/C/C++/Language/Treat wchar_t as buildin =NO)

Permanent fix: Modify the code so that wchar_t* is used properly? (Could be a big task.)

Deprecated methods

Symptom

A compile time C4996 warning is generated, which may look like:" warning C4996: 'wcscat' was declared deprecated C:\Program Files\Microsoft Visual Studio 8\VC\include\string.h(243) : see declaration of 'wcscat' Message: 'This function or variable may be unsafe. Consider using wcscat_s instead. To disable deprecation, use _CRT_SECURE_NO_DEPRECATE. See online help for details.' "

Cause

Many of the older functions of the C Runtime have been deprecated because of security holes (primarily the danger of buffer overruns, as many of these functions do not check the size of the buffers before writing information to them).

Resolution

There are several ways to resolve these errors. Although the warnings can be disabled, this is not recommended, as the potential for a security hole is not removed. If the input to these functions has already been validated, a change may not be necessary. Although deprecated, the old functions are still available, to provide backwards compatibility with legacy code. In all other cases, refer to the error message generated by the compiler, which will generally provide

(jatkuu)

the name of an alternate, secure function which can be used in place of the old function (many simply have _s suffixed to the old function name). These functions usually require an additional parameter, which specifies the maximum size of the buffer being copied to. Refer to the documentation for each function for specifics (pay close attention to the extra size parameter—some functions take a number of bytes to be copies, many string functions take the maximum number of characters to be copied; be sure to provide the correct parameter to take advantage of the increased security of these functions).

Non standard extension for enumeration

There is no need to specify enumeration name when enum is defined inside the type. More details of this can be found at the following location. A sample code below generates the warning as below warning C4482: nonstandard extension used: enum 'IControl::ePlatforms' used in qualified name.

```
class IControl
{
public:
    // enumeration for platforms supported
    enum ePlatforms
    {
        NOPLTFM =       0x00000000,
        C200 =          0x00000001
    };

    IControl();
    void Test(IControl::ePlatforms ePlt)
    {
      if (ePlt == IControl::ePlatforms::C200) // warning here
```

(jatkuu)

```
      {
      }
    }
};
```

The fix has to be done by removing the enumeration name as below

```
void Test(IControl::ePlatforms ePlt)
    {
      if (ePlt == IControl::C200)
      {
      }
    }
```

Error C2065: <var> undeclared identifier

If the variable of a for loop is declared in for(..) –statement, the variable goes out of scope after the loop. Example:

```
for (int var=0;var<5;var++)
{
do_something(var);
}
do_something_else(var);   //This var is now out of scope
```

Fix: define the variable outsize the for-statement

```
int var =0;
for (var=0;var<5;var++)
{
```

(jatkuu)

```
do_something(var);
}
do_something_else(var);  //var is still in scope
```

There is also a temporary fix (to temporary ignore this error and continue compiling): Project settings/Config properties/C/C++/Language/Force Conformance in for loop scope = No. Remember to set this back to Yes, after other part of the code compiles!

C4430: missing type specifier -int assumed

Cause: All declarations must now explicitly specify the type; int is no longer silently assumed. e.g.

```
GetSomeValue (void)
{
  Int iret=3;
  return iret;
}
```

Fix: specify the correct type for the variable / function. (or #pragma warning(disable : 4430)) e.g.

```
int GetSomeValue (void)
{
  Int iret=3;
  return iret;
}
```

(jatkuu)

ISO C++ Conformant name

VS2008 compiler is ISO/ANSI conformant and there would be warnings related to the using ISO conformant name. One of such example is shown below

```
char szbuff[20];
itoa(10,szbuff,10);
```

The usage of itoa function will result in the warning C4996: 'itoa': The POSIX name for this item is deprecated. Instead, use the ISO C++ conformant name: _itoa. See online help for details. Fix such code to use conformant names as the warning itself will lead to usage of proper names.

Resolve data type mismatch warnings

There could be several warnings related to data type mismatch in comparison or assignment of a variable or return value of the function and the actual value being returned. Simple assignment of double to float will give warning C4244: '=' : conversion from 'double' to 'float', possible loss of data

```
float fVal = 10;
double dVal = 15;
fVal = dVal;
```

Please resolve such warnings either by static_cast or trying to redefine the data type of the variables.
Also C4018: '>=' : signed/unsigned mismatch is caused by stronger type checking than in previous VC++ compiler.

(jatkuu)

XML

Remove #include <msxml2.h> But leave using namespace MSXML2; If you get errors like C2872: 'IXMLDOMAttributePtr' : ambiguous symbol, then you could use the namespace specifier explicitly e.g. MSXML2:: IXMLDOMAttributePtr See also "R400 Application development guidelines.doc" tips and tricks / migrating to MSXML 6

Standard template library (STL) related

Iterators

Error C2440: 'initializing' : cannot convert from 'int' to
'std::_Vector_iterator<_Ty,_Alloc>'
Example 1: vector<some type *>::iterator it=NULL;

Cause: Some iterators are no longer the same as pointers

FIX: Modify the code so that iterator is not handled as a pointer, e.g. by removing the "=NULL", vector<sometype *>::iterator it; Example2:

```
vector<CMyType> vec;
CMyType* pMyObj= find(vec.begin(),vec.end(),somevalue);   //error
//use this instead
vector<CMyType>::iterator it = find(vec.begin(),vec.end(),somevalue);
```

Checked_iterators

warning C4996: 'std::copy': Function call with parameters that may be unsafe...

Why this occurs: By default, VC++ uses checked_iterators. Those iterators can detect out of range situations in runtime. When such situation occurs, either an exception is thrown or runtime error occurs.

(jatkuu)

If e.g. the output iterator parameter in the copy statement is unchecked iterator (e.g. c++ array), this warning occurs. Fix options:

- Option 1: Define _SCL_SECURE_NO_WARNINGS (e.g. /D"_SCL_SECURE_NO_WARNINGS" in project options). With this option, checked_iterators are used whenever it is possible. However, no warnings are generated where compiler can't use checked_iterators.
- Option 2: Define _SECURE_SCL 0 (e.g. /D"_SECURE_SCL =0" in project options). If this option is used, then there isn't any runtime checking. This is less safe option.
- Option 3: Change the code so that checked_iterators can be used. This is the safest option, but may require more work.

Reverse_iterator Changes

The names for some of the types defined by the Standard C++ Library reverse_iterator class have changed. Also, there are different template arguments for this class. For more details, see MSDN Library "reverse_iterator changes".

Differences in iostream Implementation

The old iostream library was removed beginning in Visual C++ .NET 2003. The main difference between the Standard C++ Library and previous run-time libraries is in the iostream library. Details of the iostream implementation have changed, and it may be necessary to rewrite parts of your code that use iostream if you want to link with the Standard C++ Library. You will have to remove any old iostream headers (fstream.h, iomanip.h, ios.h, iostream.h, istream.h, ostream.h, streamb.h, and strstrea.h) you have included in your code and add one or more of the new Standard C++ iostream headers (<fstream>, <iomanip>, <ios>,

(jatkuu)

<iosfwd>, <iostream>, <istream>, <ostream>, <sstream>, <streambuf>, and <strstream>, all without the .h extension). For more details, see MSDN Library "Differences in iostream Implementation".

Windows API returning a string

If STL string class is used to store a string which some windows API function feturns, then the following doesn't compile:

```
int SomeAPIReturnningString(/*[out]*/char* pString,
                            /*[in, out]*/  int*StringSize);
string str(100,' ');
int iSize=100;
int i=SomeAPIReturnningString(str.begin(), \&iSize);
```

It causes error C2664: 'SomeAPIReturnningString' : cannot convert parameter 1 from 'std::_String_iterator<_Elem,_Traits,_Alloc>' to 'char *' This is because iterator != pointer, and on the other hand STL documentation says that the data in STL strings aren't guaranteed to be stored in contiguous memory. However, STL vector can be used in this case. Using a vector:

```
int iSize=100;
vector<char> vecBuf(iSize); //vector with max 100 items
int i=SomeAPIReturnningString(\&vecBuf[0],  \&iSize);
if (iSize <= 100)
  //construct the string from the vector
  string str(vecBuf.begin(),vecBufend());
else
  ... //error, probable the string doesn't
    //fit to the buffer (100 items).
```

(jatkuu)

Windows target version

Some Windows API functions may not be declared correctly if target is defined to be lower than 0x0600. As a result the code doesn't compile. It may cause strange compiler errors. If the C++ project header has: #define _WIN32_WINNT 0x0xyz , change it to 0x0600 to use Vista/WS2008 compatible headers. (If the code compiles OK, this definition may not be necessary.) This error can occur e.g. when you are generating proxy/stub dll with nmake /f <projectname>ps.mk. In that case, the error is coming from <projectname>ps.mk. on row: cl /c /Ox /DWIN32 /D_WIN32_WINNT=0x0400 /DREGISTER_PROXY_DLL

Registration in Post build scripts

There are several post build scripts to generate the tlb file, registering of assemblies and dlls in the existing projects. Such post build scripts will pollute the SCM build machines with lot of file registered. Such registration has to be added to the build system as command line calls. All post build scripts should be removed from the project files and SCM shall add command line calls to generate tlb files. This post build script removal is restricted only to registration of the files. The .NET projects will continue to have post build script to generate the tlb files from the assembly using tlbexp.exe

Post build step & Vista

If you are building a converted COM project in Windows Vista, you can get following error in the post-build step: Project : error PRJ0019: A tool returned an error code from "Performing registration" This happens because COM component registration requires administrative privileges, and VS2008 is running with non-admin privileges.

(jatkuu)

Incompatible warning levels

Warning C4652: compiler option 'Warning level (/W<n>)' inconsistent with precompiled header; current command-line option will override that defined in the precompiled header

Why this occurs: Some modules of the project are defined to compile with non-default warning level.

Fix: Change the warning level for each .cpp file of the project to <inherit from parent...>. (See Fig 1.)
Note: This must be done for each cpp file individually. Select cpp file from Solution Explorer, then right click/properties, which gives you the following configuration window.

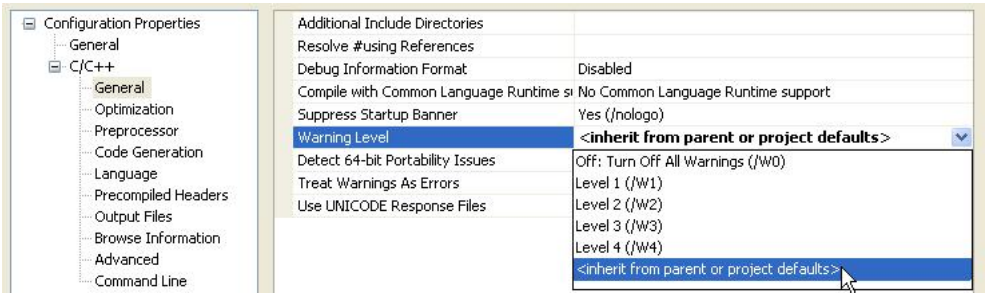


Fig 1: Fix the warning level

Other

If the working directory path contains space characters, e.g.
C:\Working directory\... , compiling idl files may fail. Error message is something like: midl : command line warning MIDL1009 : unknown argument ignored "...

Fix: Either use working directory where aren't any space characters, or modify the project idl compilation command line so that the $(InputPath) is surrounded with ". See Fig 2.

(jatkuu)

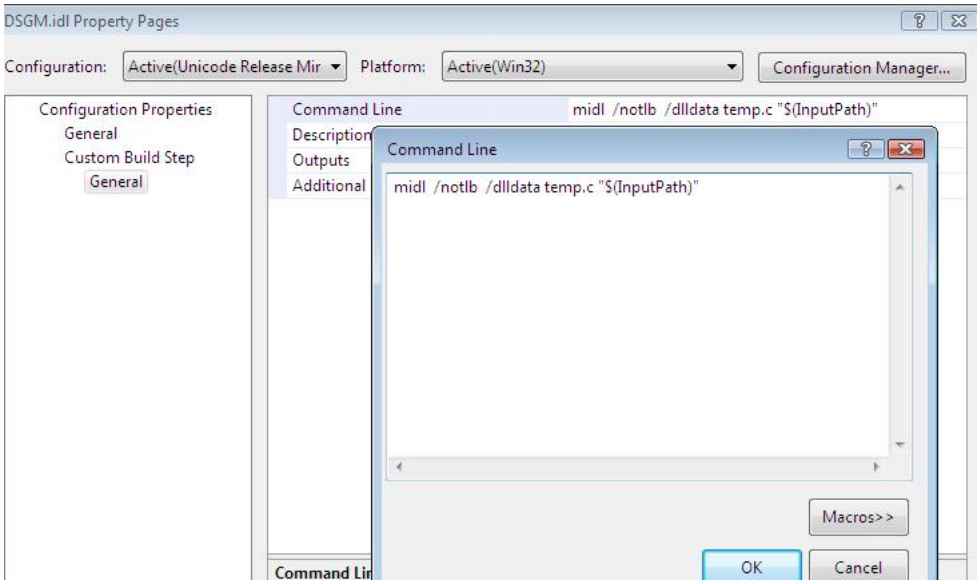


Fig 2: Adding extra “" to $(InputPath)

Also in custom build steps (mc compiler) there can be errors like: >mc : error : 0x2 trying to open file. You could try to change the $InputFile to $InputPath in the custom build step.

Linker errors

MAPINFO:LINES

Linking with option MAPINFO:LINES generates error in VS2008 Reason: Option isn’t supported in Visual studio 2008.

Fix: remove the option

COM Dlls

Link LNK4222 exported symbol ’symbol’ should not be assigned an ordinal.

(jatkuu)

Cause

The symbols DllCanUnloadNow, DllGetClassObject, DllGetClassFactoryFromClassString, DllInstall, DllRegisterServer, DllRegisterServerEx, DllUnregisterServer are always located by name using GetProcAddress. The linker warns about this kind of export because it could result in a larger image. This could happen if the range of ordinal exports is large with relatively few exports.

Fix: remove @<number> after function name from .def file, e.g.:
DllCanUnloadNow @1 PRIVATE -> DllCanUnloadNow PRIVATE

COM Exe server with proxy/stub

LINK : fatal error LNK1181: cannot open input file 'rpcndr.lib' This error can occur when the proxy/stub dll of a COM component is being generated in Post build step. The VC generated makefile, which is used for proxy/stub generation, contains rpcndr.lib. However, the Windows SDK no longer ships with rpcndr.lib.

Fix: Try linking against rpcrt4.lib, it replaces the functionality of rpcndr.lib. Open the makefile <projectname>ps.mk, and find the rpcndr.lib and replace it with rpcrt4.lib , save the makefile, and then rebuild the project.