

Thesis

Filipe Governa

Fighting Spyware
with
Mandatory Access Control
in
Windows Vista

*Fakultät Technik und Informatik
Department Informatik*

*Faculty of Engineering and Computer Science
Department of Computer Science*

Filipe Governa
Fighting Spyware
with
Mandatory Access Control
in
Windows Vista

This is submitted in the context of the examination
in the course European Computer Science
at the Department of Computer Science
of the Faculty of Engineering and Computer Science
of the Hamburg University of Applied Sciences

Supervising examiner: Dr. Martin Hübner
Secondary consultant: Dr. Wolfgang Gerken

Delivered on 14 July 2008

Filipe Governa

Title of the paper

Fighting Spyware with Mandatory Access Control in Windows Vista

Keywords

Access Control, Bell-LaPadula Model, Security, Spyware, Windows Vista.

Abstract

Inside this report, existing Mandatory Access Control (MAC) implementations are analyzed, with a strong focus on Microsoft Corporation's Mandatory Integrity Control (MIC) MAC implementation in the Windows Vista operating system (OS), and the development of an important security concept for efficiently fighting spyware in Windows Vista using the MAC access control security mechanism is described. The reasons behind the development of this important security concept are three-way: (i) the increased and growing seriousness of the threat posed by spyware today to computer user's privacy, (ii) the elevated and wide-spread usage of the Microsoft Windows OS, and (iii) the inefficiency and flawed nature of today's tools and techniques designed to fight ever-evolving spyware allied to the efficient and powerful possibilities provided by MAC in that regard. The problem addressed is spyware, namely (sensitive) information disclosure. It is addressed by tweaking MIC and taking advantage of it, more precisely by running Web browsers and e-mail clients with low rights, making sure that objects created by these applications have equally-low rights, and enabling a security policy in sensitive files that restricts *read* access to low-rights subjects, and then using MIC as the enforcement mechanism. The result is an important security concept, and it is concluded that, even though application compatibility and user experience is affected, it should be possible to use MAC to efficiently fight spyware in Windows Vista because spyware depends on the ability to read in order to collect (sensitive) data, and because the architecture that enables this (MIC) is already implemented in the OS (Windows Vista). One significant implication of the developed concept is the ability that it provides to actively, efficiently, and transparently (to the user) fight the serious and growing threat of spyware in the most used OS platform in the world.

TABLE OF CONTENTS

List of Figures	ii
List of Tables	iii
Acknowledgements	iv
Chapter I: Introduction	1
1.1. Problem Definition	1
1.2. Motivation	3
1.3. Aim of Study	4
1.4. Structure of the Work	5
Chapter II: Background Concepts	7
2.1. Mandatory Access Control	7
2.1.1. Applications	9
2.1.2. Types and Models	10
2.1.2.1. Multi-Level Security	10
2.1.2.1.A. Bell-LaPadula Model	11
2.1.2.1.B. Biba Model	12
2.1.2.1.C. Low-Watermark MAC Model	12
2.1.2.2. Multi-Lateral Security	13
2.1.2.2.A. Lattice Model	14
2.1.2.2.B. Brewer-Nash Model	14
2.1.2.3. Need-to-Know Principle	15
2.1.2.4. Other Security Models	15
2.1.2.4.A. Clark Wilson Model	15
2.1.2.4.B. British Medical Association Model	16
2.1.3. Disadvantages	16
Chapter III: Requirement Analysis	17
3.1. To Be Transparent to the User	17
3.2. To Provide Active Protection	17
3.3. To Be Easily Implementable in Windows Vista	17
3.4. To Disable Spyware	17
3.5. Additional Important Considerations	18
3.5.1. MAC Scope	18
3.5.2. Implications of the Term “Mandatory”	18
Chapter IV: Market Review	20
4.1. MAC Implementations Overview	20
4.2. Security-Enhanced Linux	21
4.3. AppArmor	21
4.4. FreeBSD	21
4.5. Trusted Solaris	22

4.6.	Mac OS X	22
4.7.	Mandatory Integrity Control	23
4.7.1.	Concept	23
4.7.1.1.	Extending the Windows Security Architecture	24
4.7.1.2.	Features	24
4.7.1.3.	Purpose	26
4.7.1.4.	Earlier Integrity Models	27
4.7.2.	Design	29
4.7.2.1.	Integrity Levels	30
4.7.2.2.	Integrity Policies	32
4.7.2.2.A.	Mandatory Label Policies	33
4.7.2.3.	Mandatory Label ACE	35
4.7.2.4.	How Windows Assigns a Mandatory Label to Objects	37
4.7.2.4.A.	Default IL	39
4.7.2.4.B.	Labeling Objects Created by <i>Low</i> Subjects	41
4.7.2.4.C.	Creating an Object with a Specific Mandatory Label	42
4.7.2.5.	Mandatory Label Inheritance	43
4.7.2.5.A.	Inheritance and Explicit Labels	46
4.7.2.5.B.	Inherit-Only Restriction	47
4.7.2.5.C.	Protected SACL and Label Inheritance	47
4.7.2.6.	How Access Checks Work with Mandatory Policy	48
4.7.3.	Implementation	49
4.7.3.1.	IEMP at <i>Low</i> Integrity	50
4.8.	Conclusion	53
Chapter V: System Design & Implementation		54
5.1.	User Profile Folder with <i>NO READ UP</i> Mandatory Label Policy	55
5.2.	Web Browsers and E-Mail Clients at <i>Low</i> Integrity	57
5.3.	Objects Created by Web Browsers and E-Mail Clients Are Assigned the <i>Low</i> IL	58
5.4.	Scenario Solution	60
5.5.	Concept Applicability Motivation	61
Chapter VI: Summary and Outlook		63
Glossary		65
References		74
Bibliography		78
Appendix A: SDDL for Mandatory Labels		82
Appendix B: Icacls and File IL's		84
Appendix C: Chml and File IL's		85
Appendix D: MIC and Windows Kernel Mode Code Integrity		91
Appendix E: Getting the IL for an Access Token		92

LIST OF FIGURES

<i>Number</i>		<i>Page</i>
1.	Mark Minasi	1
2.	Security levels in MLS	11
3.	Security levels in Multi-Lateral Security	13
4.	Security descriptor fields	35
5.	Inheritable <i>low</i> mandatory label	45
6.	Inheriting a mandatory label on a child object	46
7.	IEPM processes	52
8.	Icacs and mandatory labels	84

LIST OF TABLES

<i>Number</i>		<i>Page</i>
1.	MAC implementations	20
2.	IL SID identifier authority values	30
3.	Defined IL's and corresponding values	31
4.	IL string names	31
5.	Integrity policy categories	33
6.	Mandatory label policies in Windows Vista	34
7.	Fields contained in the ACE header	36

ACKNOWLEDGEMENTS

The author wishes to express sincere appreciation to Dr. Martin Hübner for his assistance in the preparation of this manuscript. In addition, special thanks to Dr. Wolfgang Gerken whose support was helpful during the early programming phase of this undertaking. Thanks also to all the members of Instituto Superior de Engenharia de Coimbra (ISEC) / Higher Institute of Engineering of Coimbra (PORTUGAL), University of Huddersfield (UK), and Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg) / Hamburg University of Applied Sciences (GERMANY) that I had the pleasure to meet for directly and/or indirectly helping me get to this final stage of the course.

A word of gratitude is also due to my parents Helena and Carlos, my brother Ricardo, and my girlfriend Anna, for all their continuous support to reach this stage.

Chapter 1

INTRODUCTION

This [“no-read-up” policy] seemed like a potentially nice feature in a world beset by Web-borne spyware!

Figure 1 Mark Minasi

(www.minasi.com)

Author of

ADMINISTERING WINDOWS VISTA SECURITY:

THE BIG SURPRISES

in www.minasi.com/vista/chml.htm



Mark Minasi is a best-selling author, popular technology columnist, commentator, keynote speaker, and one of the world’s leading Microsoft Windows authorities. This statement of his was in the basis of this thesis.

1.1. Problem Definition

It is widely agreed among computer science experts that spyware poses a dangerous threat to computer users’ privacy. The Anti-Spyware Coalition (ASC), a group dedicated to building a consensus about definitions and best practices in the debate surrounding spyware and other potentially unwanted technologies, calls it *one of the most dire threats facing the Internet*,^[1] Webroot Software, makers of popular anti-spyware program Spy Sweeper, calls it *an extremely dangerous online threat that jeopardizes your computer’s data and your personal information*^[2]. The following facts support this idea:

- in an estimate based on customer-sent scan logs, Webroot Software said that 9 out of 10 personal computers (PCs) connected to the Internet are infected with spyware;^[2]
- according to the same company, 88% of Spy Audit (Spy Sweeper tool) scans have found some form of unwanted program (Trojan, system monitor, cookie or adware) on consumer computers;^[2]
- also according to the same company, some form of spyware can be found on 87% of corporate PCs;^[2]
- according to a 2005 study by America Online LLC. (AOL) and the National Cyber-Security Alliance, 61% of surveyed users' computers had some form of spyware, 92% of surveyed users with spyware reported that they did not know of its presence, and 91% reported that they had not given permission for the installation of the spyware;^[3]
- according to a September 2006 study by Consumer Reports, United States (US) consumers spent \$7.8 billion in 2004 and 2005 for computer repairs, parts, and replacements because of malware attacks;^[4]
- according to a September 2007 study by the same company, the chances of getting a spyware infection are 1 in 3;^[5]
- according to the same study, the chances of suffering serious damage due to a spyware infection are 1 in 11.^[5]

At the same time, it is a fact that the Microsoft Windows operating system (OS) from Microsoft Corporation is the most used OS in the world. According to data

from Awio^[6], Hitslink^[7], OneStat^[8], and XiTi^[9] – and as of March 2008¹ –, Microsoft Windows (all versions) accounts for between 91.57% (Hitslink) and 95.94% (OneStat) of the OS's market. As of the elaboration of this thesis, Windows Vista is the most recent commercial release of Microsoft Windows.

It is also widely agreed among computer science experts that Mandatory Access Control (MAC) is an important access control security model and provides valuable security, namely in the field of data authorization (data confidentiality and data integrity).^[10] It is defined by the Trusted Computer System Evaluation Criteria (TCSEC)^[11], and is most commonly applicable to Classified National Security Information.

But it is still somewhat unexplored, unclear, and undocumented how MAC can be used to efficiently fight spyware in Windows Vista.

1.2. Motivation

The threat of spyware has grown. Consumer Reports recognizes that the chances of getting a spyware infection and of suffering serious damage due to one have not decreased;^[5] the ASC goes further in its acknowledgement and recognizes that *the threat has grown and evolved from an online nuisance to one of the most dire threats facing the Internet*,^[1] Webroot Software, on its side, says that *spyware today is more malicious and powerful than ever before*.^[12] The following facts support this idea:

- in 2006 alone, Phileas (Webroot's automated threat research system) found 3 million malicious websites, a 250% increase from 1.2 million malicious websites found in 2005;^[13]

¹ The data from XiTi exceptionally reports to December 2007.

- \$198.44 million were lost in 2006 due to cyber fraud with a median dollar loss of \$724.00/complaint, which was up from \$183.12 million in total reported losses in 2005.^[13]

Microsoft Windows' market share has not decreased significantly in the past. According to data from Awio^[6], Hitslink^[14], OneStat^[8], and XiTi^[9] – and in the period between May 2007 and March 2008 –, Microsoft Windows (all versions) has only had a slight fluctuation of between -0.53% (Awio) and -2.54% (XiTi). At the same time – and as of the elaboration of this thesis –, Windows Vista is the most recent commercial release of Microsoft Windows, and its market share has been growing significantly. OneStat goes as far as calling it *a remarkable fact*.^[8] According to data from the same companies – and in the same period of time – Windows Vista's OS's market share has had a fluctuation of between +5.02% (Awio) and +10.27% (Hitslink^[15]).

MAC provides an exciting, promising, and unexplored way of efficiently fighting this growing threat (spyware) in the most used OS in the world (Microsoft Windows).

1.3. Aim of Study

The main objective of this thesis is to generate a concept for efficiently fighting spyware in Windows Vista using MAC.

Considering an explicit example scenario: if a Windows Vista user receives an attachment in an e-mail, saves it, and then executes it, its (the attachment's) process will be able to read the user's data.

The concept generated in this thesis works as a solution for this explicit example scenario, preventing this from happening (the attachment's process being able to read the user's data) on a system running Windows Vista.

1.4. Structure of the Work

This thesis details (i) a concept for efficiently fighting spyware in Windows Vista using MAC, (ii) why it was designed, and (iii) how it works. In this thesis the following topics are discussed:

- [Chapter I: Introduction](#)
Discusses the problem definition, motivation, aim of study, and structure of the work.
- [Chapter II: Background Concepts](#)
Presents, describes, and reviews (in detail) the concept of Mandatory Access Control (MAC).
- [Chapter III: Requirement Analysis](#)
Discusses the requirements of a solution.
- [Chapter IV: Market Review](#)
Shows an overview of past work by analyzing existing MAC implementations, with a strong focus on Microsoft Corporation's Mandatory Integrity Control (MIC) MAC implementation in the Windows Vista OS.
- [Chapter V: System Design & Implementation](#)
Shows the conception of an own solution (with a detail description), and how it is designed to make use of MIC (i.e. how MIC in Windows Vista can be tweaked in order to enable it).
- [Chapter VI: Summary and Outlook](#)
Discusses the state of the project and advancement possibilities.

Summarizing this introduction, the topic of this thesis is how to efficiently fight spyware in Windows Vista using MAC, and it is important because, as we have seen, spyware, today more than ever, poses a dangerous threat to computer users' privacy.

The author thinks and will show that, even though application compatibility and user experience is affected, it should be possible to use MAC to efficiently fight spyware in Windows Vista because spyware depends on the ability to read in order to collect (sensitive) data, and because the architecture that enables this (MIC) is already implemented in this OS.

BACKGROUND CONCEPTS

2.1. Mandatory Access Control

Security is an important issue in information systems. A security model, as an important way to protect the privacy and integrity of messages from being spied by illegal users, is a formal method to verify and describe a complex information system. It is widely used not only in the abstract definition of system security requirements, but also in system design and implementation.

In International Organization for Standardization (ISO) standard 7498-2, five security objectives are identified: authentication, authorization, integrity, confidentiality, and non-repudiation. An important security mechanism associated with authorization and confidentiality issue is access control. Access control is here defined as a service which is to limit actions that an active system component (process, agent, user, etc.) can perform. The word “access” means in this context *an interaction between system components that results in the flow of information.*^[16] There are two basic forms of access control: Mandatory Access Control (MAC) and Discretionary Access Control (DAC). Traditionally, one of these two different security models can be identified depending on who is in charge of setting security rules.

From the view of security, there are two main components / types of entities in a realistic computer system. One is *object* or *target*, mainly including all passive entities (constructs), such as (usually and in practice) files, directories, storage areas, data structures, Transmission Control Protocol (TCP) / User Datagram Protocol (UDP) ports, network packets, or shared memory segments. The other is *subject* or *initiator*, mainly containing all active entities, such as (usually and in

practice) users, processes (possibly executed on behalf of a user), or threads. The object is the carrier and channel of information resource, i.e. it contains or receives information. The subject obtains information by accessing the object. To precisely describe access control issues, it is assumed that all of a system's components fall into one of these two categories.

Mandatory Access Control (MAC) refers to a type of access control by which an OS constrains the ability of a *subject* to access or generally perform some sort of operation on an *object*. Whenever a subject attempts to access an object, an authorization rule enforced by the OS kernel examines these security attributes, and decides whether the access can take place. Any operation by any subject on any object will be tested against the set of authorization rules (aka *policy*) to determine if the operation is allowed.

With MAC, this security policy is centrally controlled by a security policy administrator; users do not have the ability to override the policy and, for example, grant access to files that would otherwise be restricted. By contrast, Discretionary Access Control (DAC), which also governs the ability of subjects to access objects, allows users the ability to make policy decisions and/or assign security attributes.² MAC-enabled systems allow policy administrators to implement organization-wide security policies. Unlike with DAC, users cannot override or modify this policy, either accidentally or intentionally. This allows security administrators to define a central policy that is guaranteed (in principle) to be enforced for all users.

Historically and traditionally, MAC has been closely associated with Multi-Level Security (MLS) systems. In the seminal work on the subject which is often

² The traditional Microsoft Windows and UNIX systems of users, groups, and *read*, *write*, and *execute* permissions are examples of DAC.

referred to as “Orange Book”, the TCSEC^[11] defines MAC as *a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity*. Early implementations of MAC such as HPUX BLS, Harris CS/SX, and SGI Trusted IRIX were all focused on MLS.

More recently, with the advent of implementations such as SELinux (incorporated into Linux kernels after 2.6), MAC has started to become more mainstream, and is evolving out of the MLS niche. These more recent MAC implementations have recognized that the narrow Orange Book definition, focused as it was on MLS, is not sufficient for general use.^[17] These implementations provide more depth and flexibility than earlier MLS-focused implementations,^[18] allowing (for example) administrators to focus on issues such as network attacks and malware without the rigor or constraints of MLS systems.

2.1.1. Applications

MAC models serve to guarantee the security of information against unauthorized access, and also to enforce system requirements. The protection of information relates to:

- *confidentiality* – ensuring that information can be read only by those authorized to read it (e.g. any piece of information subject to secrecy);
- *integrity* – ensuring that information can be manipulated only by those authorized to manipulate it (e.g. the chain of command of a military employment system such as the Command-and-Control systems).

A characteristic is the implementation of access controls in IT systems. Furthermore, the security models can also be similarly deployed in organization forms, processes, and building services engineering.

Such access systems are particularly needed in the area of the military, where it concerns sensitive information about warfare, but also in the area of authorities, where it concerns information about technology, politics, foreign trade, and communications-electronics (C-E).³ Another area of application is patient data in the health industry (e.g. patient cards).

2.1.2. *Types and Models*

There are three main types of MAC (each of which may enclose different MAC models): Multi-Level Security (MLS), Multi-Lateral Security, and Need-to-Know Principle.

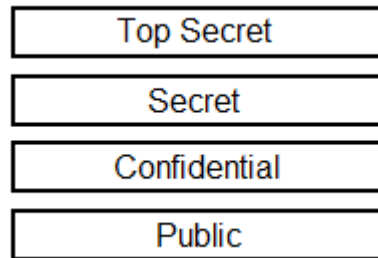
2.1.2.1. MULTI-LEVEL SECURITY

This is the simplest (and, to some extent, also historical) type of MAC. MLS systems establish the model of security levels. Each object is assigned a security level (classification [CLS]), which typically represents the level of sensitivity of the information that it contains. The individual security levels divide the objects into “layers” (vertical arrangement) (Figure 2). The term “vertical” (and, consequently, the access security) refers to the top-down and bottom-up information flow, and means that information may only flow without restrictions within layers. Classified information may not become public. Every subject is, likewise, assigned a security level (clearance [CLR]), which typically represents its trustworthiness. A subject may, then, access an object of another layer only if the security level of the subject (e.g. the CLR of a person) is at least as high as the security level of the object (e.g. the CLS of a document). As such, with MLS systems, the access is always decided on the basis of security levels.

³ See also on this “classified information” (Glossary).

MLS systems correspond to the original MAC form, which was specified in the 1970's. Implementations were mostly used on mainframes in military or other security-related areas. Until today, this kind of MAC is the furthest common.

Figure 2 Security levels in MLS



2.1.2.1.A) BELL-LAPADULA MODEL

The Bell-LaPadula Model was developed by David Elliott Bell and Len LaPadula – subsequent to strong guidance from then-CAPT Roger R. Schell, Ph.D. (USAR, Ret.) – in 1973^{[19][20][21]} to formalize the US Department of Defense (DoD) MLS policy. It addresses data confidentiality. In the Bell-LaPadula Model, information is protected against reading by unauthorized subjects. A subject can read an object only if the security level of the subject (CLR) dominates (i.e. is greater than or equal to) the security level of the object (CLS), i.e., only if $CLR \geq CLS$; additionally, a subject can manipulate an object only if the security level of the subject (CLR) is dominated by (i.e. is less than or equal to) the security level of the object (CLS), i.e., only if $CLR \leq CLS$. As such, a subject at a given security level cannot read objects of a higher security level, nor manipulate objects of a lower security level (“no read-up”, “no write-down”). This results in a bottom-up information flow (i.e. from bottom to top). There can, although, be a distinction between *trusted* and *untrusted* subjects.^[22] Trusted subjects can be relied on not to compromise security; all other subjects are untrusted. As such, the manipulation on an object at a given security level by a subject at a higher security level may

happen in the Bell-LaPadula Model via the concept of trusted subjects. Trusted subjects are not restricted by the “no-write down” security policy. Untrusted subjects are. Trusted subjects must be shown to be trustworthy with regard to the security policy. Systems based on the Bell-LaPadula Model were mainly used if data was subject to a certain secrecy. The classical Bell-LaPadula systems were replaced by Lattice or Compartment-Based systems.

2.1.2.1.B) BIBA MODEL

Also known as Biba Integrity Model, the Biba Model was developed by Kenneth J. Biba in 1977^[23]. It addresses data integrity. In the Biba Model, information is protected against manipulation by unauthorized subjects. A subject can manipulate an object only if the security level of the subject (CLR) dominates (i.e. is greater than or equal to) the security level of the object (CLS), i.e., only if $CLR \geq CLS$; additionally, a subject can read an object only if the security level of the subject (CLR) is dominated by (i.e. is less than or equal to) the security level of the object (CLS), i.e., only if $CLR \leq CLS$. As such, a subject at a given security level cannot manipulate objects of a higher security level, nor read objects of a lower security level (no write-up, no read-down). This results in a top-down information flow (i.e. from top to bottom). The Biba Model is used, for example, in Information Technology (IT) (e.g. as counter measure against attacks on security-relevant systems such as firewalls) and in military systems (where it is fundamentally important that an instruction in the command chain cannot be modified and a wrong instruction is, thus, passed on).

2.1.2.1.C) LOW-WATERMARK MAC MODEL

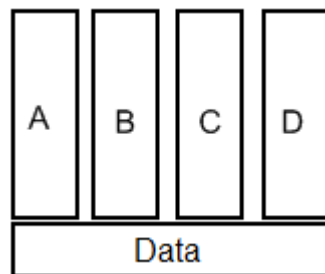
The Low-Watermark MAC (LoMAC) Model is a variation of the Biba Model that allows subjects at a given security level to read objects of a lower security level (no write-up, read all). When this happens, the security level of the reading subject is reduced to the security level of the object read, so that the subject can

no more manipulate objects of its original security level. This results in a bottom-up / top-down information flow (i.e. from bottom to top, and from top to bottom). LoMAC systems are mainly implemented in *cbroot* applications such as *honeypots*.

2.1.2.2. MULTI-LATERAL SECURITY

This is a more complex type of MAC. Multi-Lateral Security systems employ not only a vertical top-down or bottom-up arrangement consisting of security levels (like MLS), but also a horizontal arrangement consisting of labels, thus forming a lattice (i.e. they also assign access rights based on segments). Technically, both security levels and labels are used. This results in a horizontal access system (the labels) that features additional vertical characteristics (the security levels) (Figure 3). An access to an object is only possible with not only the appropriate security level (e.g. *secret*), but also with the appropriate label (e.g. *crypto*), thus properly fulfilling all the requirements. If subject A has read access on the security level *strictly confidential*, it can read objects of this security level. The same user has, however, no access to objects that are classified as *strictly confidential (crypto)*. In order to clarify more complex circumstances, these systems are also designated as Policy-Based Security systems or Rule-Based Security systems.

Figure 3 Security levels in Multi-Lateral Security



2.1.2.2.A) LATTICE MODEL

Also known as Compartment Model, the Lattice Model was specified in 1976 by Dorothy E. Denning^[24], and in 1993 by Ravi S. Shandru^[25]. It is based on the Bell LaPadula Model, and extends the accesses around labels, thus forming a lattice. If subject A has read access to the security levels *strictly confidential* and *confidential*, it can read objects of these security levels. The same subject has, however, no access to objects that are classified as *strictly confidential (crypto)*. Only if the subject has access to the CLS's *strictly confidential* (security level) and *crypto* (label) can it access these objects. In principle, the model applies a combination of security levels with the Need-to-Know Principle: objects become both vertically (according to security level) and horizontally (according to subject matter) subdivided, while subjects are assigned a security level per subject matter. An access can take place only if the requirements of both control systems are fulfilled. Special attention is put on the control of the information flow. It should not be possible confidential information to be passed on to untrustworthy subjects.

2.1.2.2.B) BREWER-NASH MODEL

This model is also known as Chinese Wall Model (where the expression “Chinese wall” comes from the financial sector), and describes certain rules that are to prevent a clash of interests to be caused. It is a variation of the Lattice Model, and was described in 1989 by David F.C. Brewer and Michael J. Nash.^[26] This model was developed to provide information security access controls that can change dynamically. It was designed to provide controls that mitigate conflict of interest in commercial organizations. As such, it builds separation of duties within the access control. This model is also built upon an information flow model. The individual data is ordinarily arranged into horizontal segments, and mutual access is barred. In this model, no information can flow between subjects and objects in a way that would create a conflict of interest.

2.1.2.3. NEED-TO-KNOW PRINCIPLE

The Need-to-Know Principle offers an alternative to the security levels model: here, objects are arranged horizontally into subject matter; each subject is assigned the subject matter for which they should be responsible. According to the specification, a subject that wants to access an object must now belong either to all or at least one of the subject matters that are assigned to the object. Thus, the spreading range of information is limited substantially, and the control of the information flow is facilitated. The advantage of this security concept consists on the fact that individual subjects are only granted the rights needed for their task. Thereby, the risk of a misuse of applications by exploitation of security vulnerabilities is minimized. That means, for example, that an application that does not need authorization for network access receives no rights for this purpose. This implicates that an attacker that would like to exploit a security vulnerability cannot misuse a program in order to establish network connections.

2.1.2.4. OTHER SECURITY MODELS

2.1.2.4.A) CLARK-WILSON MODEL

The Clark-Wilson Model was developed in 1987 by David D. Clark and David R. Wilson.^[27] It describes the integrity of commercial, non-military systems, and is a variation of the classical MAC approach. Practically every mainframe processes data on basis of the Clark Wilson Model.

1. The system is in a valid (consistent) initial state.
2. Access to the system only by means of explicitly permitted transactions.
3. Only transactions that bring the system under all circumstances into a (new) valid state are permitted.

2.1.2.4.B) BRITISH MEDICAL ASSOCIATION MODEL

The British Medical Association (BMA) Model was specified in 1996 by Ross Anderson.^[28] It combines characteristics of the Clark Wilson Model with the Bell LaPadula Model. The BMA Model is an access model that was developed for the protection of medical data. It is generally applicable to all data that is subordinate to data protection. In 1996, it was taken over by the UEMO European Medical Organization. The BMA Model is not centrally, but rather locally applied. The policy is determined by the patient.

2.1.3. Disadvantages

The main disadvantage of MAC lies in the complexity of the configuration, since, for each application, it must be determined which access authorizations the application requires.

Chapter 3

REQUIREMENT ANALYSIS

A solution should meet the following requirements:

- to be transparent to the user;
- to provide active protection;
- to be easily implementable in Windows Vista;
- most importantly, to disable spyware.

3.1. To Be Transparent to the User

There should be no requirement for administrator or user configuration and/or interaction for the solution to work correctly.

3.2. To Provide Active Protection

A solution should provide constant “real-time” protection against spyware, as opposed to passive “on-demand” protection.

3.3. To Be Easily Implementable in Windows Vista

A solution should be able to be implemented in the Windows Vista OS from Microsoft Corporation without requiring changing a lot of code.

3.4. To Disable Spyware

There is no true consensus on the definition of “spyware” (notwithstanding the valuable efforts of groups like the ASC). Additionally, there are also several types of spyware. It should, then, be noted that, in the context of this thesis, “spyware”

is any piece of software that (attempts to) collect(s) sensitive information without appropriate user consent. Technically speaking, this translates to any piece of software that (attempts to) read(s) an object containing sensitive information without user interaction. In the context of this thesis, sensitive information is any piece of personal, private, confidential, secret, or top secret data. In the context of this thesis, sensitive information translates to user data files. In the context of this thesis and Windows Vista, user data files are files located under a user profile of Windows Vista (%USERPROFILE%). Technologies such as advertising display software, remote control software, dialing software, system modifying software, security analysis software, and automatic download software are, thus, out of the scope of this thesis. Passive tracking technologies are also out of the scope of this thesis. As far as spyware is concerned, this thesis aims primarily at tracking software. Concluding, a solution should prevent sensitive information disclosure.

3.5. Additional Important Considerations

3.5.1. MAC Scope

In this thesis, MAC is used to refer to the approach where a system wide security policy restricts the access rights of subjects (usually processes). This is a wider interpretation of MAC than that in the TCSEC^[11], which focuses on MLS.

3.5.2. Implications of the Term “Mandatory”

It is defended by some that the term “mandatory” used with access controls has historically implied a very high degree of robustness that assures that the control/enforcement mechanism(s) resist(s) subversion, thereby enabling it/them to enforce an access control policy that is mandated by some regulation that must be absolutely enforced, such as the Executive Order 12958 for US classified information.

The author disagrees with this assertion. In the author's opinion, the term "mandatory" used with access controls does not imply robustness, but only that the access controls are always (when possible) checked. As such, in the author's opinion, if a system evaluates any securable object⁴ against access controls, and includes (a) security mechanism(s) that attempt(s) to restrict changes of security configuration by users other than administrators, then the system *is* a MAC system. If those mechanisms fail, then the system may be considered an insecure MAC system.

Concluding, in the author's opinion, and in the context of this thesis, MAC does *not* imply robustness.

⁴ In the context of Windows Vista, a securable object is any object with a security descriptor (i.e. files, folders, pipes, processes, threads, window stations, registry keys, services, printers, shares, inter-process objects, jobs, and directory objects).

Chapter 4

MARKET REVIEW

4.1. MAC Implementations Overview

A few MAC implementations, such as Unisys' BLACKER project, were certified robust enough to separate TS from U late in the last millennium. Their underlying technology became obsolete, and they were not refreshed. Today there are no current implementations certified by TCSEC to that level of robust implementation. However, some less robust products exist (Table 7).

Table 1 MAC implementations

Producer (Implementation)	Model Type(s)	System(s)	Certification(s)
NSA / Red Hat (SELinux)	Variant of Bell-LaPadula	RHEL, Gentoo Linux, openSUSE, Debian	LSP, RBACPP, EAL4+
Novell (AppArmor)	-	openSUSE	-
TrustedBSD	Biba, LoMAC	FreeBSD	-
Sun	Variant of Bell-LaPadula	Trusted Solaris	-
Apple	Biba, LoMAC	Mac OS X	-
Microsoft (MIC)	Variant of Biba	Windows Vista	-
Rule Set Based Access Control	Variant of Bell-LaPadula	Gentoo Linux, Debian, Fedora	-
Unisys	Biba, Bell-LaPadula, Lattice, Clark-Wilson	OS2200	TCSEC B1
Argus Systems Group (PitBull LX)	Lattice	AIX, Solaris, Linux	ITSEC F-B1, E3

4.2. Security-Enhanced Linux

Security-Enhanced Linux (SELinux) is a reference implementation of the Flux Advanced Security Kernel (FLASK) security architecture for flexible MAC. It was created in order to demonstrate the value of flexible MACs and how such controls could be added to an OS. The FLASK architecture has been subsequently mainstreamed into Linux, having been merged into the mainline version of Linux in August 2003. It utilizes a Linux 2.6 kernel feature called Linux Security Modules (LSM), which provides a kernel API that allows modules of kernel code to govern access control. The FLASK architecture has been ported to several other systems, including the Solaris OS, the FreeBSD OS, and the Darwin kernel, spawning a wide range of related work. The FLASK architecture provides general support for the enforcement of many kinds of MAC security policies, including those based on the concepts of Type Enforcement (TE), Role-Based Access Control (RBAC), and MLS. The germinal concepts underlying SELinux can be traced to several earlier projects by the US National Security Agency / Central Security Service (NSA/CSS). Red Hat Enterprise Linux (RHEL) version 4 (and later versions) come with an SELinux-enabled kernel. Although SELinux is capable of restricting all processes in the system, the default targeted policy in RHEL confines the most vulnerable programs from the unconfined domain in which all other programs run. RHEL 5 ships two other binary policy types: *strict* (which attempts to implement least privilege) and MLS (which is based on *strict*, and adds MLS labels). RHEL 5 contains additional MLS enhancements, and received 2 Labeled Security Protection Profile (LSPP) / Role-Based Access Control Protection Profile (RBACPP) / Controlled Access Protection Profile (CAPP) / Evaluation Assurance Level (EAL) 4+ certifications in June 2007. ^[20]

4.3. AppArmor

AppArmor is security software for Linux, released under the GNU General Public License, that supplements the traditional UNIX DAC model by providing MAC. SUSE Linux (now supported by Novell) and Ubuntu 7.10 have added AppArmor. AppArmor utilizes LSM. It is not capable of restricting all programs, and is not yet included in the kernel.org kernel source tree. In most Linux distributions, MAC is not installed.

4.4. FreeBSD

FreeBSD is a UNIX-like free OS descended from AT&T UNIX via the Berkeley Software Distribution (BSD) branch through the 386BSD and 4.4BSD OSs. Beginning with version 5.0, the work of the TrustedBSD project has been incorporated into releases of the FreeBSD OS. Development is a work in progress, and the implementation models, as well as the capabilities, are constantly improving. MAC on FreeBSD comes with pre-built structures for implementing MAC models such as Biba and MLS.

4.5. Trusted Solaris

Trusted Solaris is a security-evaluated OS based on Solaris by Sun, featuring a MAC model. It uses a mandatory and system-enforced access control mechanism, where CLR's and labels are used to enforce a security policy. However, note that the capability to manage labels does not imply the kernel strength to operate in MLS mode. Access to the labels and control mechanisms are not robustly protected from corruption in protected domain maintained by a kernel. The applications a user runs are combined with the security label at which the user works in the session. Access to information, programs, and devices is only weakly controlled.

4.6. Mac OS X

Mac OS X is a line of graphical OS's developed, marketed, and sold by Apple, the latest of which is pre-loaded on all currently shipping Macintosh computers. Its MAC framework is an implementation of the TrustedBSD MAC framework.^[30] A limited high-level sandboxing interface is provided by the command-line function `sandbox_init`.⁵

4.7. Mandatory Integrity Control

4.7.1. Concept

Mandatory Integrity Control (MIC) is a core component of the Windows security architecture that restricts the access permissions of applications that are running under the same user account and that are less trustworthy.

MIC extends the security architecture of the OS by assigning an integrity level (IL) to application processes and securable objects.

The IL is a representation of the trustworthiness of running application processes and objects, such as files created by the application. MIC provides the ability for resource managers, such as the file system, to use pre-defined policies that block processes of lower integrity, or lower trustworthiness, from reading or modifying objects of higher integrity. MIC allows the Windows security model to enforce new access control restrictions that cannot be defined by granting user or group permissions in Access Control Lists (ACL's).

The Windows security architecture is based primarily on granting access rights (*read*, *write*, and *execute* permissions) and privileges to users or groups that are represented internally by security identifiers (SID's). When a user logs on to Windows, the security subsystem – the Security Reference Monitor (SRM) – sets

⁵ See *sandbox_init* manual page for documentation.^[5]

the user's SID and group membership SID's in a security access token. The security access token is assigned to every application process that is run by that user. Every time the application process opens an object, such as a file or registry key, the resource manager that manages the object calls on the SRM to make an access decision. The access check determines the allowed access permissions for this user. The SRM compares the user and group SID's in the access token with the access rights in a security descriptor that is associated with the object. If the user SID is granted full access rights in the object's ACL, then the application process that user runs has full access to the object.

4.7.1.1. EXTENDING THE WINDOWS SECURITY ARCHITECTURE

MIC extends the security architecture by defining a new Access Control Entry (ACE) type to represent an IL in an object's security descriptor. The new ACE represents the object IL. An IL is also assigned to the security access token when the access token is initialized. The IL in the access token represents a subject IL. The IL in the access token is compared against the IL in the security descriptor when the SRM performs an access check. Windows Vista uses the *AccessCheck* function to determine what access rights are allowed to a securable object. Windows restricts the allowed access rights depending on whether the subject's IL is higher or lower than the object, and depending on the integrity policy flags in the new access control ACE. The SRM implements the IL as a mandatory label to distinguish it from the discretionary access under user control that ACL's provide.

4.7.1.2. FEATURES

MIC enables a number of important scenarios in Windows Vista. MIC's main features are:

- IL's are assigned automatically to every security access token during access token creation, which means that every process and thread has an effective IL for access control;
- the SRM automatically assigns mandatory labels to specific object types;
- the system uses few IL's, which makes the basic architecture simple to understand and use;
- integrity policy is flexible, which meets the access requirements of different object resource managers, and allows for extensibility;
- it integrates with existing security architecture, which minimizes impact to the large legacy of system and application code that depends on Windows security;
- there is no requirement for administrators or users to configure IL's for the enforcement mechanism to work correctly.

MIC accomplishes this by defining a new mandatory label ACE type for assigning an IL to objects. Details of this structure are described in a later section of this thesis. However, the mandatory label ACE defines an object IL without changes to the existing security descriptor data structure definition or to the commonly used Discretionary Access Control List (DACL).

MIC is based on a mandatory label that the OS assigns in order to differentiate it from discretionary access under user control. DAC allows the object owner, or the group that is granted permission, to change the object's access permissions. Windows provides a graphical user interface (GUI) for advanced users to view and modify the security permissions (represented by the DACL) on objects, such

as files and registry keys. Mandatory labels are always assigned to specific objects, and there are controls on how the object creator can set or initialize the label on object creation. No GUI for managing integrity labels is implemented for Windows Vista, because label management is available or necessary for relatively few areas.

4.7.1.3. PURPOSE

The purpose of MIC is to restrict the access permissions of applications that are running under the same user account and that are less trustworthy. Unknown, potentially malicious code that is downloaded from the Internet should be prevented from modifying system state, changing user data files, or manipulating the behavior of other application programs. The Windows SRM assigns a simple hierarchy of IL's to code running at different privilege levels for the same user. Previous versions of Windows can adjust the security access token privileges of an application process, although such adjustment is not common. Before Windows Vista, most applications ran using an administrative account with full administrator rights. Windows Vista incorporates the concept of least privilege by enabling broader use of standard user accounts. User Account Control (UAC) in Admin Approval Mode for administrator accounts means that multiple applications on the same desktop are running with different privilege levels. For example, Internet Explorer Protected Mode (IEPM) uses MIC to run the Web browser in a process with limited access permissions.

The primary security problem that MIC addresses is unauthorized tampering with user data and, indirectly, with system state. A secondary problem MIC helps with is information disclosure. However, information disclosure is prevented only with respect to access to process address space.

Untrustworthy code can try to modify user data in many ways. Some attacks may try to manipulate data directly by creating, modifying, or deleting files. Other attacks target another process running at higher privilege, with the goal of getting arbitrary code to execute in another application that does have the required level of access. There are many types of cross-process attacks. Because of the wide range of application design and implementation, MIC does not provide a complete isolation barrier. MIC is not an application sandbox. However, it can be one of the security tools that application developers use to restrict the behavior of less trustworthy applications.

4.7.1.4. EARLIER INTEGRITY MODELS

Some traits of MIC are similar to earlier integrity models for computer security. However, MIC addresses primarily tampering or elevation of privilege in the Windows application environment. Previous integrity models were more concerned with maintaining integrity of trustworthy processes by enforcing policies that prevent the reading of untrusted data.

The Biba Model is based on a hierarchy of integrity labels and the access policies that are allowed when a subject IL dominates the object IL. MIC resembles the Biba Model in the following ways:

- it uses a hierarchy of integrity labels (integrity labels are not the same as security labels in the Bell-LePadula Model);
- the system uses a set of ordered subjects, objects, and IL's;
- the subject's IL dominates (is greater than or equal to) the object's IL;
- integrity policies inhibit access to objects but are not used primarily to limit the flow of information;

- preventing information disclosure is not the main goal.

MIC in Windows Vista does not prevent information disclosure other than with respect to access to process address space.

With respect to the Windows IL's, a lower value indicates less trustworthiness, and a higher value indicates greater trustworthiness. A lower-level subject cannot modify a higher-level object. The subject's IL is not dynamic. For example, the IL of a subject does not change to a lower value if the process reads data from a *low*-integrity object. The strict integrity model in the Biba Model does not allow a higher-integrity process to read lower-integrity data. This is sometimes called a "no-read-down" integrity policy. The Windows integrity policies, which are described in more detail below, do not inhibit or prevent higher-integrity subjects from reading or executing lower-integrity objects. There are many examples of attacks where reading malformed, untrusted input data results in an exploit of a vulnerability in an application and arbitrary code execution. MIC does not inhibit or prevent reading data at any level. Windows does not enforce a strict integrity policy as described in the Biba Model. The integrity design assumes that processes that are designed to handle untrusted data from an unknown or untrusted source are running at a lower IL, or that untrusted data is verified before use. However, MIC does not enforce that constraint.

MIC does not implement a dynamic low-watermark policy. A dynamic low-watermark policy changes the IL of the subject as the subject opens lower-integrity objects. An issue with dynamic integrity is when a *high*-integrity process obtains open handles to many objects based on *high* integrity, and then suddenly becomes a lower-integrity subject after it opens a particular *low*-integrity file. Forcing all open handles to higher-integrity objects to close when the IL changes affects the application's behavior. The dynamic lower-integrity process itself

becomes a target object of other processes at the lower IL. Such processes might now be able to modify the behavior of the application (at the same lower level) that has open handles to higher-integrity objects.

MIC is not suitable for integrity protection of data to support military or commercial requirements as described in the Clark-Wilson Model. The Windows implementation of integrity controls does not build on the concepts of Constrained or Unconstrained Data Items and certified Transformation Procedures. However, these concepts can be useful for application designers when they consider information flow from untrusted sources into higher-integrity processes.

Although MIC is similar to earlier integrity models in computer security, Windows Vista does not implement any of the models. Instead, MIC limits access permissions that are available to processes running with different privilege or trust levels.

4.7.2. Design

MIC is an extension of the Windows security architecture, which is based on the SRM in the kernel. The SRM enforces access control by comparing user and group SID's in the security access token with granted access permissions in the ACL of an object's security descriptor. MIC adds an IL to the security access token and a mandatory label ACE to the System ACL (SACL) in the security descriptor.

MIC's major parts are:

- predefined IL's and their representation;
- integrity policies that restrict access permissions;

- IL assigned to the security access token;
- mandatory label ACE;
- mandatory labels assigned to objects;
- integrity restrictions within the *AccessCheck* and kernel-mode *SeAccessCheck* Application Programming Interfaces (APIs).

Each of these parts is described in more detail below. MIC is always in effect, independent of other security policy options. The IL checks, like the access control checks, are not optional. There are no security policy options to disable IL checks. Although MIC supports UAC, MIC remains in effect when UAC is disabled by security Group Policy.

4.7.2.1. INTEGRITY LEVELS

Windows defines integrity levels (IL's) by using a SID. Using a SID to represent an IL makes it easy to integrate MIC into existing security data structures without requiring code changes. IL SID's have the following form: "S-1-16-xxxx". Table 1 shows the components of IL SID's.

Table 2 IL SID identifier authority values

Value	Description
16	Represents the Mandatory Label Authority (SECURITY_MANDATORY_LABEL_AUTHORITY).
xxxx	Represents the relative identifier (RID) field that is the IL. The RID is a hexadecimal value that represents the IL.

There are 4 primary IL's in Windows Vista with 4 corresponding values. A lower value indicates a lower IL or a lower level of trustworthiness. These values are

defined in the header file, `winnt.h`. Table 2 shows the defined IL's and their corresponding values.

Table 3 Defined IL's and corresponding values

Value	Description	Symbol
0x0000	Untrusted level	SECURITY_MANDATORY_UNTRUSTED_RID
0x1000	Low IL	SECURITY_MANDATORY_LOW_RID
0x2000	Medium IL	SECURITY_MANDATORY_MEDIUM_RID
0x3000	High IL	SECURITY_MANDATORY_HIGH_RID
0x4000	System IL	SECURITY_MANDATORY_SYSTEM_RID

An example of a *medium* IL SID is this string: "S-1-16-8192". The RID value of 8192 is the decimal equivalent of 0x2000.

The RID's are separated by intervals of 0x1000, which allows for definition of additional levels. The separation also allows assigning an IL to a process that is slightly higher than *medium*: for example, to meet specific system design goals.

The defined IL SID's have string name values associated with them. Using the API ***LookupAccountSID*** will return string names for each IL SID. Table 3 shows the string names for the IL's.

Table 4 IL string names

IL SID	Name
S-1-16-4096	Mandatory Label\Low Mandatory Level
S-1-16-8192	Mandatory Label\Medium Mandatory Level
S-1-16-12288	Mandatory Label\High Mandatory Level
S-1-16-16384	Mandatory Label\System Mandatory Level

The IL's are also defined as SID strings in the Security Descriptor Definition Language (SDDL). The SDDL defines the string format that the *ConvertSecurityDescriptorToStringSecurityDescriptor* and *ConvertStringSecurityDescriptorToSecurityDescriptor* functions use to describe a security descriptor as a text string. The language also defines string elements for describing information in the components of a security descriptor. Using the SDDL to define the IL's makes it convenient to define the IL of an object when the object is created. The SDDL support for mandatory labels is described in [Appendix A: SDDL for Mandatory Labels](#).

Windows Vista uses IL SID's in the security access token to represent the subject IL and uses IL SID's in the mandatory label ACE in a security descriptor to represent the object IL.

Example code for getting the access token IL is shown in [Appendix E: Getting the IL for an Access Token](#).

For information on tools that can be used to modify IL's, see [Appendix B: Icacs and File IL's](#) and [Appendix C: Chml and File IL's](#).

4.7.2.2. INTEGRITY POLICIES

MIC uses simple policies to determine how to use the mandatory labels on objects to restrict the level of access that is available to lower-integrity subjects. This means that the policies limit the type of access that lower-IL objects are allowed to have to higher-IL objects. Table 4 shows the two policy categories.

Table 5 Integrity policy categories

Category	Description
Access token mandatory policies	Set in the access token and determine how mandatory policies apply to the subject, which is represented in the access token.
Mandatory label policies	Set in the mandatory label ACE (described below) on objects, and determine how to restrict access to the object.

The integrity policies are used when the mandatory policy check is performed when evaluating access permissions on a securable object. The policies determine how access rights are restricted across IL's.

4.7.2.2.A) MANDATORY LABEL POLICIES

The mandatory label policies are defined as flags (bits) in the *Mask* field of the mandatory label ACE. These policy flags determine the restrictions on access permissions that apply to lower integrity subjects. Table 5 shows the mandatory label policies that are defined in Windows Vista.

Table 6 Mandatory label policies in Windows Vista

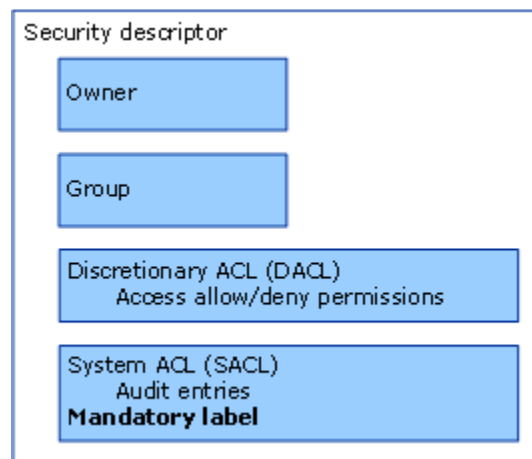
Policy	Description
SYSTEM_MANDATORY_POLICY_NO_WRITE_UP	The default policy on all object mandatory labels. The flag is equivalent to the NO_WRITE_UP access token policy. The policy restricts write access to the object by a subject with a lower IL.
SYSTEM_MANDATORY_POLICY_NO_READ_UP	Restricts read access to the object by a subject with a lower IL. The policy is used to restrict read access to the virtual memory address space of a process.
SYSTEM_MANDATORY_POLICY_NO_EXECUTE_UP	Restricts execute access to the object by a subject with a lower IL. The policy is used to restrict launch activation permissions on a COM class by lower-integrity subjects.

These defined policies meet the design goals for Windows Vista. The architecture of MIC allows for expansion by defining additional policy options that control access permissions between subjects and objects at different IL's.

4.7.2.3. MANDATORY LABEL ACE

MIC defines a new ACE type, the system mandatory label ACE. The mandatory label ACE is used to represent the mandatory label of an object in the object's security descriptor. The mandatory label contains the IL and associated integrity policy. A mandatory label ACE is used only in the SACL of the security descriptor. The SACL is a separate field in the security descriptor from the DACL.

Figure 4 Security descriptor fields



The DACL contains user and group access permissions to the object. Any user or group can make updates to the DACL with *WRITE_DAC* object access permissions. DACL's can be updated more frequently and by different users. One of MIC's features is that the security system maintains the label with an object after a mandatory label is applied to the object. Enforcing the appropriate mandatory label in the object security descriptor, with little or no impact to applications designed to manage ACL's, is accomplished by putting the

mandatory label in the SACL, where it is primarily under the control of the security system. The mandatory label is logically separate from the system audit entries in the SACL. There is no required order for the mandatory label to precede or follow the system audit entries in the SACL.

The mandatory label ACE is similar to an access allowed ACE, in that it contains an ACE header, an access mask, and a SID. The SID portion of the mandatory label ACE contains the IL SID. Table 6 shows the fields in the ACE header.

Table 7 Fields contained in the ACE header

ACE header field	Value
AceType	SYSTEM_MANDATORY_LABEL_ACE_TYPE
AceFlags	Control flags that define inheritance of mandatory label ACE type
AceSize	Size of the mandatory label ACE

The mandatory label ACE contains a **Mask** field. The mask is used to define the mandatory policies that determine restrictions on the access permissions that apply to processes (or subjects) with a lower IL. An example policy used widely in Windows Vista is the *NO_WRITE_UP* policy. The *NO_WRITE_UP* policy flag in the mandatory label ACE mask means that subjects with a lower IL (in the access token) than the IL SID in the mandatory label on the object are restricted from getting generic *write* access to the object.

There is only one effective mandatory label on an object. If a security descriptor happens to get more than one mandatory label in the SACL, the first mandatory label ACE in the SACL is the effective label for the object.

4.7.2.4. HOW WINDOWS ASSIGNS A MANDATORY LABEL TO OBJECTS

Windows assigns a mandatory label to a securable object when the object security descriptor is created. The IL on a new object is assigned in one of three ways:

- the SRM assigns the object a mandatory label when the security descriptor is created for the object;
- the creating process specifies an explicit mandatory label as security attributes when creating the object using a function, such as *CreateFile*;
- the parent container defines an inheritable mandatory label ACE that applies to child objects that are created in the container.

The easiest way to understand how object IL's are assigned is maybe to assume that every object is assigned a mandatory label with the same IL as the subject IL of the creating process (or thread). However, there are exceptions to the general idea based on object type and subject IL. The exceptions are the result of design choices that are necessary to support other system requirements for a consistent user experience when various security policies, such as UAC, are enabled or disabled.

Mandatory labels can be applied to all securable objects that have access control based on the security descriptor. There are many types of securable objects, including process and thread objects, named objects, and persistent objects such as files or registry keys. Windows Vista does not assign explicit mandatory labels to all object types when objects are created. The object types to which the SRM assigns mandatory labels at object creation are:

- processes;

- threads;
- access tokens;
- jobs.

All other object types have either an implicit default or an inherited mandatory label. Recall that, during access token initialization, an IL is assigned to the security access token that is created during an interactive logon. The first process that is launched on behalf of a user logon is *userinit.exe*, which starts the shell process, *explorer.exe*. *Userinit.exe* is started by a system service (*Winlogon*) that uses a call to *CreateProcessAsUser* and passes in the access token for the interactive logon user.

CreateProcessAsUser creates a process object and an initial thread, among other things. When the process object is created, the security descriptor for that process is assigned the IL from the access token that is assigned as the primary access token to the new process. When *userinit.exe* calls *CreateProcess* to launch the shell, the process object for *explorer.exe* is initialized. The process object includes a security descriptor and primary access token. The mandatory label on the *explorer.exe* process is set to the IL of the creating process, *userinit.exe*, which is *medium*. The primary access token for *explorer.exe* is inherited from the creating parent process, *userinit.exe*, and has an IL of *medium*. When the *explorer.exe* process creates a new thread, the thread object is given a security descriptor, and the SRM assigns an IL to the thread object based on the IL of the creating process. The thread objects within the *explorer.exe* process are assigned an IL of *medium*, which is the IL of the primary access token of the creating process.⁶

⁶ An access token object is a securable object with its own security descriptor. The token's security descriptor is used to determine allowed access during *OpenProcessToken* or *OpenThreadToken* functions. The access

By always assigning mandatory labels to process, thread, token, and job objects, MIC prevents processes for the same user at lower IL's from accessing these object types and modifying their content or behavior, such as injecting a Dynamic-Link Library (DLL) or impersonating a higher-level access token.

4.7.2.4.A) DEFAULT IL

Not all object types are assigned a mandatory label ACE in the security descriptor. If a mandatory label ACE is present in the security descriptor, that is called an “explicit mandatory label”. If no mandatory label ACE is present, the SRM uses an implicit default mandatory label for that object during the mandatory policy check. The default mandatory label assigns a *medium* IL for all securable objects. If a mandatory label is not defined in the security descriptor, the implicit default mandatory label applies to all object types.

The default object IL of *medium*, combined with the default mandatory policy of *NO_WRITE_UP*, restricts modify access to all objects by processes with a subject IL less than *medium*. The default mandatory label and policy prevent untrustworthy processes at *low* integrity from modifying any user or system files or registry keys on the system that may otherwise allow discretionary *write* access in the DACL.

The NTFS file system objects and registry keys are not automatically labeled when they are created. These objects do not have mandatory labels after upgrade from a previous version of Windows to Windows Vista. Files on non-NTFS file systems (CDFS or FAT32) that do not have security descriptors are not securable objects and do not have an IL. Every security descriptor must have an implicit mandatory label.

token object has a mandatory label in the security descriptor on the access token object. The access token also contains an IL.SID in the access token groups list that represents the subject IL.

Processes with a subject IL at or above *medium* create files and registry keys without an explicit label. As a consequence, the file system and registry objects that are created by a *high* or *system* IL process have an implicit *medium* label. Applications that use mandatory labels can define explicit labels when creating objects. However, Windows Vista does not assign labels higher than *medium* IL to the file system or registry by default. That does not mean that these objects are necessarily open to modification by lower-integrity processes. The inherited (or default) DACL on file system or registry objects that were created by a *high*- or *system*-level process grants *write* access only to members of the Administrators group or to local System or service accounts.

A number of design constraints required using the default implicit mandatory label of *medium*, instead of assigning an explicit mandatory label based on the subject's IL for most object types. A specific example is based on the ability to enable or disable UAC by using local security policy. When UAC is disabled, a user who is a member of the local Administrators group has all processes running with a full privilege access token at a *high* IL. If all objects are explicitly labeled at the subject's IL, then all files such as documents and spreadsheets that the user creates would be assigned a *high* IL. The *high* label would seem appropriate, even though the inherited DACL permissions for the user profile provide sufficient access control for user access. However, if UAC is enabled by local computer or Group Policy, most processes run by the same user are assigned a filtered security access token at a *medium* IL. After UAC is enabled, the user would not be able to open files that were created when UAC was disabled. A *medium*-integrity application that tried to open the user's *high*-integrity documents would receive an *Access Denied* error.

If a process is running with a subject IL less than the default IL of *medium*, that process will have restricted access permissions to all objects that have an implicit

medium IL. Processes with a *low* IL will not be able to modify any object with either an explicit or implicit IL of *medium* or higher, regardless of the access rights granted in the DACL to the security principal. Therefore, all objects that are created by a process with a subject IL less than the default level (*medium*) are explicitly labeled by the SRM. Access restrictions on *low*-integrity processes are possible in Windows Vista because all applications generally run at the *medium* IL and the application compatibility issues are minimal. Applications that run correctly at *low* integrity generally require specific design changes to work correctly with restricted access.

4.7.2.4.B) LABELING OBJECTS CREATED BY *LOW* SUBJECTS

Applications can be started using the *CreateProcessAsUser* function at a *low* IL. A *low* process is initialized by *CreateProcessAsUser* with the following IL information:

- the new process object is created with a security descriptor containing a mandatory label with *low* integrity;
- a new thread object is created for that process and with a security descriptor that contains a mandatory label with *low* integrity;
- a primary access token is assigned to the new process; the access token object has a security descriptor with a mandatory label at *low*, and the access token contains a *TokenIntegrityLevel* with a *low*-integrity SID that represents the subject IL.

Assuming that the *low*-integrity process is running, and that it wants to create a thread: to create a thread, it must open its own process object for *write* access to create a new thread object; if the mandatory label on the process object were *medium*, the *low* subject would fail to open the process and would not be able to create a new thread, but, because the process object is also labeled at *low* integrity,

the *low* subject is allowed to open the process for *write* access and to create a new thread object. The example shows why it is important that the mandatory labels on the process object, thread objects, and security access token be consistent at the same IL.⁷

Supposing that the *low* process creates a temporary file: the application calls *CreateFile* to create a new file, write some data, and close the file; later, the *low* process wants to re-open the same file to append data; if the temporary file has an implicit default mandatory label at *medium* IL, the *low* process will not be able to reopen the file that it created earlier for *modify* access. The same issue could arise for any securable object that a subject with an IL below the default level of *medium* creates. Therefore, all securable objects created by a subject with an IL below the default level are automatically assigned an explicit mandatory label. In other words, all kernel objects, registry keys, and file system objects are explicitly labeled at *low* when the subject IL is *low*.

4.7.2.4.C) CREATING AN OBJECT WITH A SPECIFIC MANDATORY LABEL

Most Windows applications do not need to be “integrity-aware.” The SRM automatically creates objects and assigns them a mandatory label. Because most applications run at a *medium* IL, and the default object IL is *medium*, the integrity policy does not change the allowed access control for most applications. However, some applications, such as services, are designed to support client applications at different IL’s. The service might be running at a higher IL than the client, and might want to label new objects created on behalf of the client explicitly at a lower IL. The IL of the object can be set by the creating process. The constraint is that the IL on the new object must be less than or equal to the IL of the creating process.

⁷ This applies for all IL’s, not only *low* subjects.

Creating an object with a specific mandatory label

1. Create SDDL security descriptor that defines *low* mandatory label, e.g.

```
#define LOW_INTEGRITY_SDDL_SACL_W L"S:(ML;;;NW;;;LW) "
```
2. Convert SDDL string to security descriptor using ***ConvertStringSecurityDescriptorToSecurityDescriptor***.
3. Assign security descriptor with *low* mandatory label to security attributes structure.
4. Pass security attributes parameter to call to create object, such as ***CreateFile***.

4.7.2.5. MANDATORY LABEL INHERITANCE

Windows Vista does not explicitly label files and directories in the NTFS file system. As mentioned previously, the SRM uses an implicit mandatory label with a default level of *medium* for objects that do not have a mandatory label in the security descriptor.

Applications can be designed to run at a *low* IL. IEPM is an example of a Windows Vista application that is designed to run at *low* integrity. Applications at *low* integrity might need folders in the file system where they can write temporary files or application data. In the case of IEPM, the **Temporary Internet Files** folder in the user's profile is used. A *low* application can write to a file system folder by assigning the folder an explicit mandatory label that permits *write* access from a *low*-integrity process.

Depending on the application design, there might be other cooperating processes that add files to the folder used by the *low*-integrity application. If the other

processes are also running at *low* integrity, the files created by those processes are automatically assigned a *low* mandatory label. However, if the other partner process has a higher integrity, the partner process (e.g. a file synchronization service, or user agent) might create files in the *low* folder that are not automatically labeled at *low*. The partner process creates a *medium* file in the folder used by the *low* application, which the *low* application cannot modify or delete.

MIC allows a folder with a *low* mandatory label to have child objects, files, and subfolders, each with a different, higher mandatory label because each file system object has a unique security descriptor. There are many scenarios where the intention is that, for a folder that is assigned a *low* mandatory label, all files in the folder must be assigned a *low* mandatory label so that a *low* process can modify them. MIC supports this by enabling mandatory label ACEs to be inheritable from the parent container to sub-containers and child objects.

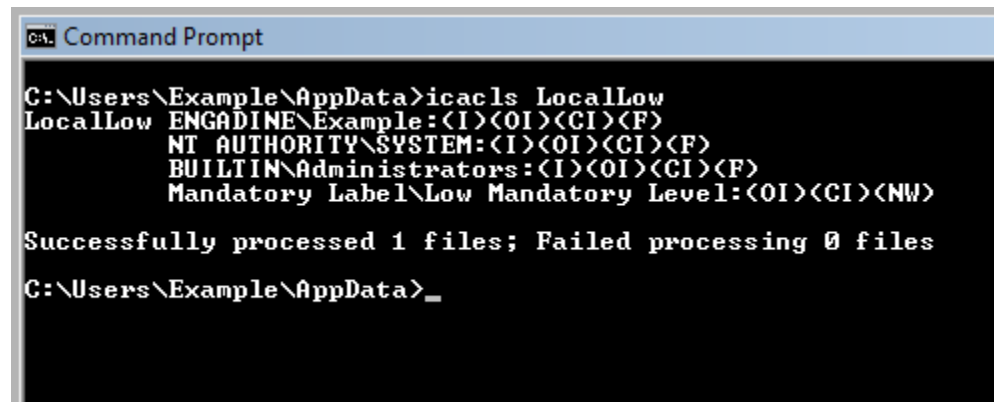
The mandatory label ACE type data structure contains an ACE header with an *AceFlags* field. The *AceFlags* field is used to define inheritance flags for the ACE type that are the same as inheritance flags for other ACE types, such as the “Access Allowed” ACE type that is used for discretionary access. Mandatory label ACEs can be defined as inheritable, for ***container inherit (CI)*** and/or ***object inherit (OI)***. Mandatory label ACEs cannot be ***inherit only (IO)***. If there is an inheritable mandatory label ACE assigned to a container, the mandatory label applies to the container itself and to the child objects for which inheritance applies.

An example of an inheritable mandatory label is the *low* mandatory label on one of the folders created under every user profile: %USERPROFILE%\AppData\LocalLow. This folder is assigned a *low* mandatory label when the profile is initialized and intended as the top-level folder that is

writable by default by *low*-integrity applications. The following image shows the inheritable mandatory label on the `AppData\LocalLow` folder displayed using the `Icacls` command. For more information on how `Icacls` supports mandatory labels, see [Appendix B: Icacls and File IL's](#).

The inheritance flags in the mandatory label indicate that the ACE is *(OI)* and *(CI)* ACE with a *NO_WRITE_UP (NW)* mandatory policy. All subfolders that are created under the `AppData\LocalLow` folder will inherit the inheritable label.

Figure 5 Inheritable *low* mandatory label

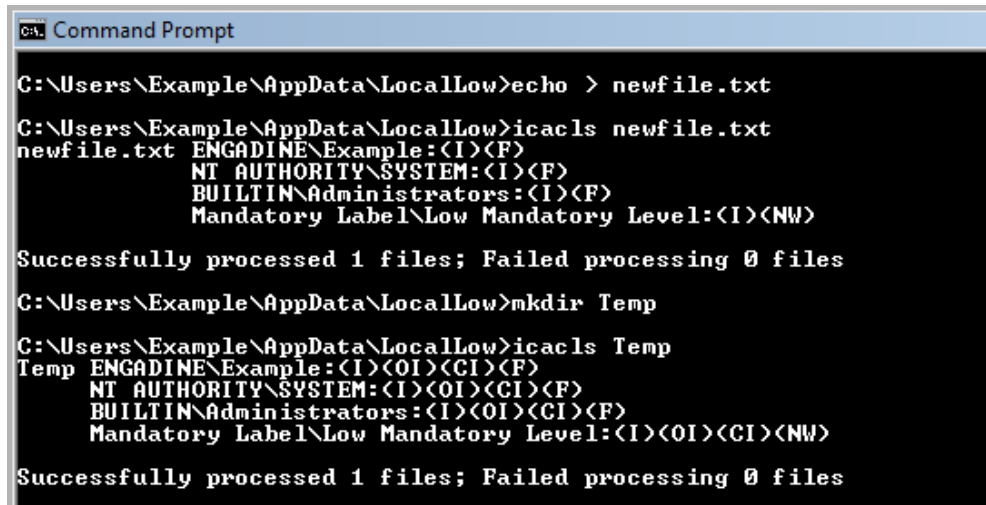


```
Command Prompt
C:\Users\Example\AppData>icaccls LocalLow
LocalLow ENGADINE\Example:(I)(OI)(CI)(F)
          NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
          BUILTIN\Administrators:(I)(OI)(CI)(F)
          Mandatory Label\Low Mandatory Level:(OI)(CI)(NW)

Successfully processed 1 files; Failed processing 0 files
C:\Users\Example\AppData>_
```

When a new file or subfolder is created from a *medium* process, the new object inherits the *low* mandatory label. In the following example, the command prompt is running in process with *medium* IL. A file, `newfile.txt`, and a new folder, `Temp`, is created in the `LocalLow` folder, and both objects inherit the *low* mandatory label from the parent container. `Icacls` indicates that the mandatory label is inherited with the *(I)* designation.

Figure 6 Inheriting a mandatory label on a child object



```
C:\Users\Example\AppData\LocalLow>echo > newfile.txt
C:\Users\Example\AppData\LocalLow>icacls newfile.txt
newfile.txt  ENGADINE\Example:(I)(F)
              NT AUTHORITY\SYSTEM:(I)(F)
              BUILTIN\Administrators:(I)(F)
              Mandatory Label\Low Mandatory Level:(I)(NW)

Successfully processed 1 files; Failed processing 0 files

C:\Users\Example\AppData\LocalLow>mkdir Temp
C:\Users\Example\AppData\LocalLow>icacls Temp
Temp  ENGADINE\Example:(I)(OI)(CI)(F)
      NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
      BUILTIN\Administrators:(I)(OI)(CI)(F)
      Mandatory Label\Low Mandatory Level:(I)(OI)(CI)(NW)

Successfully processed 1 files; Failed processing 0 files
```

Inheritance for mandatory labels ensures consistency in the IL of objects that are created under a portion of the file system namespace.

4.7.2.5.A) INHERITANCE AND EXPLICIT LABELS

Objects created in a container that has an inheritable mandatory label inherit the mandatory label from the container. However, the process that creates a new object such as a file can provide an explicit mandatory label in the security attributes parameter to the create function, such as *CreateFile*.

If the creating process provides an explicit label as a parameter to the object create function, the new object is assigned the explicit mandatory label provided by the caller and does not inherit from the parent container. The explicit mandatory label can have an IL no higher than the subject process creating the object. The IL in the explicit mandatory label provided by the subject might be higher than the IL in the parent container's inheritable mandatory label.

4.7.2.5.B) INHERIT-ONLY RESTRICTION

Inherit-Only is a flag in an ACE that indicates that the ACE does not control access to the object to which it is attached. IO ACEs are usually assigned to container objects. The IO ACE is not effective on the container itself; rather, it applies to child objects of the container. There is a restriction on subjects with an IL less than the default for setting *INHERIT_ONLY* labels on new objects. If the explicit label passed in for a new container object is an *INHERIT_ONLY* label with a level less than the default, the label is considered invalid and ignored. The rationale for this restriction is that a *low* subject creates a container and tries to set an *INHERIT_ONLY* label on the container at *low*. If the *INHERIT_ONLY* label were accepted, the effective IL for the container would be the implicit default level of *medium*.

Furthermore, subjects cannot define an *INHERIT_ONLY* label with an IL higher than the subject's IL. For example, a *medium* subject cannot define an *INHERIT_ONLY* label with an IL of *high*.

4.7.2.5.C) PROTECTED SACL AND LABEL INHERITANCE

SDDL supports defining a protected SACL, which might include an explicit mandatory label. The protected SACL sets the *SE_SACL_PROTECTED* flag in the *SECURITY_DESCRIPTOR_CONTROL* field of the security descriptor. The logical separation of *LABEL_SECURITY_INFORMATION* and *SACL_SECURITY_INFORMATION* with respect to access rights is not complete with respect to how a protected SACL behaves. Mandatory labels are stored in the SACL. A protected SACL implies that the mandatory label is also protected.

If the *SE_SACL_PROTECTED* flag is set in the *SECURITY_DESCRIPTOR_CONTROL*, then the mandatory label is also protected.

- If there is no mandatory label ACE in the protected SACL, an inheritable mandatory label ACE from the container is not applied (the new object has an implicit default mandatory label).
- If there is a mandatory label ACE in the protected SACL, changes to the inheritable label ACE on the container do not affect this object.

For more information about SDDL, see [Appendix A: SDDL for Mandatory Labels](#).

4.7.2.6. HOW ACCESS CHECKS WORK WITH MANDATORY POLICY
The ***AccessCheck*** function enforces the mandatory policy. *AccessCheck* compares the specified security descriptor with the specified access token and indicates, in the ***AccessStatus*** parameter, whether access is granted or denied. The *AccessCheck* function first compares the IL in the ***ClientToken*** with the mandatory label in the SACL of ***pSecurityDescriptor*** to determine what access rights are not available, based on the mandatory policy. After the mandatory policy check, the *AccessCheck* compares the desired access against the access rights granted in the DACL.

The default mandatory policy prevents lower-integrity processes from gaining generic *write* access to higher-integrity objects. For example: by default, a *low*-integrity process is denied generic *write* access to an object with a higher IL. If *pSecurityDescriptor* does not contain a mandatory ACE, an implicit mandatory ACE is assumed that assigns the object a *medium* IL. The ***GenericMapping*** parameter determines what access rights are associated with generic *write* access. If the *GenericMapping* parameter is all zeros, then the integrity check will not grant any specific access rights to a lower-integrity *ClientToken*. After the integrity check determines the generic access rights that are available to the caller based on the

mandatory policy, the security descriptor's DACL is compared with *ClientToken* to determine the remaining granted access rights.

4.7.3. *Implementation*

MIC is used in a number of ways in Windows Vista. Its main purpose is to restrict the access permissions of applications running under the same user account that are less trustworthy. MIC prevents less trustworthy code from modifying objects at a higher level. Most objects under the control of the *Administrators* group or *System* have a DACL that typically grants full control permission to *Administrators* and to *System*, and *read* and *execute* permissions to authenticated users. Examples of resources under control of the *Administrators* group and *System* are the Program Files directory for applications or the HKEY_LOCAL_MACHINE (HKLM) hive of the registry. MIC does not enhance the security of objects that are already properly configured to restrict different user accounts or groups from accessing them. MIC's primary purpose is to address different permissions for programs to access resources under the full control of the same user security principal.

The resources under the control of the same user security principal that need additional protection are primarily under the user's profile (C:\Users\

- in UAC, it limits access between processes running with standard user privilege and elevated processes running with full administrative rights in Admin Approval Mode;

- COM security is aware of IL's and does not allow lower-integrity clients to bind to class instances running at a higher IL;
- in the default security settings, it restricts access to the root folder of the system volume;
- in IEPM, it limits the ability of code running in the Internet browser to modify user data or user profile settings;
- to enable applications running at *low* integrity to have a writeable file location, it assigns specific folders in the user profile a *low* IL.

MIC is part of the Windows Vista security architecture. Over time, specific applications that handle untrustworthy input (primarily Internet-facing) are updated to take advantage of the ability to run at a *low* IL. Personal productivity applications are fine running at a *medium* IL as long as the user knows the source of input data. For most applications, MIC is completely transparent and does not interfere with application features. Application services can be updated to provide better isolation of server resources for client processes at different IL's.

4.7.3.1. IEPM AT *LOW* INTEGRITY

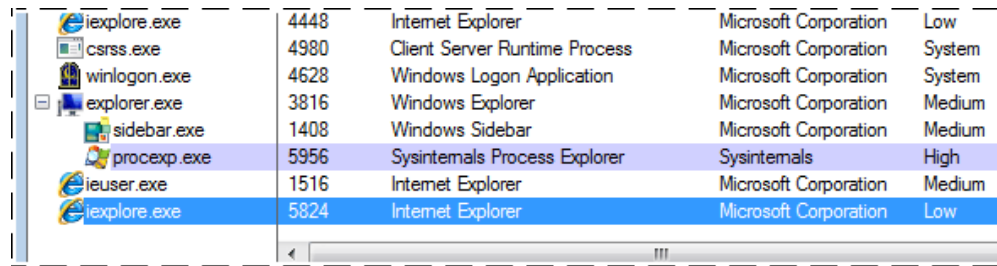
Internet Explorer (IE) is an example of an application that is designed to accept arbitrary data and extensible code from the Internet. Because the source of Internet content is rarely authenticated (signed), it should be assumed that all input from the Internet is untrustworthy. Attacks against IE, or against any other Internet browser, demonstrate the untrustworthy nature of dynamic content and data available from the Internet. From the security perspective, it should be assumed that the IE process itself is compromised and untrusted, and solutions should be looked for that limit the potential damage done by attacks on the browser.

The goal of IEPM is to reduce the access rights available to the process, in order to limit the ability of an exploit running in the browser to create unwanted startup files, modify user data files, make annoying changes to browser configuration settings, or drive the behavior of other programs running on the desktop. All code running in IEPM in a *low*-integrity process is considered untrustworthy. The default *medium* IL for objects prevents the browser from opening any directories, files, or registry keys for *write* access, except those that are explicitly labeled at *low* integrity. UIPI prevents the *low*-integrity browser code from sending any potentially damaging window messages to other applications that are running on the desktop.

The browser that is running at *low* integrity does have *read* access to user data files. Because MIC does not enforce confidentiality, it does not restrict the flow of information. The process can also use default credentials to initiate network connections, such as to a network proxy server (necessary when authentication is required to connect to the Internet), or to a network printer device to print a Web page. The *low*-integrity process can also initiate an authenticated connection to other network services and thereby authenticate to those servers as the current user.

The application design of IE needed some restructuring to run under Protected Mode at a *low* IL. The primary change is that certain program operations were moved out into a separate process, known as a broker process, running at a *medium* IL. The broker process is started at *medium* IL when the user clicks on the IE icon or on a URL link. The broker checks the URL and zone policy and launches a child process, *iexplore.exe*, at the *low* IL to make the Internet connection and render the Web page. The following image from Process Explorer shows the *ieuser.exe* broker process at *medium* IL, and the *iexplore.exe* process at *low* IL.

Figure 7 IEPM processes



iexplore.exe	4448	Internet Explorer	Microsoft Corporation	Low
csrss.exe	4980	Client Server Runtime Process	Microsoft Corporation	System
winlogon.exe	4628	Windows Logon Application	Microsoft Corporation	System
explorer.exe	3816	Windows Explorer	Microsoft Corporation	Medium
sidebar.exe	1408	Windows Sidebar	Microsoft Corporation	Medium
procexp.exe	5956	Sysinternals Process Explorer	Sysinternals	High
ieuser.exe	1516	Internet Explorer	Microsoft Corporation	Medium
iexplore.exe	5824	Internet Explorer	Microsoft Corporation	Low

Everything the user experiences in the IE Web browser is done inside the *low*-integrity process. A few specific operations, such as changing Internet Options settings, or the *Save as file* dialog, are handled by the broker process. If the URL is a trusted site, based on the default zone policy settings, the broker process starts a different instance of *iexplore.exe* in a *medium*-integrity process. All browser extensions and ActiveX controls run inside the *low*-integrity process. This has the advantage that any potential exploit to a browser extension is also running at *low* integrity.

IE is somewhat more complicated than other applications because it hosts browser extensions and ActiveX controls that are not developed by Microsoft, it launches other applications based on mime types for different file extensions, and it integrates the client window from different applications within a single parent window frame. Users also download software from the Internet through the browser and immediately launch an application package or application installer. Many of these operations require help from the broker process at a higher IL to mediate through the user to confirm an operation. Otherwise, code running in the browser could install malicious software on the system and try to modify or delete the user's data. Installation of ActiveX controls is performed using an elevated task started by UAC that requires administrator rights.

Notwithstanding, the integrity mechanism that limits *write* access to objects is not a complete isolation mechanism.

4.8. Conclusion

With the analysis of these systems, it can be concluded that, although not certified by TCSEC as being robust enough to separate TS from U, several of them are, nevertheless, flexible and powerful. Through proper configuration, they can result in highly-secure systems. This shows and proves the value of MAC as a powerful security mechanism.

Chapter 5

SYSTEM DESIGN & IMPLEMENTATION

This concept is a modification of MIC (in that it is modified as to work also as a variation of the Bell-LaPadula Model), which is an extension of the Windows security architecture.

The major parts of the concept are:

- configuration tweaking of integrity policies that restrict access permissions;
- IL assigned to Web browsers and e-mail clients;
- explicit mandatory label assigned to objects created by Web browsers and e-mail clients (when necessary).

Each of these parts is described in more detail below.

The concept can be used in two main ways in Windows Vista. Its main purpose is to prevent sensitive information disclosure. The concept prevents Web browsers, e-mail clients, and processes resulting from executing objects created by these applications from reading sensitive data. Under the current scheme (in Windows Vista), the *NO_READ_UP* mandatory label policy is only used to restrict (*read*) access to the virtual memory address space of a process, which enables the mentioned subjects to read and, thus, collect sensitive data from sensitive files. Examples of objects created by the mentioned applications whose processes have *read* access to sensitive files are attachments saved from e-mails. This concept enhances the security of sensitive files by enabling the *NO_READ_UP*

mandatory label policy on them, by running Web browsers and e-mail clients at a *low* IL, and by assigning an explicit *low* mandatory label to objects created by these applications (when necessary). The solution's primary requirement of disabling spyware is, with this, addressed.

The sensitive files that need protection are primarily user data files under the user's profile (C:\Users\). The concept can be used by Windows Vista in the following ways:

- it can restrict *read* access to the user profile folder by processes resulting from executing objects created by Web browsers and e-mail clients;
- in Web browsers and e-mail clients, it can limit the ability of code running in these applications to read user data.

The concept is designed to be a modification of MIC and a part of the Windows security architecture. Web browsers and e-mail clients can be updated to take advantage of the ability to run at a *low* IL and create objects with an explicit *low* mandatory label (when necessary). Personal productivity applications are fine running at a *medium* IL because only applications running at a *low* IL are blocked from reading user data files (typically assigned a *medium* IL). For the user, the mechanism is completely transparent.

5.1. User Profile Folder with ***NO_READ_UP*** Mandatory Label Policy

The user profile folder, typically C:\Users\ (Windows Vista)⁸, has historically been used as a convenient place to store sensitive files, and the practice is even encouraged. Personal files created by users are often saved under the user profile folder. The default security policy for the user profile folder is

⁸ In previous versions of Windows (such as Windows XP), the user profile folder was typically C:\Documents and Settings\.

designed to allow any subjects to read files under it (unless denied in the DACL). In fact, the policy allows even lower-integrity subjects to read files under it, regardless of who created them (unless denied in the DACL). This policy does not properly protect user privacy.

This concept uses the *NO_READ_UP* mandatory label policy to disable spyware (and, thus, protect user privacy). This means that the *NO_READ_UP* mandatory label policy prevents (sensitive) information disclosure.

This is done by setting the corresponding *NO_READ_UP* flag (bit) in the *Mask* field of the mandatory label ACE of user data files. By doing this, this defined policy flag will restrict *read* access to user data files (typically assigned a *medium* IL) by Web browsers and e-mail clients, and by processes resulting from executing objects created by these applications (according to this concept, all properly assigned a *low* IL).

Therefore, by enabling the *NO_READ_UP* mandatory label policy in user data files, together with running Web browsers and e-mail clients at a *low* IL and assigning an explicit *low* mandatory label to objects created by these applications (when necessary), the goal of preventing sensitive information disclosure is met. *Low*-IL applications such as the ones mentioned, and *low*-IL processes resulting from executing *low*-IL objects created by them cannot read files under the user profile folder, even if the DACL grants *read* access. The user profile folder has an inheritable mandatory label at *medium* IL, with a *NO_READ_UP* mandatory policy, that is object and container inherit, and, as such, propagates to files and subfolders. This is done by enabling the corresponding CI and OI inheritance flags in the *AxFlags* field in the ACE header of the mandatory label ACE of the user profile folder.

Concluding, this defined policy meets, then, the design goal of disabling spyware for the generated concept.

5.2. Web Browsers and E-Mail Clients at *Low Integrity*

Web browsers and e-mail clients are examples of applications that are designed to accept arbitrary data and extensible code from the Internet. Because the source of Internet content is rarely authenticated (signed), it should be assumed that all input from the Internet is untrustworthy. Attacks against Internet-facing applications demonstrate the untrustworthy nature of dynamic content and data available from the Internet. From the security perspective, it should be assumed that these applications' processes themselves are compromised and untrusted, and solutions that limit the potential damage done by attacks on these applications should be looked for. Some proposed solutions to Internet-based attacks (such as IEPM) try to maintain a rich, highly collaborative user experience. Unfortunately, rich, highly collaborative user experience allows for malicious program behavior, such as unauthorized collection of sensitive data. Rich, highly collaborative user experience does not allow for proper user privacy protection, if the applications that handle untrustworthy input (such as Web browsers and e-mail clients) have *read* access to user data files.

As such, in this concept, Web browsers and e-mail clients are assigned the *low* IL (as happens with IEPM). Applications must be specifically designed to be able to run with a *low* IL, as that means running with the fewest rights possible. These applications might need folders in the file system where they can write temporary files or application data. Folders like the `Temporary Internet Files` in the user's profile (used by IEPM) can be created and used for this goal.

The goal of assigning a *low* IL to Web browsers and e-mail clients is to reduce the access rights available to the processes, in order to limit the ability of an exploit

running in one of these applications or the applications themselves to read user data files. All code running in these applications should be considered untrustworthy. The default *medium* IL for user data files prevents them from opening any user data file for *read* access, except those that are explicitly labeled at *low* integrity.

As such, Web browsers and e-mail clients that are running at *low* integrity do not have *read* access to user data files. Because this concept enforces confidentiality, it restricts the flow of information.

Because the source of Internet content is rarely authenticated (signed), this concept that limits *read* access to sensitive files is (together with MIC), in some ways, an almost-complete isolation mechanism. The design of Web browsers and e-mail clients should, then, specifically tailor the behavior of their processes to maintain least-privilege functionality and allow for proper user privacy protection.

5.3. Objects Created by Web Browsers and E-Mail Clients Are Assigned the *Low* IL

Applications running at a *low* IL (such as IEPM, or, under this concept, any Web browser or e-mail client) might use special separate broker processes to handle certain specific operations, such as saving a file downloaded from the Internet (recall IEPM and the *Save as file* dialog). These special separate broker processes might run at an IL higher than *low*, and, as such, the objects that they (eventually) create are typically assigned an IL higher than *low* as well (as is the case with files saved from the Internet in IEPM, an operation that is handled by the above-mentioned *medium*-IL *Save as file* dialog). Unless denied in the DACL, subjects with such an IL (higher than *low*) have permissions to every *medium* and *low* IL object, such as the *read* permission that allows the processes resulting from executing these objects to read (for example) *medium* IL objects such as user data

files, regardless of eventually-enabled mandatory label policies. As such – and in order to prevent this from happening –, in this concept, objects created by Web browsers’ and e-mail clients’ broker processes running at an IL higher than *low* are assigned an explicit *low* mandatory label.⁹

Creating an object with a specific *low* mandatory label

1. Broker process creates SDDL security descriptor that defines *low* mandatory label, e.g.:

```
#define LOW_INTEGRITY_SDDL_SACL_W L"\"S:(ML;;;NW;;;LW)\".
```

2. Broker process converts SDDL string to security descriptor using ***ConvertStringSecurityDescriptorToSecurityDescriptor***.
3. Broker process assigns security descriptor with *low* mandatory label to security attributes structure.
4. Broker process passes security attributes parameter to call to create object, such as *CreateFile*.

Concluding, although the creating process (the broker process) has an IL higher than *low*, the objects created by these subjects are not assigned an IL higher than *low*. The objects created by these subjects have an explicit *low* IL.

The IL change for objects created by Web browsers’ and e-mail clients’ broker processes running at an IL higher than *low* describes a change to the IL to improve user privacy. The change for the IL improves user privacy by allowing

⁹ Objects created by subjects (such as broker processes) running at a *low* IL are automatically assigned an explicit *low* mandatory label (recall that all securable objects created by a subject with an IL below the default level are automatically assigned an explicit mandatory label), and, therefore, there is no need to manually assign an explicit *low* mandatory label.

the *NO_READ_UP* mandatory label policy to be activated when (*low*-IL) processes resulting from executing these (*low*-IL) objects attempt to read higher-integrity objects such as (*medium*-IL) user data files, thus preventing the access from taking place. The *low* IL for these objects is consistent with the idea that all input from the Internet is untrustworthy due to the fact that the source of Internet content is rarely authenticated (signed). User data files assigned the *medium* IL are, with this, protected not only from *write* access but also from *read* access by a lower-integrity process such as one resulting from executing such an object (assigned a *low* IL). The process access right that is available to lower-integrity processes such as the ones resulting from executing such objects to open a user data file is, with this, limited to *execute*.

5.4. Scenario Solution

Recalling the explicit example scenario, under the current scheme, if a Windows Vista user receives an attachment in an e-mail and saves it, it is written with *medium* integrity because the process handling the saving operation is running at *medium* IL so that the user can save the file in *medium*-IL locations such as the user profile, and, as such, the attachment gets an implicit default *medium* mandatory label. When the user executes the attachment, its process runs at *medium* integrity because the file object is labeled *medium*, therefore, the user's data (labeled *medium*) is not protected against malicious reads or writes by the attachment. It will, as such, be able to read, write to, and delete the user's data.

By running Web browsers and e-mail clients at *low* IL, and by assigning an explicit *low* mandatory label to objects created by these applications when necessary (two of the three main features of this concept), if a Windows Vista user receives an attachment in an e-mail and saves it, it is written with *low* integrity because it came from the Internet – an untrusted source. When the user executes the attachment, its process runs at *low* integrity because the file object is labeled *low*, therefore, the

user's data (labeled *medium*) is protected from malicious writes by the attachment (a positive "side effect" resultant from the changes implemented by this concept). It will still, however, be able to read the user's data. MIC implements a form of the Biba Model, which ensures integrity by controlling writes and deletions. Contrast this with the more well-known Bell-LaPadula Model (of which a form is implemented by this concept, and whose implementation in Windows Vista is dependent on the activation of the *NO_READ_UP* mandatory label policy), which describes levels of confidentiality by controlling reads.

This is, then, where the *NO_READ_UP* mandatory label policy comes into play. By enabling the *NO_READ_UP* mandatory label policy in user data files (the remaining main feature of this concept), the user's data (labeled *medium*) is protected not only from malicious writes by the attachment, but also from malicious reads. The attachment's process will not be able to read, write to, or delete the user's data.

This generated solution can be generalized to prevent virtually any process resulting from executing virtually any object created by virtually any Web browser or e-mail client, or code running in these applications, from collecting sensitive data on a system running Windows Vista.

Going one step further, the concept itself can be generalized to prevent virtually any subject with a given IL α from reading any securable object with an IL $\beta > \alpha$ on a system running Windows Vista.

5.5. Concept Applicability Motivation

Concerning the requirements, the special advantages of a MAC solution like this concept are the following:

- it is transparent to the user (MAC does not require administrator or user configuration or interaction in order to work correctly);
- it provides active protection (MAC is always in effect, independently of other security policy options);
- it can be easily implemented in Windows Vista without requiring changing a lot of code (by making use of MIC);
- most importantly, it can be used to (efficiently) prevent (sensitive) information disclosure and, thus, disable spyware (MAC constrains *read* access, an ability on which spyware depends in order to collect [sensitive] data).

This concept fulfills the requirements better than the Windows Vista built-in features (namely MIC) mainly because, unlike any Windows Vista built-in feature (MIC included), it prevents information disclosure with respect to access to areas other than the virtual memory address space of a process (such as the user profile).

Chapter 6

SUMMARY AND OUTLOOK

So it can be seen that, although application compatibility and user experience is affected (uploading pictures to Web sites and copy & paste are two examples of affected common user experiences), it should be possible to use MAC to efficiently fight spyware in Windows Vista for two main reasons. First, spyware depends on the ability to read in order to collect (sensitive) data. But, most importantly, the architecture that enables this (MIC) is already implemented in this OS (Windows Vista).

Notwithstanding the compatibility and user experience issue, the concept does not hamper the ability to start programs automatically from a Web browser or an e-mail client (such as for reading various file types – e.g. .pdf files), since it does not limit *execute* access.

An interesting positive side effect resulting from the changes implemented by this concept is that not only user data files get protected from malicious reads, as also system and user data files themselves get protected from malicious writes as well.

The implementation problems are mainly related with the closed-source proprietary nature of the Microsoft Windows OS (which does not allow for direct code access, only through API's), since the concept implies the modification of the security properties of files and folders in the user profile. Another possible problem is the dependence of the concept on Web browsers and e-mail clients being specifically designed and developed to run with low rights (i.e. on their behavior being specifically tailored to maintain least privilege-functionality), since

that is a requirement for the concept to work (i.e. for proper user privacy protection to be possible).

Recalling Mark Minasi's words: *this ["no-read-up" policy] seemed like a potentially nice feature in a world beset by Web-borne spyware*. Not only is this, indeed, a "nice" feature, as it also proves to be an efficient, useful, and valuable way of protecting computer users' privacy by preventing sensitive information disclosure.

So what? Well, the reader should care about this because Microsoft Windows does not show many truly significant signs of ceasing to be the most used OS in the world anytime soon and, at the same time, Windows Vista's market share keeps growing. Most importantly, though, the reader should care about this because the threat of spyware keeps growing and MAC provides an efficient, useful, and valuable way of efficiently fighting this growing threat to computer user's privacy.

This work does suggest at least one interesting further avenue, i.e. a way in which it could be improved by future workers. That is to try to make use of UAC to intercept denied *read* attempts and prompt the user to make the final decision on whether or not these *read* accesses can take place.

GLOSSARY

386BSD/JOLIX. A free BSD UNIX OS for PC compatible computer systems based on the Intel 80386.

ACE. Access Control Entry. An entry in an ACL.

ACL. Access Control List. A list of permissions attached to an object.

AIX. Advanced Interactive Executive. The name given to a series of proprietary OS's sold by IBM for several of its computer system platforms, based on UNIX System V with 4.3BSD-compatible command and programming interface extensions.

AOL. America Online LLC.. An American global Internet services and media company operated by Time Warner, previously headquartered in Loudoun County, Virginia (until late April 2008), currently with new offices at 770 Broadway in New York City.

API. Application Programming Interface. A set of declarations of the functions (or procedures) that an OS, library, or service provides to support requests made by computer programs.^[32]

AppArmor. Application Armor. An open source application security tool for Linux systems, released under the GNU GPL.

Apple. Apple Inc. (formerly Apple Computer Inc.). An American multinational corporation with a focus on designing and manufacturing consumer electronics and closely related software products.

ASC. Anti-Spyware Coalition. A group dedicated to building a consensus about definitions and best practices in the debate surrounding spyware and other potentially unwanted technologies.

AT&T. AT&T Inc.. The largest provider of both local and long distance telephone services, wireless service, and DSL Internet access in the US.

Biba (Integrity) Model. Formal state transition system of computer security policy developed by Kenneth J. Biba in 1977^[33] that describes a set of access control rules designed to ensure data integrity.

BSD. Berkeley Software Distribution. The UNIX OS derivative developed and distributed by the Computer Systems Research Group of the University of California, Berkeley, from 1977 to 1995.

C-E. Communications-electronics. The specialized field concerned with the use of electronic devices and systems for the acquisition or acceptance, processing, storage, display, analysis, protection, disposition, and transfer of information.

CAPP. Controlled Access Protection Profile. Specifies a set of security functional and assurance requirements for IT products.

CC / Common Criteria. Common Criteria for International Technology Security Evaluation. An international standard (ISO/IEC 15408) for computer security.

chroot. An operation on UNIX OS's that changes the apparent disk root directory for the current running process and its children.

CI. Container Inherit. A flag in Windows NT-based OS's that indicates that subordinate folders will inherit the ACE.

Classified information. Sensitive information to which access is restricted by law or regulation to particular classes of people.

CLR. (Security) Clearance. MAC security level assigned to a subject, typically representing its trustworthiness, and granting it access to (classified) information (e.g. state secrets).

CLS. (Security) Classification. MAC security level assigned to an object, typically representing the level of sensitivity of the information that it contains.

COM. Component Object Model. A Microsoft-centric interface standard for software componentry introduced by Microsoft in 1993.

Confidentiality. Ensuring that information can be read only by those authorized to read it.

Consumer Reports. An American magazine published monthly by Consumers Union that publishes reviews and comparisons of consumer products and services based on reporting and results from its in-house testing laboratory.

CPU. Central Processing Unit (or simply “processor”). A description of a class of logic machines that can execute computer programs.

DAC. Discretionary Access Control. A kind of access control defined by the TCSEC^[11] as a means of restricting access to objects based on the identity of subjects and/or groups to which they belong.

DACL. Discretionary Access Control List. An ACL that is controlled by the owner of an object and that specifies the access particular users or groups can have to the object.

Darwin. An open source UNIX computer OS released by Apple in 2000.

Debian. A computer OS composed entirely of software which is both free and open source (FOSS).

DLL. Dynamic-Link Library. Microsoft’s implementation of the shared library concept in the Microsoft Windows and OS/2 OS’s.

DoD/DOD. United States Department of Defense. The federal department charged with coordinating and supervising all agencies and functions of the government relating directly to national security and the military in the USA.

DTE. Domain and Type Enforcement. Enhanced version of TE.

DTE UNIX. DTE-enabled UNIX prototype system.

EAL. Evaluation Assurance Level. A numerical grade assigned following the completion of a CC security evaluation, an international standard in effect since 1999.

Fedora. An RPM-based, general-purpose Linux distribution, developed by the community-supported Fedora Project and supported by Red Hat.

FLASK. Flux Advanced Security Kernel. An OS security architecture that provides flexible support for security policies.

Fluke. An OS (kernel and OS architecture) whose structure relies on the properties specified by the Fluke environment specification.

FreeBSD. Free Berkeley Software Distribution. A Unix-like free OS descended from AT&T UNIX via the Berkeley Software Distribution (BSD) branch through the 386BSD and 4.4BSD OS's.

Gentoo Linux. A Linux distribution OS based on the Portage package management system.

GNU GPL. GNU General Public License. A widely used free software license, originally written by Richard Stallman for the GNU Project.

GUI. Graphical User Interface. A type of user interface which allows people to interact with a computer and computer-controlled devices like computers, hand-held devices (MP3 players, Portable Media Players, gaming devices), household appliances, and office equipment..

Honeypot. A trap set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems.

IA. Information Assurance. The practice of managing information-related risks.

ID. Identifier. An identifier of something.

IE. (Windows) Internet Explorer (formerly Microsoft Internet Explorer [MSIE]). A series of graphical Web browsers developed by Microsoft, and included as part of the Microsoft Windows line of OS's, starting in 1995.

IEEE. Institute of Electrical and Electronics Engineers. An international non-profit, professional organization for the advancement of technology related to electricity.

IEPM. Internet Explorer Protected Mode. Security feature in IE 7 for Windows Vista designed to help protect users from attack by running an IE process with restricted privileges on Windows Vista.

IL. Integrity Level. A representation of the trustworthiness of running application processes and objects in Windows Vista, such as files created by an application.

Integrity. Ensuring that information can be manipulated only by those authorized to manipulate it.

IO. Inherit Only. A flag in MIC that indicates that the ACE does not apply to the current object.

IPC. Inter-Process Communication. A set of techniques for the exchange of data among two or more threads in one or more processes.

ISO. International Organization for Standardization. An international-standard-setting body composed of representatives from various national standards organizations.

ITSEC. Information Technology Security Evaluation Criteria. A structured set of criteria for evaluating computer security within products and systems.

Java. A programming language originally developed by Sun, and released in 1995 as a core component of Sun's Java platform.

JFFS2. Journaling Flash File System version 2. A log-structured file system for use in flash memory devices.

Label. A security attribute which can be applied to files, directories, or other items in the system. It can be considered a confidentiality stamp; when a label is placed on a file, it describes the security properties for that specific file, and will only permit access by files, users, resources, etc., with a similar security setting. The meaning and interpretation of label values depends on the policy configuration: while some policies treat a label as representing the integrity or secrecy of an object, other policies use labels to hold rules for access.

(Principle of) Least/Minimal privilege. A principle that requires that, in a particular abstraction layer of a computing environment, every module (such as a process, a user, or a program, on the basis of the layer that we are considering) must be able to access only such information and resources that are necessary to its legitimate purpose. [\[34\]\[35\]](#)

Level. The increased or decreased setting of a security attribute. As the level increases, its security is considered to elevate as well.

Linux. The name usually given to any UNIX-like computer OS that uses the Linux kernel.

LoMAC. A Linux extension that implements a low-watermark security policy based on the Biba Model.

Low watermark. A dynamic policy extension to the Biba Model which permits lowering of the security levels of subjects as subjects open less secure objects for

the purpose of accessing information which is less secure. In most cases, the original security level of the user is restored after the process is complete.

LSM. Linux Security Modules. A framework that allows the Linux kernel to support a variety of computer security models while avoiding favoritism toward any single security implementation.

LSPP. Labeled Security Protection Profile. A PP within the CC that represents a set of security functional and assurance requirements for IT products.

MAC. Mandatory Access Control. A type of access control by which the OS constrains the ability of a subject or initiator to access or generally perform some sort of operation on an object or target.

MAC OS X. Macintosh Operating System 10. A line of graphical OS's developed, marketed, and sold by Apple, the latest of which is pre-loaded on all currently shipping Macintosh computers.

MIC. Mandatory Integrity Control. A security feature in Windows Vista and Windows Server 2008 (and a core component of the Windows security architecture) that restricts the access permissions of applications that are running under the same user account and that are less trustworthy.

Microsoft. Microsoft Corporation. An American multinational computer technology corporation.

Microsoft Windows. The name of several families of software OS's by Microsoft.

MLS. Multi-Level Security. The application of a computer system to process information with different sensitivities (i.e. at different security levels), permit simultaneous access by users with different security CLR's and needs-to-know, and prevent users from obtaining access to information for which they lack authorization.

Novell. Novell Inc.. An American software corporation specializing in network OS's such as Novell NetWare and SUSE Linux, secure identity management products, and application integration and collaboration solutions.

NSA/CSS. National Security Agency / Central Security Service. A cryptologic intelligence agency of the US government, administered under the US DoD.

(System) Object. An entity (an allocated region of storage) through which information flows under the direction of a *subject*. This includes directories, files, fields, screens, keyboards, memory, magnetic storage, printers or any other data storage/moving device. Basically, an object is a data container or a system resource; access to an *object* effectively means access to the data..

OI. Object Inherit. A flag in Windows NT-based OS's that indicates that subordinate files will inherit the ACE.

openSUSE. A community project, sponsored by Novell and AMD, to develop and maintain a general purpose Linux distribution.

OS / O/S. Operating System. The software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer.

PC. Personal Computer. A computer whose original sales price, size, and capabilities make it useful for individuals, and intended to be operated directly by an end user, with no intervening computer operator.

PP. Protection Profile. A document used as part of the certification process according to the CC.

RBAC. Role-Based Access Control. [BGI\[37\]](#) An approach to restricting system access to authorized users.

RBACPP. Role-Based Access Control Protection Profile. Specifies a set of security functional and assurance requirements for IT products.

Red Hat. A company dedicated to free and open source software, and a major Linux distribution vendor.

RHEL. Red Hat Enterprise Linux. A Linux distribution produced by Red Hat and targeted toward the commercial market, including mainframes.

RID. Relative Identifier. The part of a SID that uniquely identifies an account or group within a domain in the context of the Windows NT line of computer OS's.

RPC. Remote Procedure Call. A technology that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

RSBAC. Rule Set Based Access Control. An open source access control framework for current Linux kernels, which has been in stable production since January 2000 (version 1.0.9a).

SACL. System Access Control List. An ACL in Windows NT-based OS's that controls the generation of audit messages for attempts to access a securable object.

SDDL. Security Descriptor Definition Language. Defines the string format that the *ConvertSecurityDescriptorToStringSecurityDescriptor* and *ConvertStringSecurityDescriptorToSecurityDescriptor* functions use to describe a SID as a text string in Windows NT-based OS's.

SELinux. Security-Enhanced Linux. A reference implementation of the FLASK security architecture for flexible MAC.

Sensitive information. Information or knowledge that might result in loss of an advantage or level of security if revealed (disclosed) to others who might have low or unknown trustworthiness and/or indeterminable or hostile intentions.

SID. Security Identifier. A unique name (an alphanumeric character string) in Windows NT-based OS's that is assigned by a Windows Domain controller during the log on process that is used to identify an object, such as a user or a group of users in a network of Windows NT/2000 systems.

Solaris. Solaris Operating System. A free UNIX-based OS introduced by Sun in 1992 as the successor to SunOS.

SRM. Security Reference Monitor. A tamperproof, always-invoked, and small-enough-to-be-fully-tested-and-analyzed security module in Windows NT-based OS's that controls all software access to data objects or devices (verifiable).

Sun. Sun Microsystems, Inc.. A multinational vendor of computers, computer components, computer software, and IT services, founded on 24 February 1982.^[38]

SUSE Linux. A major retail Linux distribution, produced in Germany, and owned by Novell.

TCP. Transmission Control Protocol. One of the core protocols of the Internet protocol suite.

TCSEC. Trusted Computer System Evaluation Criteria. A US Government DoD standard that sets basic requirements for assessing the effectiveness of computer security controls built into a computer system.

TE. Type Enforcement. Table-oriented MAC mechanism/model, well-suited for confining applications and restricting information flows.

TrustedBSD. Trusted Berkeley Software Distribution. Project that provides a set of trusted OS extensions to FreeBSD.

Trusted Solaris. A security-evaluated OS based on Solaris by Sun, featuring MAC.

UAC. User Account Control. A technology and security infrastructure introduced with Microsoft's Windows Vista OS.

Ubuntu. A computer OS that has consistently been rated among the most popular of the many Linux distributions.^{[\[39\]](#)[\[40\]](#)[\[41\]](#)[\[42\]](#)}

UDP. User Datagram Protocol. One of the core protocols of the Internet protocol suite.

UIPI. User Interface Privilege Isolation. A technology introduced in Windows Vista and Windows Server 2008 to combat code injection exploits.

Unisys. Unisys Corporation. A global provider of information technology services and solutions, based in Blue Bell, Pennsylvania, US, and incorporated in Delaware.^{[\[43\]](#)}

UNIX. A computer OS originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, and Douglas McIlroy.

UNIX-like. An OS that behaves in a manner similar to a UNIX system, while not necessarily conforming to or being certified to any version of the Single UNIX Specification.

Windows Vista. A line of OS's developed by Microsoft for use on PCs, including home and business desktops, laptops, tablet PCs, and media center PCs.

REFERENCES

1. ^^{a b} Anti-Spyware Coalition Definitions Document, June 29, 2006.
<http://www.antispywarecoalition.org/documents/DefinitionsJune292006.htm>
2. ^^{a b c d} Webroot | Spyware Basics.
http://www.webroot.com/En_US/csc/online-spyware-basics.html
3. ^ AOL/NCSA ONLINE SAFETY STUDY. *America Online & The National Cyber Security Alliance*. 2005.
http://www.staysafeonline.info/pdf/safety_study_2005.pdf
4. ^ Consumer Reports, September 2006.
http://www.webroot.com/En_US/csc/online-spyware-basics.html
5. ^^{a b c} Consumer Reports, September 2007.
http://www.consumerreports.org/cro/electronics-computers/computers/internet-and-other-services/net-threats-9-07/spyware/0709_net_spy.htm
6. ^^{a b} W3Counter global Web stats for March 31, 2008.
<http://w3counter.com/globalstats.php?date=2008-03-31>
7. ^ Net Applications OS market share for March 2008.
<http://marketshare.hitslink.com/report.aspx?qprid=8>
8. ^^{a b c} OneStat press release, April 1, 2008.
http://www.onestat.com/html/aboutus_pressbox58-microsoft-windows-vista-global-usage-share.html
9. ^^{a b} XITiMonitor report, January 21, 2008.
<http://www.xitimonitor.com/en-us/internet-users-equipment/operating-systems-december-2007/index-1-2-7-116.html>
10. ^ Yixin Jiang, Chuang Lin, Hao Yin, Zhangxi Tan. **SECURITY ANALYSIS OF MANDATORY ACCESS CONTROL MODEL**. Department of Computer Science and Technology, Tsinghua University. 2004.
11. ^^{a b c d} (December 1985) **TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA**. United States Department of Defense. DoD Standard 5200.28-STD.
<http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html>
12. ^ Webroot | Spyware Protection and Prevention.
http://www.webroot.com/En_US/csc/spyware-protection.html
13. ^^{a b} Webroot Software, March 2007.
http://www.webroot.com/En_US/csc/online-spyware-basics.html
14. ^ Net Applications OS market share trend.
<http://marketshare.hitslink.com/report.aspx?qprid=9>
15. ^ Net Applications OS versions market share trend.

- <http://marketshare.hitslink.com/report.aspx?qpid=11>
16. ^ **A GUIDE TO UNDERSTANDING SECURITY MODELLING IN TRUSTED SYSTEMS**, National Computer Security Center, Report NCSC-TG-010, USA, 1992.
 17. ^ Loscocco, Peter A.; Smalley, Stephen D.; Muckelbauer, Patrick A.; Taylor, Ruth C.; Turner, S. Jeff; Farrell, John F.. **THE INEVITABILITY OF FAILURE: THE FLAWED ASSUMPTION OF SECURITY IN MODERN COMPUTING ENVIRONMENTS**. NSA. Retrieved on 2008-03-15. <http://csrc.nist.gov/nissc/1998/proceedings/paperF1.pdf>
 18. ^ Loscocco, Peter A.; Smalley, Stephen D.. **MEETING CRITICAL SECURITY OBJECTIVES WITH SECURITY-ENHANCED LINUX**. Retrieved on 2008-03-15. <http://www.nsa.gov/selinux/papers/ottawa01.pdf>
 19. ^ Bell, D. Elliott and LaPadula, Leonard J. (1973). **SECURE COMPUTER SYSTEMS: MATHEMATICAL FOUNDATIONS**. MITRE Corporation. <http://www.albany.edu/acc/courses/ia/classics/bellapadula1.pdf>
 20. ^ Bell, D. Elliott and LaPadula, Leonard J. (1976). **SECURE COMPUTER SYSTEMS: UNIFIED EXPOSITION AND MULTICS INTERPRETATION**. MITRE Corporation. <http://csrc.nist.gov/publications/history/bell76.pdf>
 21. ^ Bell, David (December 2005). Looking Back at the Bell-La Padula Model. *Proc. 21st Annual Computer Security Applications Conference*: 337-351. doi:10.1109/CSAC.2005.37. <http://dx.doi.org/10.1109/CSAC.2005.37>; <http://www.selfless-security.org/presentations/lookingback/looking-back.html> (slides for the talk)
 22. ^ S. Castano, M. Fugini, G. Martella, and P. Samarati. **DATABASE SECURITY**. Addison-Wesley, 1994.
 23. ^ Biba, K. J. **INTEGRITY CONSIDERATIONS FOR SECURE COMPUTER SYSTEMS**, MTR-3153, The Mitre Corporation, April 1977.
 24. ^ Dorothy E. Denning, **A LATTICE MODEL OF SECURE INFORMATION FLOW**, *Commun. ACM*, v.19 n.5, 1976.
 25. ^ Ravi S. Sandhu, **LATTICE-BASED ACCESS CONTROL MODELS**, *Computer*, v.26 n.11, p.9-19, November 1993.
 26. ^ Dr. David F.C. Brewer, Dr. Michael J. Nash: **THE CHINESE WALL SECURITY POLICY** (PDF) Gamma Secure Systems Limited. 1989. Retrieved on 2008-03-13. http://www.cs.purdue.edu/homes/ninghui/readings/AccessControl/brewer_nash_89.pdf
 27. ^ David D. Clark, David R. Wilson. **A COMPARISON OF COMMERCIAL AND MILITARY COMPUTER SECURITY POLICIES**. 1987.

28. [^](#) Ross J. Anderson. **A SECURITY POLICY MODEL FOR CLINICAL INFORMATION SYSTEMS** (PDF). University of Cambridge Computer Laboratory. 1996. Retrieved on 2008-03-13.
<http://www.cl.cam.ac.uk/~rja14/Papers/oakpolicy.pdf>
29. [^](#) National Information Assurance Partnership. **THE COMMON CRITERIA EVALUATION AND VALIDATION SCHEME VALIDATED PRODUCTS LIST**. Retrieved on 2008-03-15.
<http://www.niap-ccevs.org/cc-scheme/vpl/>
30. [^](#) TrustedBSD Project. TrustedBSD Mandatory Access Control (MAC) Framework. Retrieved on 2008-03-15.
<http://www.trustedbsd.org/mac.html>
31. [^](#) sandbox_init(3) man page (2007-07-07). Retrieved on 2008-03-15.
http://developer.apple.com/DOCUMENTATION/Darwin/Reference/ManPages/man3/sandbox_init.3.html
32. [^](#) **QUICKSTUDY: APPLICATION PROGRAMMING INTERFACE (SAPI)**. ComputerWorld.
<http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=43487>
33. [^](#) Biba, K. J. **INTEGRITY CONSIDERATIONS FOR SECURE COMPUTER SYSTEMS**, MTR-3153, The Mitre Corporation, April 1977.
34. [^](#) Saltzer 75.
35. [^](#) Denning 76.
36. [^](#) Ferraiolo, D.F. and Kuhn, D.R. (October 1992). **ROLE BASED ACCESS CONTROL** (PDF). *15th National Computer Security Conference*. 554-563.
<http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf>
37. [^](#) Sandhu, R., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (August 1996). **ROLE-BASED ACCESS CONTROL MODELS** (PDF). *IEEE Computer* 29 (2): 38-47. IEEE Press.
<http://csrc.nist.gov/rbac/sandhu96.pdf>
38. [^](#) **THE GLAMOUR IN MASS TRANSIT**. Sun Microsystems, Inc. (24 February 2007). Retrieved on 2007-02-25.
http://blogs.sun.com/jonathan/entry/brutal_efficiency_virtualization_by_another
39. [^](#) **DISTROWATCH 2005**. distrowatch.com. Retrieved on 2008-05-13.
<http://distrowatch.com/index.php?dataspan=2005>
40. [^](#) **DISTROWATCH 2006**. distrowatch.com. Retrieved on 2008-05-13.
<http://distrowatch.com/index.php?dataspan=2005>
41. [^](#) **DISTROWATCH 2005**. distrowatch.com. Retrieved on 2008-05-13.
<http://distrowatch.com/index.php?dataspan=2005>
42. [^](#) **2007 DESKTOP LINUX MARKET SURVEY** (2007-08-21). Retrieved on 2007-12-27.

<http://www.desktoplinux.com/cgi-bin/survey/survey.cgi?view=archive&id=0813200712407>

43. [^ http://www.sec.gov/cgi-bin/browse-edgar?company=Unisys&CIK=&filenum=&State=&SIC=&owner=include&action=getcompany](http://www.sec.gov/cgi-bin/browse-edgar?company=Unisys&CIK=&filenum=&State=&SIC=&owner=include&action=getcompany)

BIBLIOGRAPHY

Mandatory Access Control (MAC):

- <http://www.wikipedia.org>;
- Krzysztof Juszczyszyn, **VERIFYING ENTERPRISE'S MANDATORY ACCESS CONTROL POLICIES WITH COLOURED PETRI NETS**, Wroclaw University of Technology, Wroclaw, Poland, 2003;
- Yixin Jiang, Chuang Lin, Hao Yin, Zhangxi Tan, **SECURITY ANALYSIS OF MAINDATORY ACCESS CONTROL MODEL**, Department of Computer Science and Technology, Tsinghua University, Beijing, P. R. China, 2004;
- Ninghui Li, Ziqing Mao, Hong Chen, **USABLE MANDATORY INTEGRITY PROTECTION FOR OPERATING SYSTEMS**, Center for Education and Research in Information Assurance and Security (CERIAS) and Department of Computer Science, Purdue University, 2007;
- Indrakshi Ray, Mahendra Kumar, Towards a location-based mandatory access control model, Department of Computer Science, Colorado State University, **COMPUTERS & SECURITY** 25 (2006) 36 – 44, Science Direct, Elsevier, www.sciencedirect.com, www.elsevier.com/locate/cose, 29 June 2005;

- Sylvia Osborn, **MANDATORY ACCESS CONTROL AND ROLE-BASED ACCESS CONTROL REVISITED**, Department of Computer Science, The University of Western Ontario, 1997;
- Thuong Doan, Steven Demurjian, T.C. Ting, Andreas Ketterl, **MAC AND UML FOR SECURE SOFTWARE DESIGN**, University of Connecticut, 29 October 2004.

Mandatory Integrity Control (MIC):

- msdn.microsoft.com;
- Access Control (<http://go.microsoft.com/fwlink/?LinkId=90927>);
- Access Control [Security] (<http://go.microsoft.com/fwlink/?LinkId=90928>);
- The COM Elevation Moniker (<http://go.microsoft.com/fwlink/?LinkId=90939>);
- Biba Integrity Model: Biba, K. J. Integrity Considerations for Secure Computer Systems, Technical Report MTR-3153, MITRE Corporation, Bedford, Massachusetts, June 1975;
- D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In IEEE Symposium on Computer Security and Privacy, April 1987;
- Driver Signing Requirements for Windows (<http://go.microsoft.com/fwlink/?LinkId=90930>);

- LOMAC Low Water-Mark Integrity Protection by Timothy Fraser
(<http://go.microsoft.com/fwlink/?LinkId=90932>);
- Process Security and Access Rights
(<http://go.microsoft.com/fwlink/?LinkId=90929>);
- Security Descriptor Definition Language (SDDL) [Security]
(<http://go.microsoft.com/fwlink/?LinkId=90935>);
- SID_IDENTIFIER_AUTHORITY Structure [Security]
(<http://go.microsoft.com/fwlink/?LinkId=90933>);
- SID Strings [Security] (<http://go.microsoft.com/fwlink/?LinkId=90936>);
- SYSTEM_MANDATORY_LABEL_ACE Structure [Security]
(<http://go.microsoft.com/fwlink/?LinkId=90934>);
- TOKEN_INFORMATION_CLASS enumeration [Security]
(<http://go.microsoft.com/fwlink/?LinkId=90938>);
- Understanding and Working in Protected Mode Internet Explorer
(<http://go.microsoft.com/fwlink/?LinkId=90931>);
- User Account Control
(<http://go.microsoft.com/fwlink/?LinkId=82373>).

Security-Enhanced Linux (SELinux):

- <http://www.wikipedia.org>;
- <http://www.nsa.gov/selinux/>.

AppArmor, FreeBSD, Trusted Solaris, Mac OS X:

- <http://www.wikipedia.org>.

Appendix A

SDDL FOR MANDATORY LABELS

The SDDL is a convenient way to represent access permissions in a string format. SDDL defines ACE strings and SID strings to represent fields of access control entries. The *ConvertSecurityDescriptorToStringSecurityDescriptor* and *ConvertStringSecurityDescriptorToSecurityDescriptor* functions can be used to convert the mandatory label ACE from a binary to a string format and back.

The definitions for SDDL strings are in the header file, *sddl.h*.

The SDDL string for a mandatory label ACE is defined as follows:

```
#define SDDL_MANDATORY_LABEL TEXT("ML") // Integrity label
The SDDL strings for the mandatory label policy flags, which
are in the access mask, are the following:
#define SDDL_NO_WRITE_UP      TEXT("NW")
#define SDDL_NO_READ_UP      TEXT("NR")
#define SDDL_NO_EXECUTE_UP   TEXT("NX")
The SDDL SID strings for the IL's are the following:
#define SDDL_ML_LOW          TEXT("LW")
#define SDDL_ML_MEDIUM      TEXT("ME")
#define SDDL_ML_HIGH        TEXT("HI")
#define SDDL_ML_SYSTEM      TEXT("SI")
```

An example of the SDDL for a mandatory label ACE in a SACL that specifies *NO_WRITE_UP* policy for *low* IL is the following: *S:(ML;;;NW;;;LW)*.

SDDL strings can be used with the *ConvertStringSecurityDescriptorToSecurityDescriptor* function to initialize a security descriptor with an explicit mandatory label that can

be used as the security attributes parameter when creating a new object, such as a file, using *CreateFile*.¹⁰

¹⁰ When using *ConvertSecurityDescriptorToStringSecurityDescriptor*, the new security information flag *LABEL_SECURITY_INFORMATION*, must be specified, in order to convert an explicit mandatory label into the SDDL string equivalent. Without the *LABEL_SECURITY_INFORMATION* flag, a mandatory label, if it exists, will not show up in the SACL portion of the string.

Appendix B

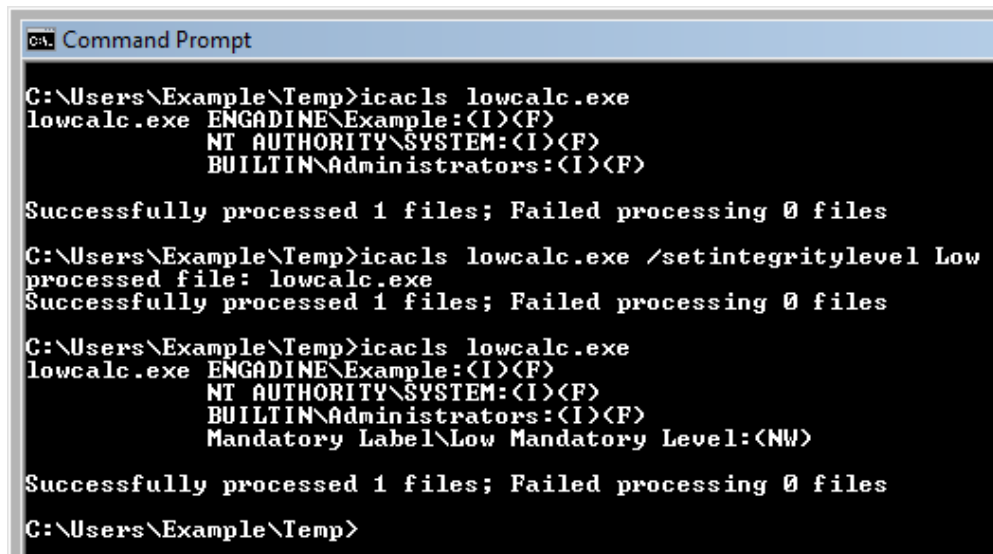
ICACLS AND FILE IL'S

Icacls is a command-line tool developed by Microsoft Corporation that can be used to manage the security settings on files. The Windows Vista version of Icacls supports mandatory labels on files.¹¹

Icacls can be used to view and set the IL for a file. Icacls displays the IL SID for a file if the file has an explicit mandatory label ACE. Icacls does not show the IL SID for the implicit default IL. Icacls uses the *NO_WRITE_UP* integrity policy only when setting the IL of a file.

The following image shows an example of using Icacls to view or set the IL of a file.

Figure 8 Icacls and mandatory labels



```
Command Prompt
C:\Users\Example\Temp>icaccls lowcalc.exe
lowcalc.exe  ENGADINE\Example:(I)<F>
              NT AUTHORITY\SYSTEM:(I)<F>
              BUILTIN\Administrators:(I)<F>

Successfully processed 1 files; Failed processing 0 files

C:\Users\Example\Temp>icaccls lowcalc.exe /setintegritylevel Low
processed file: lowcalc.exe
Successfully processed 1 files; Failed processing 0 files

C:\Users\Example\Temp>icaccls lowcalc.exe
lowcalc.exe  ENGADINE\Example:(I)<F>
              NT AUTHORITY\SYSTEM:(I)<F>
              BUILTIN\Administrators:(I)<F>
              Mandatory Label\Low Mandatory Level:(NW)

Successfully processed 1 files; Failed processing 0 files

C:\Users\Example\Temp>
```

¹¹ *icaccls.exe* is an update to an older program, *cacls.exe*. *Cacls.exe* does not recognize mandatory labels.

A p p e n d i x C

CHML AND FILE IL'S

Chml is a command-line tool developed by Mark Minasi that (as Icacls) can be used to modify IL's.

C.1. Features

Chml's main features are:

- installs simply and with a minimum of trouble (chml is, therefore, a command-line tool that is just a simple *EXE* file, with no setup program required);
- allows for viewing a file or folder's IL;
- allows for changing a file or folder's IL;
- allows for experimenting with extensions of the basic MIC and going beyond the standard “no-write-up” policy (no *low*-integrity process can *modify* a higher-integrity object) to the implemented but seldom used “no-read-up” policy, which blocks any attempts by a lower-integrity process or user to *read* the object.

C.2. Download

Chml can be downloaded from www.minasi.com/vista/chml.exe. It can be saved somewhere on the Windows path. If stored in the `\windows\system32` folder, chml can be used simply by opening a command prompt window and typing `chml`.

C.3. Preparation

Chml can be used “right out-of-the-box” to view a file or folder’s IL by typing `chml fileorfolder`, as in `C:\>chml \windows\notepad.exe`.

To modify an object’s IL, the user account must be given a new-to-Vista permission, “Modify an object label”. It can be found in the User Rights part of Group Policy on a Vista machine:

1. open `gpedit.msc`;
2. navigate to Computer Configuration / Windows Settings / Local Policies / User Rights Assignment;
3. open “Modify an object label” (right-hand pane);
4. add user account (by default, there are no user accounts listing with this privilege);
5. close Group Policy Editor;
6. log off, then back on to finish getting new privilege on logon token.

Now an elevated command prompt must be opened (right-click “Command Prompt” icon in Accessories, then choose “Run as administrator”) and `chml` is ready to be used. There is online help, but here are few quick-starts.

C.4. Modifying an IL

To see `chml`’s basic powers in action, a folder is created. In this example, one named `c:\test` is created. Then it is set to *low* integrity by typing:

```
chml c:\test -i:l
```

The run looks something like this:

```
C:\>chml c:\test -i:l

chml v1.010 -- Change windows Integrity Level
by Mark Minasi (c) 2006 www.minasi.com
"chml -?" for syntax, examples and notes.

Integrity level of c:\test successfully set to low.

C:\>
```

The syntax is simple: `chml` is followed with the name of the folder, followed by a lowercase `i`, a colon, and then one of the letters `u`, `l`, `m`, `h`, or `s`, which signify *untrusted*, *low*, *medium*, *high*, or *system*.¹² Now, a file named `testfile.txt` is created inside `c:\test` – it does not matter what text is in it, or if there is any text there. Then `chml` is asked what integrity label the file has:

```
C:\>echo Hi there>\test\testfile.txt

C:\>chml c:\test\testfile.txt -b
c:\test\testfile.txt's mandatory integrity level=low

C:\>
```

What is being seen here is that, just as objects in folders can inherit permissions from their parent folders, they can also inherit IL's.

C.5. Seeing the Effect of “No Write Up”

As discussed before, MIC's main value is to keep lower-integrity processes from modifying higher-integrity objects. This will be demonstrated by setting a file's IL to *high*, then opening up a non-elevated command prompt (which will run at *medium*) and see that the file cannot be erased.

¹² If the copyright and help banner are to be skipped, the `-b` option should be added.

From the elevated command prompt, `c:\test` is raised to *high* integrity:

```
C:\>chml c:\test -i:h -b
Integrity level of c:\test successfully set to high.
C:\>
```

A second command prompt is opened, but not elevated. If `c:\test\testfile.txt` is tried to be erased, an “Access is denied” error will be received.

C.6. Setting Something to “No Read Up”

Now the “no-read-up” policy will be tried out. As discussed before, there are actually 3 MIC policies: *no read up*, *no write up*, and *no execute up*, and any combination of them can be applied to any object using the `-nr`, `-nw` and/or `-nx` switches. From the elevated command prompt, `c:\testfile`’s policy is set to “no read up / no write up”:

```
C:\>chml c:\test -i:h -nr -nx -b
Integrity level of c:\test successfully set to high.
C:\>
```

Now, if the file or folder is, at all, tried to be examined from the non-elevated command prompt, or even from Explorer, for that matter, again an “Access is denied” will be seen. The reason why an IL of *high* had to, once again, be assigned is that integrity labels are simple: each object only gets 1, so, to modify one, it is not enough to just specify changes, it is necessary to redefine the whole integrity label.

C.7. Setting an IL to *System*

As discussed before, MIC only lets a user elevate IL's to be equal to the user's IL. Many OS processes are not explicitly labeled as *system*, but, when they run, they run at a *system* level. As *system* is of higher trustworthiness than *high*, that implies that, if someone were to create a file and somehow manage to label it as *system*, then no administrator could delete it.

But there are ways to elevate an account to *system*. Armed with that power, one can be sure that, if malware that marks itself as *system* ever appears, there are strategies for lowering its IL back down to one that will allow a user to delete it. Here are 3:

- run scheduled chml task as *system*,
- use `psexec -s` to run chml as *system* (psexec can be found at the SysInternals Web site: www.SysInternals.com);
- boot WinPE – which logs a user on as *system* – and run chml.

In every case, the `-i:s` option must be used, which is available in chml. For example, to set `c:\test\testfile.txt` using psexec, psexec would be downloaded and put in a user's system path, and the following command would be typed:

```
C:\>psexec -s chml c:\test\testfile.txt -i:s -b

PsExec v1.73 - Execute processes remotely
Copyright (C) 2001-2006 Mark Russinovich
Sysinternals - www.sysinternals.com

Integrity level of c:\test\testfile.txt successfully set to
system.

chml exited on ENT64 with error code 0.

C:\>
```

C.8. Chml Vs. Icacls

Chml can do things that Icacls cannot:

- set IL's at *untrusted* or *system*;
- assign “no-read-up” or “no-execute-up” integrity policies;
- view the “raw” integrity control label (chml's `-sddl` option enables this);
- create and apply a hand-crafted “raw” integrity control label (chml's `-wsddl` option enables this).

The author of this command-line tool (Mark Minasi) can be contacted at help@minasi.com.

Appendix D

MIC AND WINDOWS KERNEL MODE CODE INTEGRITY

Windows Vista enforces a requirement on 64-bit platforms that all kernel mode binaries must be digitally signed. Verifying the digital signature ensures that the integrity of the binary file has not been tampered with, because the signature was applied when the image was created. For more information on the Windows driver signing requirements, see *Driver Signing Requirements for Windows* (<http://go.microsoft.com/fwlink/?LinkId=90930>). The Windows Vista component implementing kernel mode binary image signature verification is known as Code Integrity. Code Integrity verifies the integrity of kernel-mode binary files as the image is loaded into memory by the OS loader (*Winload.exe*) or the kernel. Code Integrity is not part of MIC for access control described in this thesis.

Appendix E

GETTING THE IL FOR AN ACCESS TOKEN

The *GetTokenInformation* API can be used to retrieve the access token IL from the access token. *GetTokenInformation* has a parameter to indicate what access token information class to retrieve. The *TOKEN_INFORMATION_CLASS* parameter has a defined value for the IL, *TokenIntegrityLevel*. *GetTokenInformation* returns a *TOKEN_MANDATORY_LABEL* data structure.

E.1. To Determine the IL of a Process

1. Open handle to access token of current process.
2. Get the IL of access token.
3. Compare IL SID to system-defined IL RID's.

The following code sample shows how to do this.

```
void showProcessIntegrityLevel()
{
    HANDLE hToken;
    HANDLE hProcess;

    DWORD dwLengthNeeded;
    DWORD dwError = ERROR_SUCCESS;

    PTOKEN_MANDATORY_LABEL pTIL = NULL;
    LPWSTR pStringSid;
    DWORD dwIntegrityLevel;

    hProcess = GetCurrentProcess();
    if (OpenProcessToken(hProcess, TOKEN_QUERY, &hToken))
    {
```

```

// Get the IL.
if (!GetTokenInformation(hToken, TokenIntegrityLevel,
    NULL, 0, &dwLengthNeeded))
{
    dwError = GetLastError();
    if (dwError == ERROR_INSUFFICIENT_BUFFER)
    {
        pTIL = (PTOKEN_MANDATORY_LABEL)LocalAlloc(0,
            dwLengthNeeded);
        if (pTIL != NULL)
        {
            if (GetTokenInformation(hToken, TokenIntegrityLevel,
                pTIL, dwLengthNeeded, &dwLengthNeeded))
            {
                dwIntegrityLevel = *GetSID'subAuthority(pTIL->Label.Sid,
                    (DWORD)(UCHAR)(*GetSID'subAuthorityCount(pTIL->Label.Sid)-
                    1));

                if (dwIntegrityLevel == SECURITY_MANDATORY_LOW_RID)
                {
                    // Low Integrity
                    wprintf(L"Low Process");
                }
                else if(dwIntegrityLevel>=SECURITY_MANDATORY_MEDIUM_RID
                    && dwIntegrityLevel<SECURITY_MANDATORY_HIGH_RID)
                {
                    // Medium Integrity
                    wprintf(L"Medium Process");
                }
                else if (dwIntegrityLevel>=SECURITY_MANDATORY_HIGH_RID)
                {
                    // High Integrity
                    wprintf(L"High Integrity Process");
                }
                else if (dwIntegrityLevel >=
                    SECURITY_MANDATORY_SYSTEM_RID)
                {
                    // System Integrity
                    wprintf(L"System Integrity Process");
                }
            }
        }
    }
}

```

```
    }  
    LocalFree(pTIL);  
  }  
}  
CloseHandle(hToken);  
}  
}
```