FOI-R--1518--SE
December 2004
1650-1942

Technical report

Intrusion Analysis in Military Networks – File Systems and Logging

Arne Vidström
Mats Persson
Martin Karresand

Command and Control Systems
Box 1165
SE-581 11 LINKÖPING
Sweden

Swedish Defence Research Agency
Command and Control Systems
Box 1165
SE-581 11 LINKÖPING
Sweden

FOI-R--1518--SE
December 2004
1650-1942

Technical report

Intrusion Analysis in Military Networks – File Systems and Logging

Arne Vidström
Mats Persson
Martin Karresand

Issuing organization	Report number, ISRN	Report type
Swedish Defence Research Agency Command and Control Systems Box 1165 SE-581 11 LINKÖPING Sweden	FOI-R--1518--SE	Technical report
	Programme Areas: C4ISTAR	
	Month year: December 2004	Project no.: E7091
	General Research Areas: Commissioned Research	
	Subcategories: C4I	

Author/s (editor/s)	
Arne Vidström Mats Persson Martin Karresand	Project manager: Mikael Wedlin Approved by: Johan Allgurén Sponsoring agency: Swedish Armed Forces Scientifically and technically responsible

Report title

Intrusion Analysis in Military Networks– File Systems and Logging

Abstract

This report presents a study of the technical aspects of four file systems, NTFS, FAT32, Ext2, and Ext3. Their structure on disk and organization of data, files, and directories is described at a level enabling further research of the field. This report does, however, not describe how writing, changing, and deleting files is done in the respective file system.
Apart from file systems the report also covers the basics of logging and different tools for doing system integrity checking. The report is concluded with a chapter presenting suggested future work, sprung from the file system and logging studies.

Keywords

Further bibliographic information	Language
	English

ISSN	Pages
1650-1942	36

Distribution	
By sendlist	Price Acc. to pricelist Security classification: Unclassified

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets forskningsinstitut Ledningssystem Box 1165 SE-581 11 LINKÖPING Sweden	FOI-R--1518--SE	Teknisk rapport
	Forskningsområde: Ledning, informationsteknik och sensorer	
	Månad, år: December 2004	Projektnummer: E7091
	Verksamhetsgren: Uppdragsfinansierad verksamhet	
	Delområde: Ledning med samband, telekom och IT-system	

Författare/redaktör	Projektledare
Arne Vidström Mats Persson Martin Karresand	Mikael Wedlin
	Godkänd av: Johan Allgurén
	Uppdragsgivare/kundbeteckning: FM
	Tekniskt och/eller vetenskapligt ansvarig

Rapportens titel

Intrångsanalys i militära nätverk– filsystem och loggning

Sammanfattning

Den här rapporten presenterar en studie av de tekniska aspekterna hos fyra filsystem, NTFS, FAT32, Ext2 och Ext3. Dessa filsystems struktur på hårddisken beskrivs på en nivå som ska möjliggöra fortsatta studier inom området. Dessutom beskrivs filsystemens organisation av data, filer och kataloger. Rapporten täcker dock inte hur skrivning, ändring och radering av data går till.

Förutom filsystem beskriver rapporten även grunderna för loggning ur olika aspekter och verktyg som kan användas för riktighetsverifiering av system. Avslutningsvis presenteras förslag på framtida arbete, sprungna ur filsystems och loggningsstudierna.

Nyckelord

Övriga bibliografiska uppgifter	Språk
	Engelska
ISSN: 1650-1942	Antal sidor: 36
Distribution: Enligt missiv	Pris: Enligt prislista Sekretess: Öppen

Contents

List of Tables

1. Introduction

When trying to catch a spy one very important aspect is to actually get hold of the real traces left by him or her, not the smoke screens and false tracks placed by the spy. The deceptive information that conceal the true facts from the analyst has to be cut away to let him get to the raw data. Performing an analysis is however as laying a puzzle without knowing what it will look like in the end, and the pieces are often hidden, or even deleted.

Information hiding and deletion is a real problem for any type of analyst trying to uncover the truth. The problem might be especially evident in the IT world, where everything concerns information. A computer intrusion analyst consequently needs to fully master the interaction between the physical hardware in a computer and the software placed on top. The deeper the analyst can dig the more traces may be found.

At one of the lower levels in a computer the file system is found. It is one of the links between the hardware and the OS (operating system) and as such will contain a lot of traces of information that has been present in the system. Hence it is a good place to look for signs of intrusions.

Looking for traces in only one place is however not enough. All information there are will be needed. The system logs, originally implemented to enable easier debugging, are part of the layer above the file system. Hence they also are part of the puzzle the analyst has to fit together, piece by piece.

To be able to fully utilize the different possibilities for finding all the pieces of the puzzle from a system at hand, a good manual is always a help. This report is meant to be one piece in that puzzle, by providing the technical foundations of file systems and logging functions.

1.1 Background

This report is one of the reports of the first year in the three year project named "Warfare in the IT-domain"[1], financed by the Swedish Armed Forces. The project aims at answering the following questions:

- How is a successful battle to be fought when using software based IT-weapons?
- What type of conventional weapons are needed in addition to the software based to win a battle in the IT domain?
- What types of IT related threats are there?
- What methods to use for IT surveillance and intrusion analysis?

[1] "Strid i IT-domänen" in Swedish.

- What is the current state-of-the-art within the area?

The main focus of the report is on the technical aspects of computer file systems and the logging facilities used in computer systems. A presentation of some tools for system integrity checking is also included.

The report is tightly connected to the report "Intrusion Analysis in Military Networks – An Introduction" [1]. They will, together with the other reports yet to be written in the project, form a suite covering different aspects of the intrusion analysis field, especially aimed at military networks.

1.2 Purpose

The report is meant to work as a foundation to build future research on, as it presents the basics of logging and storage of logs. By having that knowledge at hand, the higher dimensions of intrusion analysis can also be understood. In the long run that knowledge and the knowledge built on that can be used for building better intrusion detection, intrusion prevention, intrusion response, and forensic tools. It may also lead to a better understanding of the computer security and its related fields in general.

This work is also meant to expose questions that need to be answered and in that way give a road map to future research within the field.

1.3 Scope

The scope of the report is the on-disk structure of file systems, limited to four common operating systems. These are NTFS (New Technology File System), FAT32, Ext2 (The Second Extended File System), and Ext3 (The Third Extended File System). The two first are used by Windows and the latter ones are used in the Linux operating system. The report does not cover file creation, deletion and changing. The technical level is meant to be deep enough to enable further research in the area, yet general enough as not to be restricted to one specific patch level or implementation of the file systems.

The part covering logging does introduce the basics of some logging facilities in different environments. In addition to that some common tools for system integrity checking are introduced. The selection of tools is not meant to be exhaustive.

1.4 Structure

The report is structured into 4 chapters. The first chapter introduces the background and formalia concerning the report. Chapter 2 deals with the technical aspects of file systems and their structure. It also presents different logging facilities at hand and some tools for performing system integrity checking. In chapter 3 some remaining research issues and unanswered questions are presented as well as future work related to the project. In the last chapter, number 4, the conclusions drawn from the material in the report can be found.

2. File systems, logging, and integrity checking

This is the fundamental part of the report and in this chapter the main types of file systems will be presented. They are taken from both Windows and Linux, but all of them represent the x86 architecture of the PC world. What is presented is only the technical layout of the file systems, how writing and deletion is done is not covered.

The chapter does also present an overview of logging from different points of view. There is also a small part on system integrity checking.

2.1 File system

This section will present the NTFS (New Technology File System) and FAT32 file systems from the Windows side of the spectrum. From the Linux side the Ext2 (The Second Extended File System) and Ext3 (The Third Extended File System) file systems were chosen. There are however an ever increasing amount of file systems created on the Linux and BSD side, which might have to be covered in the future work in the project.

2.1.1 The NTFS file system NTFS (New Technology File System) is the native file system of the Microsoft Windows NT based operating systems to date. The fundamental structure of NTFS on disk has not changed much since it was first introduced. NTFS has been officially described by Microsoft at some level of detail [2] but the complete specification has not been released. Several persons have been able to contribute to a better understanding of NTFS through extensive reverse engineering. Through this work unofficial detailed documentation has been made available, although still not complete, as part of the Linux-NTFS Project [3]. The description of NTFS presented in our report is based on that documentation and verified against an NTFS partition formatted by Windows XP SP0. What is described here are those parts of NTFS which at the time of writing were considered to be important when performing intrusion analysis. As a consequence, the tables presented do not represent the complete on disk structures but only selected members at specified offsets.

The starting point when interpreting an NTFS partition is the boot sector, which is also the first sector of the partition. Table 2.1 describes the layout of the boot sector. There are a few terms in the table that need some further explanation at this point. All the clusters in a partition are numbered, starting at zero, and each one of these numbers is referred to as an *LCN* (Logical Cluster Number). The clusters belonging to a particular file are numbered in a similar way, also starting at zero. In this case the term used is *VCN* (Virtual Cluster

Table 2.1: The boot sector in NTFS

Offset	Size (B)	Description
03h	8	OEM String "NTFS" (Do not assume this value!)
0bh	2	Bytes per sector
0dh	1	Sectors per cluster
28h	8	Sectors on volume
30h	8	LCN of VCN 0 of the MFT
40h	4	Clusters per MFT Record
44h	4	Clusters per Index Record

Table 2.2: The FILE Record header

Offset	Size (B)	Description
00h	4	Magic string "FILE"
10h	2	Sequence number (Number of reuses)
12h	2	Hard link count
14h	2	Offset to the first attribute
16h	2	Flags (1 = In use, 2 = Directory)
18h	4	Real size of the FILE Record
1ch	4	Allocated size of the FILE Record
20h	8	File reference to Base FILE Record

Number). The *MFT* (Master File Table) is a table containing records for all files on the file system. *Index Records* are used to build directories. The MFT will be explained in detail next, and the Index Records will be explained in detail later.

Since the LCN of VCN 0 of the MFT can be found in the boot sector, the MFT can easily be located on the disk. For example, if the LCN is 786432 and there are 8 sectors per cluster, then the starting sector of the MFT counted from the beginning of the partition will be $786432 * 8 = 6291456$.

The MFT contains *FILE Records* for all files on the partition. There are two different kinds of FILE Record. Every file is described with at least a *Base FILE Record*. If there is not sufficient space in such a record, then *Extension FILE Records* can be used as a complement. A FILE Record consists of a header (Table 2.2), a variable number of attributes (attributes will be explained later) and an end marker (ffffffffh). The first 24 FILE Records are reserved for various purposes, and the most important ones as far as intrusion analysis is concerned are described in Table 2.3. As can be seen in the table, file system internal file names start with the character $. Attribute names also begin with the same character.

In a basic file system each file contains only one portion of data, but in NTFS a file can contain a number of completely separate portions. These portions are called *attributes* in NTFS terminology. For example, the regular data of a file is stored in the so called *unnamed $DATA attribute*. There are several types of attributes with different contents, but they all start with an *attribute header*. A selection of attributes can be seen in Table 2.4. There are two variations of the attribute header layout, depending on if the attribute

Table 2.3: Reserved positions in the MFT

Position	Filename	Description
0	$MFT	The MFT
2	$LogFile	Transaction log file
3	$Volume	Serial number, creation time, dirty flag
5	.	The root directory
6	$Bitmap	Cluster map (in-use vs. free)
8	$BadClus	Bad clusters
9	$Secure	Security descriptors (only W2K and onwards)

Table 2.4: Selected attribute types

Type	Name
00h	$STANDARD_INFORMATION
20h	$ATTRIBUTE_LIST
30h	$FILE_NAME
60h	$VOLUME_NAME
70h	$VOLUME_INFORMATION
80h	$DATA
90h	$INDEX_ROOT
a0h	$INDEX_ALLOCATION
b0h	$BITMAP

is resident or not. The data of a *resident attribute* is stored inside the FILE Record itself whereas a *non resident attribute* is stored at a separate position on the disk, pointed to from the FILE Record. Tables 2.5 and 2.6 show these two types of attributes.

The data belonging to a non-resident attribute is stored in one or more *data runs*. Each data run consists of a sequential number of clusters, so the only things needed to define it are the starting cluster and the size measured in number of clusters. More than one data run is used when attribute data is fragmented on disk. If a file is compressed or sparse, the representation in data runs will be different from the basic cases, and these advanced cases will

Table 2.5: Standard attribute header (Resident version)

Offset	Size (B)	Description
00h	4	Attribute type
04h	4	Length (Including header)
08h	1	Non-resident flag (0 = Resident, 1 = Non-resident)
09h	1	Name length (0 = Not named)
0ah	2	Offset to the name (Name is in Unicode)
0ch	2	Flags (1 = Compr. 4000h = Encr. 8000h = Sparse)
10h	4	Attribute length
14h	2	Attribute offset
16h	1	Indexed flag

Table 2.6: Standard attribute header (Non-resident version)

Offset	Size (B)	Description
00h	4	Attribute type
04h	4	Length (Including header)
08h	1	Non-resident flag (0 = Resident, 1 = Non-resident)
09h	1	Name length (0 = Not named)
0ah	2	Offset to the name (Name is in Unicode)
0ch	2	Flags (1 = Compr. 4000h = Encr. 8000h = Sparse)
10h	8	Starting VCN
18h	8	Last VCN
20h	2	Offset to data runs representation
28h	8	Allocated size of the attribute
30h	8	Real size of the attribute
38h	8	Initial data size of the stream

not be covered here.

The most simple case is an ordinary file with only one data run. The data run representation in the MFT entry starts with a single byte header. The lower nibble (half byte) tells us how many bytes following the header that are used to store the length of the data run. The higher nibble tells us how many bytes following the length that are used to store the starting LCN of the data run. When the attribute data is fragmented, more than one of these header-length-LCN combinations are used in sequence. In all the cases the data run representations are terminated by a null byte header.

Now we are ready to take a more or less complete look at how an ordinary file is stored on disk. The starting point we need is the MFT entry of the file. Usually the entry is found through the directory structures, but it can also be found through some other method. Either way, the MFT entry contains a FILE Record which in turn contains a number of attributes. For an ordinary file on a modern NTFS partition, three different attributes are present. They are $STANDARD_INFORMATION, $FILE_NAME and $DATA, in that order. The $STANDARD_INFORMATION attribute contains the information in Table 2.7 and the $FILE_NAME attribute the information in Table 2.8. It is important to remember that the offsets in these tables are relative to the Attribute offset in the Standard attribute header that each attribute starts with. The $DATA attribute is not named and simply contains the data runs of the file data. What is sometimes referred to as *multiple data streams* or *alternate data streams* is a special case when more than one $DATA attribute is present and the extra ones are named.

The DOS file permissions of the $STANDARD_INFORMATION attribute, together with the flags information in the $FILE_NAME attribute contain one of the alternatives shown in Table 2.9. The names of the alternatives indicate what attribute they belong to of file permission or flag.

It should be noted that the file name is stored in the MFT entry as well as in the directory. The information held in *filename namespace* in the $FILE_NAME attribute is given in Table 2.10. What is also very interesting is the fact that the timestamps are stored in two places in the MFT entry as

Table 2.7: $STANDARD_INFORMATION

Offset	Size (B)	Description
00h	8	C Time (File created)
08h	8	A Time (File altered)
10h	8	M Time (MFT changed)
18h	8	R Time (File read)
20h	4	DOS file permissions

Table 2.8: $FILE_NAME

Offset	Size (B)	Description
00h	8	C Time (File created)
08h	8	A Time (File altered)
10h	8	M Time (MFT changed)
18h	8	R Time (File read)
28h	8	Allocated size of the file
30h	8	Real size of the file
38h	4	Flags
40h	1	Filename length in characters
41h	1	Filename namespace
42h	2L	Filename (In Unicode)

Table 2.9: DOS File Permissions / Flags in $FILE_NAME

Flag	Description
0001h	Read-Only
0002h	Hidden
0004h	System
0020h	Archive
0040h	Device
0080h	Normal
0100h	Temporary
0200h	Sparse File
0400h	Reparse Point
0800h	Compressed
1000h	Offline
2000h	Not Content Indexed
4000h	Encrypted

Table 2.10: Filename namespaces

Flag	Description
0	POSIX
1	Win32
2	DOS
3	Win32 and DOS (Both names are identical)

Table 2.11: Attributes in a directory MFT entry

Attribute
$STANDARD_INFORMATION
$FILE_NAME
$INDEX_ROOT
$INDEX_ALLOCATION
$BITMAP

well as in one place in the directory. It might be possible to find out more information from these three timestamps than if only one had been used. This will be studied further in our future work.

This far we have seen that the starting point when interpreting the contents of an NTFS partition is the boot record, and that the MFT is the next place to go to when trying to find the contents of a file. Now it is time to take a look at the directory structure in NTFS.

In Table 2.3 we could see that one of the default entries in the MFT is entry 5, the root directory. Table 2.11 contains a list of attributes that can be found in a directory MFT entry. Of these attributes the first one to look at is $INDEX_ROOT. Following the Standard attribute header of this attribute comes an Index Root structure (Table 2.12), an Index Header structure (Table 2.13) and a variable number of Index Entry structures (Table 2.14).

On our reference disk the $INDEX_ROOT of the root directory contained only one Index Entry with Flags = 3, meaning that the entry was the last one and that it pointed to a subnode. The VCN of this subnode in the Index Allocation attribute was zero.

The $INDEX_ALLOCATION attribute starts with a Standard attribute header followed directly by one or more data runs. The data is a series of *Index Blocks*, each of which contain a starting *Index Header* (Table 2.15) followed by a variable number of Index Entries. All directory entries (files and other directories) are represented by an Index Entry each. The Index Entry points to the correct position in the MFT. Also, the Index Entry contains a Stream, which

Table 2.12: Index Root

Offset	Size (B)	Description
00h	4	Attribute type
08h	4	Size of Index Allocation entry (In bytes)
0ch	1	Clusters per Index Record
0dh	3	Padding (Align to 8 bytes)

Table 2.13: Index Header of $INDEX_ROOT

Offset	Size (B)	Description
00h	4	Offset to the first Index Entry
04h	4	Total size of the Index Entries
08h	4	Allocated size of the Index Entries
0ch	1	Flags ($INDEX_ALLOCATION present: 0 = no, 1 = yes)
0dh	3	Padding (Align to 8 bytes)

Table 2.14: Index Entry

Offset	Size (B)	Description
00h	2	Incremented every entry update (Valid unless last entry)
02h	6	Offset in the MFT (Valid unless last entry)
08h	2	Length (L1) of the Index Entry
0ah	2	Length (L2) of the stream
0ch	1	Flags (1 = Points to subnode, 2 = Last entry)
10h	L2	Stream (Present unless last entry)
L1 - 8	8	VCN of subnode in index allocation (Present if last entry)

is a $FILE_NAME attribute structure (see Table 2.8) without the Standard attribute header in front of it. The Index Entries tied to a directory build a kind of B-tree that represents all the directory entries.

This concludes our detailed look at NTFS. Major blocks that have been omitted from the discussion are quotas, security descriptors and journaling. Security descriptors in particular are very interesting in the field of intrusion analysis and looking at how they are implemented in NTFS is included in our future work. Quotas might not be of the biggest interest, while journaling should at least be looked upon further to make sure how much of it could be useful in intrusion analysis. It should be noted that journaling seems to be the least well documented part of NTFS.

2.1.2 The FAT32 file system FAT is an older and much simpler file system than NTFS. There are three versions with minor variations: FAT12, FAT16 and FAT32. Nowadays normally only FAT12 and FAT32 are used - FAT12 for floppy disks and FAT32 for hard disks. Only FAT32 will be covered here since it is probably the most important when performing intrusion

Table 2.15: Index Header of $INDEX_ALLOCATION

Offset	Size (B)	Description
00h	4	Magic string "INDX"
10h	8	VCN of this Index Block in the allocation
18h	4	Offset to the Index Entries (Add 18h to this value!)
1ch	4	Size of Index Entries
20h	4	Allocated size of Index Entries
24h	1	0 = Leaf node, 1 = Not leaf node

Table 2.16: The boot sector in FAT32

Offset	Size (B)	Description
03h	8	OEM String "MSWIN4.1" (Do not assume this value!)
0bh	2	Bytes per sector
0dh	1	Sectors per cluster ($2^n : n = 0$ to 7 are valid)
0eh	2	Reserved sectors (Typically 32)
10h	1	Number of FAT copies
20h	4	Number of sectors on the partition
24h	4	Number of sectors per FAT
28h	2	Flags (Bit 0-3: # of active FATs, Bit 7: one active FAT)
2ah	2	FAT32 version number
2ch	4	Root directory start cluster number

analysis. All FAT versions are officially described in detail by Microsoft [4] [5]. The correct way to determine if a file system is FAT12, FAT16 or FAT32 is to look at the number of clusters on the volume. If there are at least 65525 clusters the file system is FAT32. Despite this it should not be assumed that this is always the case since some disk utilities use incorrect methods to determine the FAT type. In general however, one can use the simple rule that a FAT partition larger than 2 GB is most likely FAT32. This means that most FAT partitions encountered on modern disks are FAT32.

As with NTFS, the starting point when interpreting a FAT32 partition is the boot sector (Table 2.16). The word at offset 0eh in the boot sector specifies the number of Reserved sectors in the partition. This value should be added to the sector number of the first sector of the partition (the boot sector) to find the starting sector of the *File Allocation Table* (FAT). The most common case is a hard disk with only one FAT32 partition starting at sector 63, 32 Reserved sectors, and thus a FAT that starts at sector $63 + 32 = 95$.

The FAT contains a large sequence of 32 bit unsigned integers. Each integer position corresponds to a physical cluster on disk. When one of these clusters belong to a file, the value stored in the corresponding FAT position is the cluster number of the next cluster of that file. Only the 28 least significant bits of each position are really used since the top 4 bits are reserved. To put things short: the FAT is responsible for chaining together the clusters of files and marking which clusters are used for what (store files, damaged, free). The last cluster in a file is marked 0fffffffh, while a damaged cluster is marked 0ffffff7h. Free clusters are marked 0, and when the file system is mounted the FAT is scanned and an in-memory list of all free clusters is built.

The first two entries in the FAT are not used in the same way as the rest. The two most significant bits in the second of these two entries can be used to determine if ScanDisk should be run at the next boot because the filesystem was not properly unmounted or because physical errors were encountered while reading or writing to the partition.

At this point we know how to find all the data in a file as long as we know the number of its first cluster. The number of its first cluster is usually found in a directory, and the first directory we can find in a partition is the root directory. The double word at offset 2ch in the boot sector contains the start

Table 2.17: FAT32 Directory Entry

Offset	Size (B)	Description
00h	11	Short name (8.3)
0bh	1	Attributes
0dh	1	10th of sec. file creation time (0-199) (Optional)
0eh	2	File creation time (Optional)
10h	2	File creation date (Optional)
12h	2	Last access date (Optional)
14h	2	High word of first cluster number
16h	2	Time of last write
18h	2	Date of last write
1ah	2	Low word of first cluster number
1ch	4	File size in bytes

Table 2.18: FAT32 Attributes

Value	Description
01h	Read only
02h	Hidden
04h	System
08h	Volume ID
0fh	Long name
10h	Directory
20h	Archive

cluster number of the root directory. It should be noted that the first cluster of a partition is always numbered 2 - there simply are no clusters 0 or 1.

In the FAT a directory looks just like a regular file. A directory is really a variable length sequence of 32-byte Directory Entries (Table 2.17). The attributes of a Directory Entry are shown in Table 2.18, their ordinary usage is easily understood.

A long file name is stored in one or more Directory Entries before the regular Directory Entry. For long file name entries a special combination of attributes is used, which guarantee that these entries will not be mistaken for regular entries by Windows. The two first cluster entries are always zero for a long directory entry. The long name storage technique will not be covered in further detail here.

The first byte in the short name can either be the first character of the name or a special code. When the value is e5h the entry is free, when it is 00h the entry is also free and also all entries after it are free, and finally when the value is 05h the first character in the file name really is e5h. The short name of the root directory is set to the volume label and its timestamps are not valid.

As for timestamps, there is never any time (only date) available for the last access, which can be seen in Table 2.19. Also, the time of the last write can only contain an even number of seconds, shown in Table 2.20. The creation time however is complemented with a 10th of second count, so the even number of seconds problem does not apply to it. Registering only an even number of

Table 2.19: FAT32 date format

Bits	Description
0-4	Day (1-31)
5-8	Month (1-12)
9-15	Year (0-127 = 1980-2107)

Table 2.20: FAT32 time format

Bits	Description
0-4	2-second count (0-29 = 0-58 seconds)
5-10	Minutes (0-59)
11-15	Hours (0-23)

seconds might perhaps not be a real problem, but it could be in cases where many things have happened quickly and the order needs to be determined. It could also be a problem that those not familiar with the design details naturally assume that all second values are possible.

2.1.3 The Ext2 file system Ext2 (The Second Extended File System) is one of the most commonly used file systems in Linux. The primary documentation is the Linux kernel source code but there are other sources of documentation that are easier to read although not official [6].

The starting point when interpreting an Ext2 partition is the *superblock* (Table 2.21), located at the 1024th byte of the partition. An extended version (2) of the superblock also exists and the additions are shown in Table 2.22. Everything except the superblock is divided into equal sized *blocks*. A large number of blocks are grouped together as a *block group*, and there are normally a number of block groups in a partition. Each block group contains a superblock copy (padded with empty sectors to be one block large), a copy of the group descriptor (Table 2.23) array, a block bitmap, an inode bitmap, an inode table, and finally the data blocks. Each one of these is located at a block boundary. The bits of the block bitmap each represent one block in the block group. A one means that the block is free and a zero means that it is taken. The *inode bitmap* works the same way but for the inodes.

An *inode* (Table 2.24) is used to represent a file. The block array in the inode contain a number of entries. Entries 1-12 contain block numbers to the data of the file. Entry 13 points to a block containing another array of block numbers pointing to more file data. Entry 14 points to an array of block numbers pointing to other arrays of block numbers pointing to blocks of file data. Entry 15 has the same kind of function but with yet another level of arrays. All in all the block pointers form a tree-like structure with different depth depending on the number of data blocks needed. How the different pointer types are related to each other can be seen in Figure 2.1.

Directories are stored as files, with the root directory pointed to by the second entry in the inode table (inode number 2). It should be noted that the inode table is not replicated among the block groups, but rather split among them. The size of the inode table is determined when the file system is

Table 2.21: The Ext2 superblock

Offset	Size (B)	Description
00h	4	Total number of inodes
04h	4	Total number of blocks
08h	4	Number of blocks reserved (see 80h & 82h)
12h	4	Total number of free blocks
16h	4	Total number of free inodes
20h	4	Number of the first data block
24h	4	Block size: 1024 « N, where N = this value
32h	4	Blocks per group
40h	4	Inodes per group
44h	4	The last time the file system was mounted
48h	4	The last time the file system was written to
52h	2	Number of mounts since last full check
54h	2	Max mount times before full check
58h	2	State to determine if cleanly unmounted
64h	4	Time of last full check
68h	4	Max time interval between full checks
72h	4	Created by OS (e.g. 0 = Linux, 3 = FreeBSD)
76h	4	0 = Original version, 1 = Version 2 (Dynamic inodes)
80h	2	Reserved UID
82h	2	Reserved GID

Table 2.22: The Ext2 superblock, version 2 additions

Offset	Size (B)	Description
84h	4	First inode number for use by standard files
88h	4	Inode size
90h	4	Block group number hosting superblock

Table 2.23: Group descriptor

Offset	Size (B)	Description
00h	4	Block id of the first block of the block bitmap
04h	4	Block id of the first block of the inode bitmap
08h	4	Block id of the first block of the inode table
12h	2	Total number of free blocks in the group
14h	2	Total number of free inodes in the group
16h	2	The number of inodes allocated to directories
18h	2	Padding
20h	12	Reserved

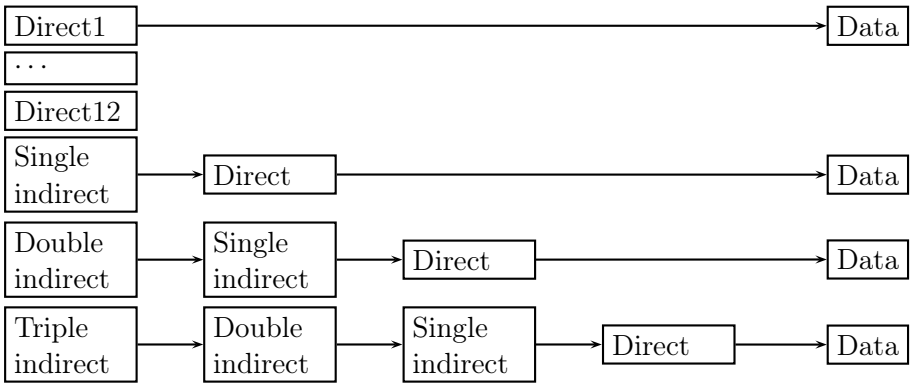

Figure 2.1: The structure of the block pointers in an inode.

Table 2.24: The inode

Offset	Size (B)	Description
00h	2	File format and access rights
02h	2	Owner UID
04h	4	File size in bytes
08h	4	Last access time (Seconds since 1970-01-01)
12h	4	Creation time (Seconds since 1970-01-01)
16h	4	Last modify time (Seconds since 1970-01-01)
20h	4	Deleted time (Seconds since 1970-01-01)
24h	2	Owner GID
26h	2	Links count
28h	4	Number of blocks reserved for file data
40h	15 x 4	Blocks array

formatted. A directory consists of a sequence of *directory entries* (Table 2.25). In the case that the inode represents a symbolic link with a 60 characters long target name or less, the name is stored inside the blocks array. If the name is longer, a block is allocated to hold the name.

2.1.4 The Ext3 file system Ext3 is almost the same as Ext2 except that it has added journaling support. The journaling in Ext3 is described in quite a bit of detail in [7] and [8]. Specific structures can be found in the Linux kernel source code.

The journaling logging can be configured, with an option to the mount command, to one of three separate levels:

Table 2.25: Directory entry

Offset	Size (B)	Description
00h	4	Inode number
04h	2	Offset to the next entry from the beginning of this one
06h	1	Name length (L)
08h	L	Name

Table 2.26: The journal superblock

Offset	Size (B)	Description
0ch	4	Journal block size
10h	4	Total number of blocks in actual log
14h	4	First block of journal file usable for log
1ch	4	First block of actual journal log

Journal, where both metadata and data is logged – the slowest but also the safest level

Ordered, where nothing but metadata is logged but data is written before the metadata – the default level

Writeback, where nothing but metadata is logged – the fastest level

The usual way to store the journal is in a hidden file named .journal in the root directory. The number of the journal inode is specified directly in the superblock, after the Ext2 structure members, at offset 228h. Although not used in practice, it is also possible to specify another device for the storage of the journal.

The journal file starts with a *journal superblock* (Table 2.26). Following the journal superblock comes *journal records* of which there are three kinds:

Revoke Records, used to perform rollback – always one block large

Descriptor Records, used to describe the data – two or more blocks large

Commit Records, used to commit transactions – always one block large

The exact layout of the journal records will not be described in further detail here. The total journal is not very large compared to the whole disk. For example, on the 20 GB reference disk used when writing this section, the journal on the main partition was 4 MB large. Although it can probably be assumed that there is not that much to be found in the journal from an intrusion analysis standpoint, this fact ought to be verified in our future work.

2.2 Logging

Logging in computers is the act of writing a short message about some activity to permanent storage. This generates a list of messages that later can be viewed and analyzed. Most of the current operating systems have some logging facility enabled. Since the storage capacities of persistent memory is limited, the amount of logging also must be limited. If every single instruction executed in the processor would be logged, the log would quickly fill up all available storage space.

In order to make a good intrusion analysis you need as much logging as possible. Not only what the processor does, but every single bit processed anywhere in the computer. Theoretically, you want to log every transition of state in the computer, so the states later can be reconstructed up to the point

where the computer were compromised. This is of course impossible since the activity of the logging would be logged itself, generating an exponential increase of log messages. This problem is mentioned in Laurie's article [9].

Therefore the entropy of the logging must be increased by some sort of aggregation. At higher levels of security requirements each event, packet or block could be logged. At lower levels of requirements the unimportant parts can be stripped off or only the important events is logged. The easiest way is to just let some applications send general log messages or let the kernel only send critical messages to the log files. This is how most operating systems do.

Usually, operating systems logs authorization requests like login and logout. Also, they often logs starting and stopping of services. Some applications like mail servers or web servers logs each connection attempt and a note of what they sent or received. It is possible to let the built in firewall log each packet on the network, but this is not usually the case. The default logging on most operating systems is critical kernel message and messages from sensitive applications.

Logging can be divided into two major types, host based and network based. The network packets can be logged at many points. In the computer by using a network logger program, like *tcpdump* in Linux, or letting the firewall make the logs. The switches or routers can do logging of the packets, or a network tap can listen to a connection.

2.2.1 Host-based logging There are a lot of different things you can log in a computer, but generally they can be divided into the following types:

- Filesystem calls. Log each block written or read.
- Kernel system calls (including parameters). These tend to be difficult to interpret since they are so many.
- Memory audit. Each byte written is impossible to log, but memory status can be logged. In Linux there are programs like *wmmemfree*, *memdump* or *vmstat*.
- Process accounting. Log each process started or stopped, the program executed and the username which started the process.
- Application messages. Including security appliance logs which contains security policy violations.

What is currently logged in different operating systems? In Price's thesis [10] there is a large survey of logging capabilities in different operating systems. The thesis mentions the older VAX Security Kernel [11] which did have quite extensive logging capabilities, and this made it reach the highest security classifications. A few other operating systems, (HP-UX, Solaris, Windows NT) are analyzed in this thesis. Generally, the logging facilities in these must be explicitly enabled or a special module installed. Unfortunately, they are only logging certain sensitive applications like remote login, or in HP-UX case, omitting some applications that are "trusted". Many had detailed logs

of system calls, but often they lacked some small important piece of log data, which made them difficult to audit.

The Crosbie and Kuperman paper [12] concluded that commercial kernel audit was inadequate. There is a conflict between what type of logging is provided by the OS and what an intrusion detection/analysis system needs. Furthermore, the paper suggest a model for kernel audit in real-time, by letting the kernel write records about system calls into a special driver interface. The records is then read by a audit system.

An extensive overview of logging is available at Ranum's web site [13].

What is logged in Linux? Logging in Linux (and in many other Unix-es) is seen as convenient tool for the system administrator to check the status of the system. Or to see what went wrong when something crashed. In all Linux distributions the default logging is saved in the */var/log* directory. Here you usually find

- The *syslog* and *messages* which contains messages from running services. When they start and stop, or when they encounter a special event. These log files can also get messages from the kernel.
- The *kern.log* contains messages from the kernel.
- The login and logout of users are recorded in*wtmp* and can also be logged in *auth.log*
- Process accounting is stored in some file in */var/log* but the filename varies. In this you can see how data which processes has started and how much time they have spent.

This level of logging is of course inadequate for doing a good intrusion analysis. The problem with these simpler logs is that they are intended to be human readable so they are stripped of some detailed information. Automatic tools can handle much more detailed logs.

What is logged in Windows? Windows uses a proprietary binary format (.evt) that logs into three files: system, application and security. The system log contains records about system crashes, component failures and other events of interest. The application log contains records from applications. The security log contains security critical events such as logging in and out, system overuse and accesses to system files. An *event viewer* is used to examine the logs or export them to a file of comma separated entries. [14]

2.2.2 Linux kernel logging In the older 2.4 version of the Linux kernel there existed some patches for the kernel that made it possible to log all system calls. Every system call was logged including the parameters for the call. When the newer 2.6 kernel was developed they decided to remove this facility and replace it with the much improved Linux Security Modules. With these modules it should be possible to log many more events in the kernel with the help of hooks at security critical places. For some reason, which will be

explained later, the old system call logging functionality was re-introduced in the 2.6.7 kernel.

Since the source code of Linux is openly available, it is easier to see exactly what is logged, and where in the kernel code this happens.

LSM - Linux Security Modules Linux Security Modules is a part of the SELinux kernel [15]. LSM is intended as way to control operations at the various security critical points in the Linux kernel. This is done by adding a set of security attributes to some kernel structures and by inserting some hooks in the kernel code. When a LSM is loaded its hook functions are added, and the hooks are called when some security critical event happens. The LSM function can then check the security attributes and accept or deny the access. For example, it can deny loading of certain binaries by adding a function in the *Program Loading Hook*.

The hooks into the kernel are placed at the most convenient places, where they maximize the security functionality. This goal is not compatible with adequate kernel logging, since logging needs to be consistent and all-embracing. In other words, every log record should be in the same event domain, all should be system calls. And all system calls should be recorded, none should be excluded. Since LSM has hooks only at important points, it is not good for logging purposes.

auditd In the 2.6.7 kernel it is possible to do a trace of every system call. This tracing inside the kernel must be enabled at boot time, and to get access to the tracing functionality during runtime from user-space you must recompile the kernel. This connection to user-space is done via a netlink socket and through this, you control the tracing.

A software package named *auditd* contains two user-space programs to control and receive log messages. The name "auditd" would be somewhat misleading since auditing means reviewing records, or should mean in this special situation analyzing system calls. The auditd program is actually only a daemon listening for log messages and printing them to the screen or somewhere else. This package contains a program named *auditctl* which is used to control the auditing. With this you enable or disable the logging, set filters, receive log messages and limit the rate of messages per second. It is possible to filter certain system calls or certain parameters to the calls. The help page gives these options:

```
usage: auditctl [options]
  -h         Help
  -s         Report status
  -e [0|1]   Set enabled flag
  -f [0..2]  Set failure flag
             0=silent 1=printk 2=panic
  -p <pid>   Set pid of auditd (testing only)
  -r <rate>  Set limit in messages/sec (0=none)
  -l         List rules
  -a <l,a>   Add rule at end of <l>ist with <a>ction
```

```
-A <l,a>   Add rule at beginning <l>ist with <a>ction
-d <l,a>   Delete rule from <l>ist with <a>ction
           l=task,entry,exit a=never,possible,always
-S syscall Build rule: syscall name or number
-F f=v     Build rule: field name, value

-m text    Send a user-space message
-L uid,txt Set login uid and send login message
```

Unfortunately, from a forensic viewpoint, the auditd package actually tries to do auditing and not really complete logging. It is possible to do some filtering, which could exclude many system calls deemed to be unimportant. It also do not include all the parameters in the system call since they tend to become quite large, or that some parameter values does not matter. For example, the system call *chroot("foo")* would fail when a ordinary user calls it since users are not allowed to even try to call it. The parameter "foo" does not matter at all, and is therefore not logged. From a audit viewpoint this is correct, but when doing forensics this can be an important key to solve a puzzle.

Hopefully, it is possible to modify the auditd package and the kernel to generate complete records of system calls in the Linux kernel.

2.3 System integrity checking

When doing an intrusion analysis it is useful to quickly be able to see what damage is done to the system which is stored on the hard discs. The damage can be modified or deleted files. It is much more difficult to detect files that have been copied or just read.

Tripwire [16] is a system integrity tool that maintains a database of checksums over specified files and directories in a system. The program monitors key attributes of files that should not change, including binary signature, size, expected change of size or modification dates. With Tripwire it is possible to run periodical checks to see if anything has changed and send warnings to the administrator. This can be used as a slow intrusion detection system, but also for integrity assurance, change management, policy compliance, or later analysis.

Cfengine [17] is a tool better used for policy compliance. It is actually an engine for distributed configuration and administration of large computer networks. It has a autonomous agent with a special policy language. With this language you describe the state of the configuration in the whole network. If for example a configuration file changes somewhere in the network, cfengine detects this and restore the file to its original state. When doing intrusion analysis it is useful when cfengine logs the state changes in the network.

There is also a tool that according to its creator is meant to be better than Tripwire. The following is a citation from the AIDE manual [18].

> "AIDE (Advanced intrusion detection environment) is an intrusion detection program. More specifically a file integrity checker.

> Aide constructs a database of the files specified in Aide.conf, Aide's configuration file. The Aide database stores various file attributes including: permissions, inode number, user, group, file size, mtime and ctime, atime, growing size and number of links. Aide also creates a cryptographic checksum or hash of each file using one or a combination of the following message digest algorithms: sha1, md5, rmd160, tiger (crc32, haval and gost can be compiled in if mhash support is available).
>
> Typically, a system administrator will create an AIDE database on a new system before it is brought onto the network. This first AIDE database is a snapshot of the system in it's normal state and the yardstick by which all subsequent updates and changes will be measured. The database should contain information about key system binaries, libraries, header files, all files that are expected to remain the same over time. The database probably should not contain information about files which change frequently like log files, mail spools, proc filesystems, user's home directories, or temporary directories.
>
> After a break-in, an administrator may begin by examinining the system using system tools like ls, ps, netstat, and who — the very tools most likely to be trojaned. Imagine that ls has been doctored to not show any file named "sniffedpackets.log" and that ps and netstat have been rewritten to not show any information for a process named "sniffdaemond". Even an administrator who had previously printed out on paper the dates and sizes of these key system files can not be certain by comparison that they have not been modified in some way. File dates and sizes can be manipulated, some better root-kits make this trivial.
>
> While it is possible to manipulate file dates and sizes, it is much more difficult to manipulate a single cryptographic checksum like md5, and exponentially more difficult to manipulate each of the entire array of checksums that Aide supports. By rerunning Aide after a break-in, a system administrator can quickly identify changes to key files and have a fairly high degree of confidence as to the accuracy of these findings.
>
> Unfortunately, Aide can not provide absolute sureness about change in files. Like any other system files, Aide's binary and/or database can also be altered."

YAFIC (Yet another file integrity checker) [19] is the name of exactly such a tool. It is using the SHA-1 algorithm and once again the author states that is is meant to be better than the other integrity checkers there are. Its main feature seems to be its size and consequently its speed. The following is quoted from the web site of the tool:

> "yafic is Yet Another File Integrity Checker, similar to programs like Tripwire, integrit, and AIDE. I created yafic because no existing file integrity checker did all the things I wanted. I wanted something fast, simple, and yet be flexible enough to be used in different

situations. yafic uses NIST's SHA-1 hash algorithm to fingerprint files."

There is also a list [19] of features of yafic. The list is quoted below:

- Configuration file format similar to Tripwire.
- Ability to track changes in file attributes like permissions/mode, inode #, number of links, user id, group id, size, access time, modification time, creation/inode modification time.
- Hashes files using SHA-1, a 160-bit hash algorithm.
- Attribute templates (like Tripwire). Add/subtract individual attribute flags.
- Configuration files are parsed in order, making them more intuitive. For example, a rule that prunes a directory can still have its subdirectories/contents scanned by subsequent explicit rules.
- An alternate root besides / may be specified. Paths specified in the configuration file will be interpreted relative to the new root. Useful for checking multiple jail(8) installations.
- Attempts to be platform independent. Makes no assumption about the size of stat(2) fields. If your platform's off_t or time_t are 64-bits wide, yafic will adjust. The tradeoff is that databases cannot be shared across platforms with differing stat's. (Though doing so doesn't really make much sense.)
- Report is short, and to-the-point, allowing easy parsing by scripts. Inspired by integrit.
- Optionally displays SHA-1 hash of resultant database in report. (You can use sha to verify it.)
- Can view the contents of any resultant database.
- Can compare the contents of any two databases.
- Can cryptographically sign and verify databases.

Sentinel [20] is a program that also equals the Tripwire tool, but this tool uses the RIPEMD-160 hashing algorithm instead of the MD5 that Tripwire uses. The tool is only available for Linux. There is not much of information regarding the tool, but in the readme.sentinel file accompanying the source code download the following text can be found:

> "Sentinel is designed to detect changes to the integrity of directories and files on a disk. It tries to defeat the usual attacks (even by persons with root/superuser equivalent access) that have plagued other products such as the commercial "Tripwire(TM)", fcheck.pl or viper.pl. It is by no means a replacement for any other product and does not claim to offer *any* of the functionality of

any other product such as Tripwire(TM) , fcheck or viper. It is *not* a replacement for any of these products or any others. It is by no means *better* or *worse* than any of these tools. It cannot be compared as it has a different purpose and a different development model with different goals."

3. Future work

In this chapter issues and questions left for future work are presented. As this is the first year report most of the work is still left to be done. We will not be able to cover all of the questions that are raised in this chapter during the two remaining years, but we have included them here both for others to be able to continue our work and as a guide for us.

3.1 File system

The section describing the file system basics raised a lot of questions. Some of them are not directly connected to file systems, but to the surrounding layers. Many of the questions concern the hardware layer, and especially disk controllers and cache. We also saw that the tools used for intrusion analysis are important to understand in detail.

The main questions that need to be answered by future work are:

- How do other common file systems work in detail? Primarily ReiserFS.
- How does file deletion work in all the file systems investigated?
- What happens on disk when various log cleaning and manipulation tools are used?
- What can various forensics tools do, and in which areas can they be tricked to misinterpret a disk?
- How do disk controllers work? How does this relate to intrusion analysis?
- Do disk controllers change anything on disk without being asked to by the operating system?
- How does the interfaces to ATA, S-ATA and SCSI work in detail?
- What is the state-of-the-art in data recovery from crashed hard disks?
- Is there any information remaining in the caches of a hard disk when it has been wiped, but the power is still switched on?
- Do CD-ROM disks contain slack space?
- What pitfalls are there to be avoided when performing secure disk wiping?
- Which tools correctly handle disks that have a setmax other than the size of the physical disk?

- How do the G and P lists work in detail? How can they be read and / or manipulated?
- How well do disk write blockers work?
- How can the information in swap files and partitions be extracted in the best way? Which information is available from them?
- Can the journal in a journaling file system be used for intrusion analysis?
- How can as much information as possible be extracted, using software only, from a damaged hard disk?

As can be seen from the list the questions might raise yet other questions. These are of course possible to address directly, but we chose to leave them out because the list tended to grow exponentially when we increased their level of detail.

3.2 Logging

What special requirements does intrusion analysis have on logging? How do you stop the hackers from turning off logging or modifying the logs? There are a number of logging utilities that offer secure and remote logging [14], but are they secure, and what security measures are used to protect the central log servers? The logs can get quite large if they are not filtered or aggregated, but how should such a filtering be done? What log messages are uninteresting and hence should be filtered out? Finally, when the logs are properly filtered, the correct way to perform the intrusion analysis has to be chosen.

Tcpdump is a logging facility for network traffic and thus important to know. The requirement for an effective tcpdump log is to use a binary format, but that has to be converted into a human readable format to be of any real use. Hence we have to learn the core of the tcpdump tool and its different logging formats.

Having the correct time is utterly important in an intrusion analysis situation. Therefore we have to study how the system clock is handled in different operating systems. This issue is also connected to file systems, for example NTFS saves MAC times in three different places (see Section 2.1.1). Are all of those the same and what time is really written to them, raw system clock or the OS converted time?

Different applications also retain information, a typical example is Word. What other applications do the same, what can be found, and how can this be counter-acted?

3.3 Integrity checking

The tools for integrity checking presented in the previous chapter are only a small part of all the tools available. Thus a more thorough survey of the tools has to be made. At the same time the have to be evaluated and tested empirically to give an idea of their advantages and disadvantages.

The concept of controlling the integrity of the binaries and executables in the computer is a interesting idea that has to be further developed.

4. Conclusion

This chapter contains the conclusions drawn from the material presented in the report.

Our studies of different file systems pointed us towards the importance of really understanding the core parts of a computer. Without knowledge of how the storage is structured, we cannot follow an intruder's every step. Nor can we develop countermeasures and security solutions if we do not know what really is happening during an intrusion.

The studies showed that there are several important questions left that need to be answered, one is how time is handled in different operating systems. Another is how deletion is done and what can be recovered when deleted. We also need to expand the studies to include newer journaling file systems from the Linux family.

Logging is in a way connected to all of the above mentioned issues. The report presented the basics of logging from which we can see that the way logging has been done over time has changed back and forth. One example is the kernel hooks that were available in earlier versions of the kernel, then disappeared, only to be reinstated again in kernel 2.6.7.

All these issues show one thing, the unquestionable need for more research in the field. Intrusion analysis is the key to many different security related areas and if we do not master the foundation on which everything rests we cannot succeed in securing our systems. Thus we need to keep pace with the hackers and one way to do that is by being able to trace their actions in our systems.

Bibliography

[1] M. Karresand, "Intrusion analysis in military networks – an introduction," Dept. of Systems Development and IT Security, Command and Control Systems, Swedish Defence Research Agency, P.O. Box 1165, SE-581 11 Linköping, Sweden, Tech. Rep. FOI-R–1463–SE, Dec. 2004.

[2] D. Solomon and M. Russinovich, *Inside Microsoft Windows 2000*. Microsoft Press, 2000.

[3] R. Russon, "NTFS documentation," http://linux-ntfs.sourceforge.net/ntfs/, last visited 2004-11-16.

[4] Microsoft Corporation, "FAT: General overview of on-disk format, version 1.02," 1999.

[5] ——, "Long filename specification, version 0.5," 1992.

[6] D. Poirier, "The second extended file system - internal layout," 2002.

[7] S. Tweedie, "The linux journalling filesystem layer, 1.0 draft," 2001.

[8] D. Bovet and M. Cesati, *Understanding the Linux kernel*. O'Reilly, 2002.

[9] B. Laurie, "Network forensics," *ACM Queue*, vol. 2, no. 4, pp. 50–56, June 2004.

[10] K. Price, "Host-based misuse detection and conventional operating systems audit data collection," Master's thesis, Purdue University, 1997.

[11] K. F. Seiden and J. P. Melanson, "The auditing facility for a vmm security kernel," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1990.

[12] M. J. Crosbie and B. A. Kuperman, "A building block approach to intrusion detection," in *RAID 2001 Fourth International Symposium on Recent Advances in Intrusion Detection*, 2001.

[13] M. Ranum, "Log analysis," http://www.loganalysis.org/, last visited 2004-11-26.

[14] C. Peikari and A. Chuvakin, *Security Warrior*. O'Reilly, Jan. 2004.

[15] National Security Agency, "SELinux," http://www.nsa.gov/selinux/, last visited 2004-11-26.

[16] "Tripwire," http://www.tripwire.org/, last visited 2004-11-30.

[17] M. Burgess, "Cfengine - a configuration engine for unix and windows," http://www.cfengine.org/, last visited 2004-11-30.

[18] R. Lehti, "The Aide manual," http://www.cs.tut.fi/~rammer/aide/manual.html, last visited 2004-12-03.

[19] A. Saddi, "Yet another file integrity checker," http://philosophysw.com/software/yafic/, last visited 2004-12-03.

[20] Zurk Technology Inc., "Sentinel security toolkit development," http://zurk.sourceforge.net/zfile.html, last visited 2004-12-03.