Reference: CU/CFC/PROJECT/4 – NORRIS

CRANFIELD UNIVERSITY

DEFENCE COLLEGE OF MANAGEMENT AND TECHNOLOGY

DEPARTMENT OF INFORMATICS AND SENSORS

MSc THESIS

Academic Year 2008-2009

Peter Norris BSc (Hons), MBCS

THE INTERNAL STRUCTURE OF THE WINDOWS REGISTRY

Supervisor: Professor AJ Sammes

February 2009

This thesis is submitted in partial fulfilment of the requirements for the Degree of Master of Science.

DISTRIBUTION

<u>EXTERNAL</u>

Peter Norris (Student)	2 Hard

<u>INTERNAL</u>

Prof AJ Sammes (Supervisor)	1 Hard
DCMT Library	2 Hard, 1 Soft

EXECUTIVE SUMMARY

THE INTERNAL STRUCTURE OF THE WINDOWS REGISTRY

The Windows Registry is a vital source of Forensic information about the current state of a Windows computer and also about events which have happened on the computer.

Fragments of the Registry can be found in other than the Registry such as in the pagefile, unallocated or slack space in the file system or in memory. These fragments can in theory be used to rebuild a damaged Registry file or to reconstruct the previous state of a Registry file. They can also stand on their own as items of evidence.

In more recent versions of the Windows Operating System the Registry Keys contain a Last Written date and time. This is particularly evidentially valuable as it can contribute to the development of a timeline of activity on the computer.

In order to find fragments of Registry Keys it is necessary to understand the structure of the Registry and of the Keys that it consists of down to the level of individual bits and bytes.

This project primarily aims to understand the structure of the Registry Files and other parts of the Registry and to do so in ways that are academically sound and which will hence provide a firm basis for any forensic examinations.

A secondary aim is to provide ways of analysing fragments of Registry which might be found in other than the Registry files such as, for example, unallocated space in the file system.

INFORMATIVE ABSTRACT

1. Originator's Report number. CU/CFC/PROJECT/4 – NORRIS	
2. Originator's Name & Location. Peter Norris Defence College of Management and Technology, SHRIVENHAM, Swindon, Wiltshire, SN6 8LA.	
3. MOD Contract Number & Period Covered. Not Applicable	
4. Sponsor's Name & Location. Not Applicable	
5. Report Classification & Caveats on use. UNCLASSIFIED – None	**6. Date Written.** February 2009 **Pagination.** N/A **References.** DCMT Library
7a. Report Title. The internal structure of the Windows Registry	
7b. Translation/Conference Details. Prepared for number 4 Forensic Computing Course Project.	
7c. Title Classification. UNCLASSIFIED	
8. Author. Peter Norris BSc (Hons) MBCS	
9. Descriptors/keywords. Forensic Computing, Windows Registry, Key, Value, hbin, regf	
10a. Abstract. Fragments of the Windows Registry can often be found in pagefiles, unused space in the file system and in memory. This project aims first of all to understand in detail the structure of the Registry Files and so of these fragments. A secondary purpose is to attempt to find ways of identifying and recovering Registry fragments in a forensically sound way.	
10b. Abstract Classification. UNCLASSIFIED	

ACKNOWLEDGEMENTS

I am grateful for and acknowledge the help and encouragement given to me by the following and others.

Professor AJ Sammes

Marc Kirby

Lindsey Gilles

Lindy Sheppard

Professor Brian Jenkinson

Professor Rob Witty

Chris Hargreaves

Jim Gordon

Matt Lacey

Jolanta Thomassen

Adam Clark

Special thanks is due to Tony Sammes whose patient, relevant and well judged support and guidance have made a mountain of difference. Without his help I could not have achieved as much as I have.

Thanks also to Brian Jenkinson whose original idea for this project it was and to Adam Clark for having the patience to be my proof-reader and for having made so many suggestions for improving the writing.

Contents

Appendices – Contents

Supplements – Contents

Glossary

Term	Explanation
ACE	Access Control Entry, defines what control is applied and which user or group it refers to. Part of a Security Descriptor
ACL	Access Control List, list of ACEs. Part of a Security Descriptor.
ADSIEdit	Active Directory System Information Editor. The Active Directory equivalent of regedit for the Registry
ANSI-C	C Programming Language, distinct from C++
Bit	A single binary digit. Can have a value of either 0 or 1.
Byte	A unit of storage, either in memory or on disc. Consists of 8 bits and can therefore hold up to 2^8 or 256_{10} values.
C++	Object Orientated Programming Language
CPAN	Comprehensive Perl Archive Network. A web site which holds a repository of Perl code and modules
DACL	Discretionary ACL. Used to Control Access.
Distro	A Linux Distribution or Build. Eg Ubuntu or RedHat
DWORD	Four Bytes
FILETIME	Windows 64 bit time format
Group Policy	A policy which may be applied centrally in a business to many computers.
Group Policy Editor	A Windows program for editing Group Policies
Hash	A reduction of a block of data in some way to some fixed length number. Sometimes intended to be unique.
Helix	Bootable CD with Linux and tools made for forensic use
Hive	A part of the Registry, the Registry is made of Hives
Linux	An Operating System based on Unix
MD5	A hash function that is designed to provide a unique signature for a block of data.
MD5 Password Hash	A number that represents the password. As this process is irreversible it uniquely identifies the password without revealing it.
Microsoft Access	A Database Program from Microsoft
MSDN	Microsoft Developer Network

Term	Explanation
NT	New Technology. Family of Windows Operating Systems, distinct from Windows 9.x such as Windows 95.
NTFS	NT File System
Pagefile	A system or special file in the computers file system that is used as a temporary store for pages of memory for which there is not enough real or physical memory.
Perl	A Programming Language
Python	A Programming Language
regedit	A Windows program for editing Registry data
Registry File	A file that stores part of the Registry.
SACL	Security ACL. Used for Auditing Access
Salt	An extra element of data added to an item to be encrypted to reduce vulnerability to dictionary attack.
SAM	Security Accounts Manager. Name of a Registry file and hive
Security Descriptor	A data structure that defines the Security on a Windows object, which can be a Registry Key
SID	Security ID. Identifies a Security Trustee such as a User or a Group. Some are common and well-known such as S-1-5-18
Slack Space	Space in a computers file system is allocated in blocks. It is quite normal for a file to not completely fill the last block allocated to it. The space between the end of the file and the end of the last block is called Slack. This will often contain parts of the file that was previously stored at that location in the file system.
SP2	Service Pack 2
ULONG	Unsigned Long. Four Bytes
Unallocated Space	Space in a computers file system is allocated in blocks. Any blocks that are not currently allocated to a file are known as Unallocated. They may contain parts or all of files that were previously stored there.
Unix	An Operating System
VBScript	Windows Scripting Language based on Visual Basic
Visual Basic	Microsoft version of BASIC. A Programming Language.
Windows	A widely used Operating System produced by Microsoft.
Windows CE	A Windows 9.x operating system

Term	Explanation
Windows Registry	“A central hierarchical database used in Microsoft Windows … to store information that is necessary to configure the system for one or more users, applications and hardware devices.” (Microsoft, 2008j)
WORD	Two Bytes

Chapter 1 – Introduction

This dissertation has been produced as part of the MSc in Forensic Computing.

1.1 Purpose of Project

The Windows Registry is a central repository of configuration and other information about the Operating System and the programs that run on it.

It is a vital source of Forensic information about the current state of a Windows computer and events which have happened on the computer. This is often crucial in solving criminal cases.

Fragments of the Registry data can be found in other than the Registry such as in the pagefile, unallocated or slack space in the file system or in memory. These fragments can be used to rebuild a damaged Registry file or to reconstruct the previous state of a Registry file. They can also stand on their own as items of evidence.

In more recent versions of the Windows Operating System the Registry Keys contain a Last Written date and time. This is particularly evidentially valuable as it can contribute to the development of a timeline of activity on the computer.

Conceptually the Registry is presented as a series of Hives each of which consists of a number of Keys. Each Key consists of either further Keys or Values or both. Values can each be one of a number of pre-defined types such as a number or a string of characters.

Physically the Registry can be found in a number of files. These files are mainly stored in the Windows System folder in a folder called Config with the notable exception of the Registry Files that hold the user specific information. These are held in each Users profile and will normally only be loaded if the User is logged on.

In order to find fragments of Registry Keys it is necessary to understand the structure of the Registry Files, and of the Keys and Values that they consist of, down to the level of individual bits and bytes.

The value of this work is that it can allow fragments of the registry to be captured as evidence. This can show a previous state of at least parts of the Registry or can provide the only evidence of the contents of the registry in cases where the hard disk has been partially erased such as to delete some or all of the current Registry Files.

1.2 Overall direction of Project

The intention was for the project to run through a number of stages.

1. Discovery of the structure of the Registry through both Literature and other searches and by experimentation.
2. Determination of a pattern matching algorithm to determine the likelihood that an arbitrary byte string contains one or more Registry keys.
3. The production of a computer program to automatically search an arbitrary byte string for possible Registry keys.

It was predicted that as the project unfolded its direction may change and in particular that the production of programs might not be achievable.

In practice unravelling the structure was found to be both rewarding and valuable and the project tended towards providing a detailed breakdown of the Registry structures.

This was successful to the point where nearly all the bytes of Registry Files have been ascribed their correct Microsoft names and the full purpose of all but a small number of not very significant bytes have been understood.

Some programming was done in the course of the investigation and ideas for a number of future programs have been developed.

It has been possible to determine a simple and effective pattern which it is expected will reliably find Registry Keys in an arbitrary block of data with a very low level of false positives.

1.3 Management of the Project

The project was conducted under the Supervision of Professor AJ Sammes.

Project meetings were held, initially monthly, at which the following Model Agenda was used unless circumstances indicated otherwise.

- Review of Notes of previous Meeting
- Review Progress against Plan
- Specific Issues – PN
- Specific Issues – AJS
- Date, Time and Place of next meeting

It was possible to increase the pace of work in the later months and during this time meetings were held every two to three weeks to reflect this.

A project plan was produced and maintained in the form of a Gantt chart.

The Initial Gantt chart is shown in Appendix 1.

In initial planning activities were listed in their anticipated order and a time for each was estimated. These were then aggregated into sensible phases and entered onto a Gantt Chart.

A time management plan for the Project has been produced. This is in two parts. The first part is a narrative explanation of how the needed amount of time can be found in the course of a normal week, undisturbed by non-regular events such as holidays and Bank Holidays. This document also explains where contingency can be found. Some 430 hours of contingency were initially identified.

The second part is an Excel spreadsheet which shows, week by week, how many hours can be delivered allowing for all anticipated activities, such as holidays.

The Excel spreadsheet and the 'hours per task' figures from the Gantt Chart were used to add start times to the tasks. These were rigorously matched so that they both showed each phase, and each task within each phase, to start and finish at the same dates and times.

The planned completion date for the project was initially set at Wednesday 10th December 2007 at 19:00. This allowed 2 months of leeway before the hand-in date. The project was re-planned in mid-September and a new plan was produced showing a completion date of Friday 19th December and this was approximately achieved. This plan is shown in Appendix 2.

1.4 Aim and Scope

The aim was to develop and test a verifiable definition of the structure of the Windows Registry, concentrating on Windows XP. Some attention was paid to Vista and other Windows NT versions. Windows 9.x versions were excluded.

The intention was to use this knowledge to attempt to design a method and create a tool to find Windows Registry keys and values in other than the Registry.

1.5 Report Structure

The remainder of this report contains the following chapters:

Chapter 2, *Literature Survey*, covers the currently available sources of information that have been found and used as input to this project. Derive an understanding of the current state of knowledge of the structure of the Registry Files. Determine a list of items to verify and areas that need further research.

Chapter 3, *Methodology*, explains the chosen methods used for examining this area of work and explains the reasons for these choices and why other methods were not used.

Chapter 4, *Experimental Work*, what experiments were done using the Methodology outlined in Chapter 3. What results were obtained. How this affected the understanding as derived from Chapter 2.

Chapter 5, *Results*, explains and discusses the results that were found.

Chapter 6, *Critical Analysis*, appraises and reviews the project to show understanding of it's weaknesses and possible area for improvement.

Chapter 7, *Conclusion*, presents a summary of the main conclusions from the project and indicates areas of further work.

Due to the volume of information produced by this project the material has, were possible, been split off into a number of Appendices and Supplements.

The distinction made between an Appendix and a Supplement is that the Appendices are for the more important and smaller pieces of information and the Supplements are for the less important and larger pieces of information.

The Appendices are printed and bound with the main body of the report; the Supplements are not.

The main body of the report, with the Appendices, the Supplements, the Programs and Source Code have all been provided on a companion CD which should be available at the back of the printed version of the report.

Chapter 2 – Literature Search

In this chapter the purposes of the Literature source are briefly reviewed.

The various sources discovered during the Literature Search will be reviewed.

It will be shown that there are very few academic resources currently available but there are a large number of informal sources some of which are very valuable.

A simple outline of the basic structure of the Registry will be presented before we move on to more detailed analysis.

2.1 Purpose of Literature Search

The purpose of the Literature Search is to determine the current state of knowledge about the subject area and so avoid on the one hand repeating work that has already been done and on the other hand discover material which can be used to support the project. To find shoulders to stand on.

It is important to find information in peer-reviewed academic articles as they provide a solid base on which to build. In addition is information in other sources which includes books and internet material. Although not peer-reviewed these sources do provide a body of information which can provide understanding and insights.

2.2 Sources

Only a small number of books and academic sources were found.

Most of the sources of information are either from the source code of utilities or projects or else are from researchers, forensic and otherwise, who have an interest in unravelling the structure of the Registry. In some cases sources fit into both categories.

This is a fast moving and emerging field and the preponderance of more informal sources is perhaps to be expected.

2.3 Anonymity

Much of the material available on the internet is anonymous or the authors are no longer contactable.

The desire of some of these authors to remain anonymous could be because of real or imagined concerns about the legality of their work. This may be concern over copyright issues or perhaps because much of the work is focussed on cracking the structure of the SAM Registry File in order to extract or reset User Account Passwords.

2.4 Root Sources

Two outstanding references emerged from the Literature Search as both essential for anyone who wants to learn about the internals of the Registry and as the sources or starting points for most if not all the work that has been done in this area.

The first of these is the section entitled “Registry Internals” in the book “Microsoft Windows Internals” (Russinovich et al., 2005a). This is, in itself a version of the article that first appeared in the “Windows NT Journal” c. 1999 and is also available online (Russinovich, undated).

This article is well-written and contains a most lucid and accurate description of the internals of the Registry. It rewards careful and repeated reading.

The other resource is effectively anonymous being signed only with the initials “B.D.”. It can be found on the internet where it is referenced and used by a number of other authors and researchers. It will be referred to here as WinReg.txt and is available on this web site and others (B.D., undated).

This document covers both Windows 9.x versions and Windows NT versions of the registry. For both of these it provides some moderately detailed byte level analysis of the structures and shows quite clearly how the basic structures of an NT registry fit together.

WinReg.txt has been improved on and modified by a number of researchers. Notable by Petter Nordahl-Hagen in his work to produce the “Offline NT Password & Registry Editor” tool (Nordahl-Hagen, undated) and by Nigel Williams in his simple Registry Editor program dosreg,c (Williams, 2000).

2.5 Learning Perl

Early research showed that many utilities that parsed Registry Files cold or offline are written in either C or Perl. The author was already competent at C and to open up these Perl utilities to examination it was decided as part of the Project brief to learn Perl as well.

The initial attempt was to do this using the book “Programming Perl” (Wall et al., 2000a) which was written by the author and originator of Perl. This was found to be very much an enthusiasts book and was not a suitable book for learning Perl from scratch. An example of this is where a feature is introduced followed by the comment that this feature is of no use unless you want to re-write the Perl Debugger.

A second attempt was made using the book “Learning Perl” (Schwartz et al., 2005). This was written by someone who has specialised in teaching Perl and was a much better book to learn from. This might have been easier to digest as the main radical differences between Perl and other languages had already been introduced by the first book.

2.6 Journals and other Academic Sources

The following journals were searched for any articles relevant to the internal structure of the Registry Files. Search words used were "Windows Registry" and "HBIN". ("hbin" is the signature found at the front of an hbin record and is so fundamental to the internal structure of the Registry and so well known that the term will almost certainly be found in any article that refers to the internal structure of Registry Files).

All issues of the following Journals (which were all the relevant titles that could be found) were searched and initially no relevant results were found.

Digital Forensic Research Workshop
Digital Investigation
International Journal of Digital Evidence
Journal of Digital Forensic Practice
Journal of Digital Forensics, Security and Law
Small Scale Digital Device Forensics Journal
International Journal of Electronic Security and Digital Forensics

The online libraries of the Institution of Engineering and Technology (IET), the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers IEEE were searched.

A number of articles dealing with the Registry were found but these were all to do with the external structure of the Registry as visible by a tool such a regedit.exe. Initially no articles were found that dealt with the internal structure of the Registry.

The MSc dissertation of David Titheridge (Titheridge, 2008) contains a discussion about the internal structure of the Registry Files. Initially this was the only academic reference that was found.

Following the 2008 Digital Forensic Research Workshop held in Baltimore, MD two useful and relevant articles which had been presented there were published in "Digital Investigation". These were "Recovering deleted data from the Windows registry" (Morgan, 2008a) and "Forensic Analysis of the Windows registry in memory" (Dolan-Gavitt, 2008).

The author of the latter article, Brendan Dolan-Gavitt, is also the author of the "Push The Red Button" blog which he writes under the pseudonym "moyix".

On the 11th December 2008 Jolanta Thomassen was kind enough to supply a copy of her MSc dissertation "Forensic Analysis of Unallocated Space in Windows Registry Hive Files" (Thomassen, 2008) barely a month after submission. It was useful and interesting.

2.7 Non-Academic Sources

Non-Academic sources can be split into books, which are arguably semi-academic in that they have permanence and attributable authorship, and internet resources which may be quite ephemeral, of widely varying credibility and at times hard to determine authorship.

In a fast moving research area such as this, internet based information can provide valuable insights and information and cannot be ignored. The internet information found can be split into two areas, pure information and software tools designed to look into the Registry, particularly the source code thereof.

2.7.1 Books

Just two books were found with information about the structure of the Windows Registry Files.

The first and most authoritative is the previously mentioned “Microsoft Windows Internals” by Mark Russinovich and David Solomon (Russinovich, 2005a).

The other book is Carvey (2007a) which has an entire chapter (chapter 4) devoted to the registry. Within this chapter is a Section called “Registry Structure within a Hive File” which has an excellent description of the structure of Registry Files. This also contains some code fragments for decoding parts of the Registry using Perl.

Contained on the companion DVD for this book is a Perl script called regp.pl which parses and decodes a Registry File and outputs its contents in plain text.

This script has a number of noteworthy features, shared with other tools in this area of work.

- It is offline and so ideal for Forensic work as it is strictly read-only and does not rely on any Win32 API calls which may have hidden side-effects.
- In itself it documents the structure of Registry Files and demonstrates the completeness of that knowledge.

2.7.2 Internet Sources of Information only

2.7.2.1 WinReg.txt

One of the most influential documents, which has already been mentioned, is the text file WinReg.txt by the otherwise anonymous B.D. (B.D. undated).

2.7.2.2 Beginning to see the light

Another, purely informational source, is the single web page “security accounts manager” on the web site “Beginning to See the Light” (Clark, undated). This large page has four sections, one of which is titled “Registry Structure”. This section has a very detailed description of the internal structure of the Registry and contains some information not found elsewhere. Of particular note is the description of the “sk” record and the internal structure of the NT Security Descriptor that it contains which appears to be quite unique and is applicable in other areas where the NT Security Descriptor is used, for example in the NTFS.

A change log on the site indicates that this was first written in January 2001 and last updated in April 2005.

This web site also has two utilities for download, one of which is to decode 64 and 32 bit dates, the other calculates the checksum for the Registry File header. The date utility is notable in that it converts 64 bit times, arithmetically, to 100 Nano-Second intervals.

2.7.2.3 Probert

One of the most valuable discoveries was a PDF on the Tokyo University web site of a presentation on the internal structure of the registry given by David Probert of the Microsoft Windows Kernel Development Team (Probert, undated). This is valuable not only for its apparent authenticity and authority but also as it covers areas not mentioned anywhere else.

2.7.2.4 Microsoft

The only source on the Microsoft web site which was specifically to do with the structure of the Registry was the previously mentioned article by Mark Russinovich (Russinovich, undated).

A large number of articles were of use, more than twenty. Most of these are to do with the structure of the NT Security Descriptor – more on this later. These articles are too numerous to detail here, they are referenced where used.

2.7.2.5 Windows IR

Harlan Carvey has some interesting things to say in his blog “Windows Incident Response” about the registry, mainly in terms of looking at it externally (Carvey, 2008a).

He has produced a tool called RegRipper whose purpose is to extract a timeline of events by ordering keys according to their last written Date/Time. It has a command line version called rip.exe. Both of these are written in Perl (Carvey, 2008b).

2.7.2.6 Push the Red Button

Harlan Carveys blog has references to another blog of interest called “Push The Red Button” (believed to be a reference to the First Responder nature of the work).

In this blog the author (tag moyix but known to be Brendan Dolan-Gavitt) explores the reconstruction of Registry Hives from data held in memory. There are five articles of interest.

> "Challenges in Carving Registry Hives from Memory" about carving Registry data out of memory (moyix, 2007).
>
> "Enumerating Registry Hives" about the data structures used to represent hives in memory (moyix, 2008a)
>
> "Reading OpenKeys" shows how to determine which keys are open. This is mainly of value in an Incident Response situation (moyix, 2008b).
>
> "CredDump: Extract Credentials from Windows Registry Hives" is a tool for extracting MD5 hashes from SAM in an offline way. This does not need any Windows support (obviously does not work by DLL injection) (moyix, 2008c).
>
> "Cell Index Translation" is all about translating in-memory Cell Index to actual address in memory (moyix, 2008d).

The source code for CredDump was found on Google (moyix, 2008e) and was saved for analysis.

The main thrust of these articles is towards the analysis of memory dumps such as might be taken as a "live forensics" response to an incident or as part of a seizure before the target computer is turned off.

2.7.3 Open Source Operating System Projects

There are three open source projects which have reverse engineered the Windows Registry.

One is the WINE project which seeks to provide a Windows environment on other than Windows. Another is the SAMBA project which provides Windows file sharing on other than Windows. The ReactOS project aims to provide a complete Windows clone and does not rely on any other underlying operating system.

There is a risk in studying these open-source registry emulations which is that although they will need to provide a very good, if not perfect, emulation of the native windows API and other calls, that does not mean that the underlying raw Registry File format needs to be the same.

2.7.3.1 WINE

The WINE project which is intended to provide a Windows compatible environment but based on the Linux kernel. This is an ambitious project which has recently (June 2008) reached its first release.

From “World Wine News” of 18th June 2008 (WINE, 2008a)

> “The Wine team is proud to announce that Wine 1.0 is now available. This is the first stable release of Wine after 15 years of development and beta testing.”

From the WINE web site (WINE, 2008b)

> “Wine is an Open Source implementation of the Windows API on top of X, OpenGL, and Unix.”.

The WINE source code was downloaded for later examination, a 14MByte bzip (Sourceforge, 2008a).

2.7.3.2 SAMBA

The venerable SAMBA project aims to provide Windows compatible network file sharing compatibility from Unix/Linux platforms.

From the SAMBA web site (SAMBA, 2008a).

> “Samba is an Open Source/Free Software suite that has, since 1992, provided file and print services to all manner of SMB/CIFS clients, including the numerous versions of Microsoft Windows operating systems.”

The SAMBA Source code was downloaded for later examination, a 24MByte tar (Sourceforge, 2008b).

A cursory study of the SAMBA Registry file structures seems to indicate that they are broadly compatible with the Windows registry file structures. However at least one comment has been found which indicates an incompatibility.

2.7.3.3 ReactOS

This is another Windows clone, like WINE, but this is designed to run on bare hardware rather than on top of Linux. From their web site (ReactOS 2008a).

> “ReactOS® is a free, modern operating system based on the design of Windows® XP/2003. Written completely from scratch, it aims to follow the Windows® architecture designed by Microsoft from the hardware level right through to the application level. This is not a Linux based system, and shares none of the unix architecture.”

Currently it is in alpha testing only.

Due to it's different approach it is possibly more of a clone, and hence more useful to this project, than WINE.

The source code was downloaded, a 62MByte zip (Sourceforge, 2008b).

2.7.4 Open Source Tools

2.7.4.1 CredDump

This tool came from the "Push The Red Button" blog (moyix, 2008c) and is designed to extract MD5 password hashes from the SAM offline without any help from Windows. At the time of writing no other open source tool is known to do this offline. It needs the SYSTEM registry file as well to get the Syskey salt.

Its value to this project is the insight the code gives to the structure of these Registry Files. It is written in Python (Martelli, 2006).

Source has been saved from the Google repository for examination (moyix, 2008e).

The main code walking algorithm is rawreg.py.

2.7.4.2 BKHIVE and SAMDUMP2 Programs

These programs are written in C and make use of the information from WinReg.txt (B.D., undated).

BKHIVE is designed to extract the Syskey from the System hive which is needed to decode (or de-obfuscate) the MD5 password hashes from the SAM hive. To do this it traverses the key structure of the System hive.

SAMDUMP is designed to actually do the extraction and de-obfuscation of the MD5 hashes.

From the "bkhive.h" file and the "hive.h" file used by SAMDUMP2.

> "Hive file access, pretty lame and bugged but do the work O_o
> Thanks to B.D. for file structure info"

The use of these to this project is as simple pieces of code to show an understanding of the registry structure. The source code files for both files are dated 13th October 2006 or earlier.

2.7.4.3 REGLOOKUP Program

This is a small program written by Timothy D. Morgan.

From the web site (Morgan, 2008b)

> "RegLookup is a small command line utility for reading and querying Windows NT-based registries. … Original source was borrowed from the program editreg, written by Richard Sharpe. It has since been rewritten to use the regfio library, written by Gerald Carter."

and

> "I have branched it from editreg.c because I wanted something that was 100% read-only, and would be easy to use in scripts for forensic investigations."

This is another piece of work that relies on WinReg.txt from B.D. (B.D., undated).

Regfio is part of the SAMBA project (SAMBA, 2008a).

Editreg was an NT Registry viewer/editor that was part of the SAMBA project but was discontinued in May 2005 (Samba, 2005).

2.7.4.4 Offline NT Password & Registry Editor Program

This was written by Petter Nordahl-Hagen (Nordahl-Hagen, undated) and is based on WinReg.txt (B.D., undated).

The program is principally designed to allow the local account passwords to be reset or set to a fixed value of blank by directly modifying the SAM Registry File and to do this offline (that is, not from Windows).

The functionality of this tool has been extended to allow registry files to be edited.

The tool runs from a Linux platform which may be on a bootable CD.

A lot of this program will be concerned with the details of how user account passwords are obfuscated and stored in the SAM. The main interest for this project is what the program reveals about how the registry is structured.

2.7.4.5 Perl Parse:Win32Registry Module

Much of the current work being done by Dolan-Gavitt, Carvey and others is based on Perl.

One of the apparent reasons for this is the availability of the Parse:Win32Registry module (McFarland, 2008) which neatly encapsulates the basic routines needed to access the registry offline. As this tool is in current use by multiple users it carries a significant amount of contemporary authority.

The module is available on CPAN (CPAN, 2008) which is the well-respected and widely used web based repository for all manner of Perl modules and programs. CPAN stands for Comprehensive Perl Archive Network (Wall, 2000b).

Version 0.30 of this module came with three utility scripts or tools

regdiff.pl	compare two registry files and outputs differences
regdump.pl	displays the keys and values of a registry file
regfind.pl	search the keys, values, data, or types of a registry file for a matching string.

This module was updated on 28th September 2008 to version 0.40 and a number of new utility scripts added which are listed below.

regexport.pl	outputs registry keys in the same format as a Windows .reg file
regscan.pl	dumps all the entries in a registry file. This will include defunct keys and values that are no longer part of the current active registry.
regstats.pl	counts the number of keys and values in a registry file. It will also provide a count of each value type if requested.
regtimeline.pl	displays keys and values in date order.
regtree.pl	simply displays the registry as an indented tree, optionally displaying the values of each key.
regview.pl	is a GTK+ Registry Viewer. It offers the traditional tree of registry keys on the left hand side, a list of values on the right, and a hex dump of the value data at the bottom.

The last of these, regview.pl, needs Gtk2-Perl to be installed (GTK+, 2008).

From the GTK web site.

“GTK+ is a highly usable, feature rich toolkit for creating graphical user interfaces which boasts cross platform compatibility and an easy to use API”

Source code for this module was obtained with a view to analysing it to understand the view of the registry Files that it supports.

2.7.4.6 dosreg

This is a small registry walking utility, written in C by Nigel Williams (Williams, 2002).

From his web site

> "A DOS/Linux based NT4.0 Registry editor. This is basic code that illustrates the structure of NT4.0 Registry files. The initial structure was published on the net some time ago and is included at the top of the code. Portions were left out and some parts were incorrect. The gaps were filled in using NTICE. The whole program was completed in approximately three days."

This program is based on WinReg.txt (B.D., undated) and is notable for containing some useful annotations and corrections to that work.

The source code file regfio.c from the SAMBA project contains an acknowledgement that most of the information was obtained from dosreg and of the contribution made by Nigel Williams with the comment "Thanks Nigel!".

2.7.4.7 Regview.c (Dolan-Gavitt)

There is a regview.c program referred to by Dolan-Gavitt in his blog (moyix, 2007) which has been downloaded.

The source of this is obscure. It is designed to walk a registry structure and should not be confused with the other regview.c source which was written for a book on Windows CE Device drivers.

Neither of these should be confused with the regview.exe program distributed with the Windows 2003 Resource Kit Tools (Microsoft, 2003). This is a tool to view the Registry.pol files which in later versions of Windows Server (2003 onwards) are text files produced by the Group Policy Editor.

2.7.4.8 Regview.c (Windows CE)

There is a regview.c program which was written for the book on Programming for Windows CE (Boling, 2001). The code is reproduced by Hague (Hague, et al, undated).

This is a professionally written program. However since Windows CE is not a Windows NT Operating System it is not of relevance to this project (Microsoft, 2008f).

This regview program should not be confused with the one referred to by Dolan-Gavitt (moyix, 2007).

Neither of these should be confused with the regview.exe program distributed with the Windows 2003 Resource Kit Tools (Microsoft, 2003a). This is a tool to view the Registry.pol files which in later versions of Windows Server (2003 onwards) are text files produced by the Group Policy Editor.

2.7.4.9 RegUtils

These are a set of Registry Utilities produced by Michael Rendell of Memorial University, Canada (Rendell, undated).

These are all Windows 9.x utilities and so of no use to this project.

2.7.5 Other Sources

2.7.5.1 Advanced Forensics Course

During a Short Course at Cranfield University called “Advanced Forensics” the structure of the Registry was explained by Prof. Brian Jenkinson (Jenkinson, 2008). The material he presented was derived from his research and provides an excellent and detailed view of the Registry File structure. This material has been incorporated into this project

2.7.5.2 Other Researchers

Attempts have been made to engage with other researchers in this field. This proved to be of limited value and may represent an opportunity missed.

2.8 Registry Structure Quick View

From the above work a simple overall view of a registry file can easily be obtained.

> A Windows NT Registry File consists of a 4K header block which starts with the signature “regf” and which contains other information. Following that are a number of hbins, normally 4K in size but always a multiple of 4K in size, which have a signature of “hbin”. In each hbin, after the header, is a block of data which is made up of cells which represent the Keys and Values of the Registry. “Windows Internals” p 200-201 (Russinovich, 2005a).
>
> A *block* is by definition 4K in size “Windows Internals” p 200 (Russinovich, 2005a).
>
> “…a bin can contain one or more blocks…”, “Windows Internals” p 206 (Russinovich, 2005a).

Probert describes an attribute of a "Bin" as being "Size is increment of 4K" (he must have meant "multiple of"). "Windows Kernel Internals" p10 (Probert, undated).

So a bin or hbin is one or more 4K blocks.

> "Windows organizes the registry data that a hive stores in containers called cells. A cell can hold a key, a value, a security descriptor, a list of subkeys, or a list of key values". "Windows Internals" p 200 (Russinovich, 2005a).

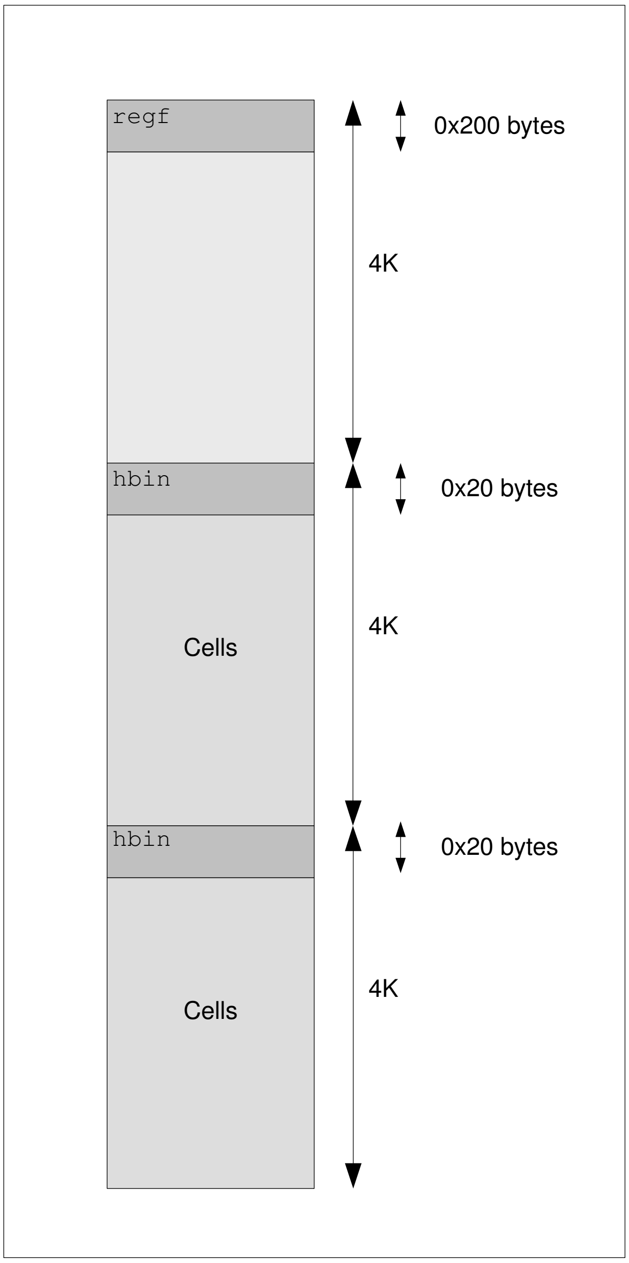
The hbins are a container for the cells, cells do not cross hbin boundaries.

> "When a cell joins a hive and the hive must expand to contain the cell, the system creates an allocation unit called a *bin*.", "…Bins also have headers that contain a signature, *hbin*, …" "Windows Internals" p 201 (Russinovich, 2005a). Probert describes a "Bin" as being a "Collection of cells". "Windows Kernel Internals" p10 (Probert, undated).
>
> "In file, on disk, each Registry Hive consists of two files. The PRIMARY and the .LOG file. The PRIMARY file holds the data, the .LOG file is used as a simple transaction log to ensure that updates that are interrupted can be completed on next start-up". "Windows Kernel Internals" p8 (Probert, undated).
>
> "PRIMARY grows in 256K increments – to avoid fragmentation". "Windows Kernel Internals" p8 (Probert, undated).

Other useful pieces of basic information that can be gleaned from "Windows Internals" (Russinovich, 2005a). and "Windows Kernel Internals" (Probert, undated) include the following

- Cell sizes are a multiple of 8 bytes, including size element.
- A negative cell size means that the cell is in use, a positive cell size means that it is free.
- Cells are referenced by a Cell Index which is the offset from the first hbin when in the Registry File
- Volatile cells (in memory only) have a Cell Index with the MSB (Most Significant Bit) set.
- Hbins have a header of size 0x20 bytes
- First 4K block has a signature of "regf" as part of a header which is of size 0x200 bytes.

Below is a simple diagram of a typical Registry File which is presented here so as to provide a basis and a stepping stone for more detailed analysis.



2.9 Summary

In this chapter the purposes of the Literature Search were briefly reviewed.

The various sources discovered during the Literature Search were reviewed.

It was shown that there are few academic resources currently available but there are a large number of informal sources some of which are very valuable.

A simple early description of the Registry was outlined with justifications.

Chapter 3 - Methods

The two main aims of the project are

1. Gain a good understanding of the structure of the registry. Understand existing sources of information, correlate and aggregate what is known. Identify and resolve conflicts and anomalies. Discover new information.

2. Do something useful, in a forensic sense, with that information. The initial project brief was to discover Registry data in other than the Registry but the direction of the Project was open to review.

In this Chapter the methods that were used will be explored.

These can be classified as follows.

Data Collection – examples are needed for examination
Standard Tools
Special Tools
In Memory explorations

3.1 Data Collection

Examples of Registry data are needed to analyse. The project needs to be fixed in reality.

The main considerations in accessing sources of data to analyse were.

- Availability
- Completeness
- Ease of use
- Relevance
- Validity

The obvious source is to collect Registry files from various systems. A method is needed for this and that aspect is explored later.

Once sufficient understanding of the Registry structure has been gained then examination of places where parts of the Registry might be found will be needed. For that pagefiles, disk images and possibly other sources of data will be needed.

3.1.1 Registry Files

Registry files are the obvious source of Registry data and so examples will be needed. If Registry data is found elsewhere than in Registry files, in the pagefile or in unallocated space in the file system for example, then it will be as fragments of a Registry file or as fragments of the registry as held in memory.

Consideration was given to the Registry files and other files that should be collected and the following was decided as the set that was needed. It was not always possible to get every one of each of these files from every donor computer.

From “%SystemRoot%\system32\config”

COMPONENTS (Vista Only)
BCD-Template (Vista Only)
DEFAULT
default.LOG
SAM
SAM.LOG
SECURITY
SECURITY.LOG
SOFTWARE
software.LOG
SYSTEM
system.LOG
userdiff
userdiff.LOG

From C:\Boot (Vista Only)

BCD

From “C:\”

pagefile.sys
hiberfil.sys

From “C:\Documents and Settings”, for each User

NTUSER.DAT
ntuser.dat.LOG

From “C:\Documents and Settings\%User%\Local Settings\Application Data\Microsoft\Windows”, for each User

UsrClass.dat
UsrClass.dat.LOG

A clean set of fresh Registry files was collected from a new build of Windows XP SP2. The system was built and then, having done the minimum to get the system built, the Registry files were extracted.

This produced a complete set of very clean, fresh Registry files which have not been coloured by any significant interaction with users or application programs.

This makes analysis of these example files as simple and straightforward as can be achieved. It also makes the files smaller which is an advantage.

The disadvantages of this approach is that Registry files which are relatively juvenile will not have the depth of change, churn and quantity of keys and data to be expected in a normal Registry file. This makes them untypical and lacking in depth.

To combat the disadvantages an effort was made to collect Registry files from a number of sources. The intention was to try to make sure that Registry files from a wider 'gene pool' were studied. No attempt was made to make sure that this sample was statistically valid either in terms of it being a representative sample or in terms of complete coverage of likely sources. Efforts were made to get files from a diverse set of computers.

The main focus of the project is on Windows XP on the basis that this Operating System is still the most commonly encountered during Forensic examinations. To be exact Windows XP SP2.

Care was taken to get at least one set of Registry files from each of NT4, Windows 2000, Windows 2003 Server and Windows Vista.

All these files were collected onto a CD-ROM with a folder for each Computer named Computer01, Computer02 etc. The User files were placed in a separate series of folders named User01, User02 etc regardless of which computer they came from. The mapping of the original sources and these folders was kept in case it should be needed.

Later in the Project, when a method had been developed, volatile hives and the volatile parts of hives were extracted from memory from one computer and saved as files. The two volatile hives were named "Registry" and "Machine", the volatile parts of hives were named as the file name for the stable part with the extension ".vol". These were added to the collection of files. Not all hives were found to have a volatile part.

Registry files from the companion DVD to the book Windows Forensic Analysis (Carvey, 2007c) were added at a later date.

Programs and scripts were developed to allow these hives to be pulled apart record by record, and loaded into an Access databases for further analysis by queries or by further scripts. This proved immensely valuable in providing a mechanism for checking across large quantities of data. These scripts produced one database file for each Registry File processed as well as one overall database of all the information. More on these scripts and programs later in this chapter.

In all 174 registry files were analysed from 18 computers and 40 users. Most of the files were from Windows XP SP2.

Care has been taken not to allow any of the information from these files to be leaked by this project, perhaps in examples or other ways. All the extracts shown that contain real data have come from the bare install, otherwise known as clean Registry files.

3.1.2 Registry File collection method

The Registry files of loaded hives are held open and locked. The hives of users that are not loaded can be accessed if the logged on user has sufficient access rights, normally by being a local administrator. However this is of little help as most of the files still cannot be accessed.

There are methods available for collecting these files by using the snapshot file capability provided in the Volume Shadow Service. This would not help with NT4 Registry files as this feature does not exist on that Operating System and it needs specially installing for Windows 2000 and XP pre-SP2. There is also a question mark over using this method as it is slightly obscure which might result in damaged file capture and/or a reduction in confidence.

Another potential method is to do a System State backup using NTBackup and then restore that to another location, perhaps on the same machine. The System State backup backs up the Registry, the COM database and those files needed to boot the system. It is not possible to do a partial restore of the System State data, it is all or nothing. This took about 3 minutes each to backup and to restore for about 6 minutes in total.

Disadvantages of this approach are that it does not restore all the desired files, the pagefile.sys, hiberfil.sys and the .LOG files are not made available. The System State backup/restore feature was introduced in Windows 2000 and it is not available for NT4. It was informative to see from the System State restore how many and what files are needed to ensure that the system could boot up.

If live acquisition is ruled out then that leaves a number of cold or offline access methods, where the computer is not booted to its native Operating System.

The Hard Disk could be imaged and then the files extracted using a tool such as EnCase from Guidance Software (Guidance, 2008). While this was feasible it was thought to be too unwieldy a method and too intrusive when co-operation from 3rd parties was being sought.

Another approach is to boot the machine to a floppy disk or a CD and then gain direct access to the computer hard disk in that way. There are a number of such bootable options available, some of them targeted at forensic use.

NTFS4DOS was looked at (NTFS4DOS, undated). This is available and free. It is a copy of DOS with NTFS drivers which enable read/write access to NTFS volumes. Using this would allow the machine to be booted to DOS and the files could then be copied to an alternative location on the machine in question. This was ruled out as the command line interface is rather fiddly and hence error prone and also because of a lack of confidence in software that writes to, rather than reads from, NTFS .

Helix 1.9 (Helix, 2008) was chosen as it is known to be have the needed capabilities and is familiar to the author. By default it mounts volumes as Read Only (which is safe) and it is known to be capable of writing to USB devices formatted to FAT32.

To receive the files an 80GByte USB drive was reformatted with 32GByte FAT32 volumes (32GByte FAT32 is a known limitation of Windows XP which was the Operating System on the machine that was used for formatting). This size limitation was not a restriction as none of the files being captured would be anywhere near that size.

A detailed protocol for capturing these files was developed and is described in Appendix 3.

3.1.3 Disk Images

One aim of the project is to see what can be recovered from an arbitrary block of data such as a Disk Image. This might allow for otherwise inaccessible evidence to be collected and bought to bear on a case.

To test possible methods some disk images were needed. A small number of these were collected for use in the project. Size was a factor, smaller the better, as was the age of the image; older images will have more to look for.

Two images were collected. One was from a Windows XP computer, the other was from a Windows 2003 Server. This is not a large sample size but was felt to be sufficient and manageable. Time did not allow either of them to be used.

3.2 Standard Tools

Whenever doing any research work the researcher (or investigator) will always need basic tools which can be relied upon, tools which people have confidence in. This is both to provide a solid basis for the work and to help convince others of the validity of the work. It is also needed to ensure reproducibility.

What follows is a discussion of the Standard Tools which were used on this project.

3.2.1 WinHex

WinHex (X-Ways, 2008) is a well known and respected tool for examining files or images in terms of their binary data which is shown in hexadecimal format. The program runs on Windows and these two facts plainly are the derivation of its name.

3.2.2 RegEdit

The Windows XP version of regedit.exe was used extensively to cross check what was being seen with WinHex and other ways of looking at the Registry and correlating those views with what was happening with the data.

In Windows XP, effectively, the programs regedit.exe and regedt32.exe are the same, regedt32.exe being merely a stub program that calls regedit.exe.

Careful experiments with regedit to make changes and WinDbg to see what changes this caused in the data held in memory, and vice-versa, were used to derive understanding about what parts of the Registry data structures were used for what purposes. Care was needed to only make changes that would not crash the machine being worked on and to guard against buffering and masking by the interposing layers.

Details of these experiments can be found in Chapter 4.

3.2.3 WinDbg

From the blog article “Enumerating Registry Hives” (moyix, 2008a) the idea of examining Windows memory on a live machine using the WinDbg tool was discovered.

WinDbg (and its command line companion kd.exe) can be used to examine and debug items of the Windows Kernel while a computer is running.

Providing that the right Symbol files are made available the proper Microsoft definition of internal data structures can be shown.

These two facts make this a very powerful tool for understanding exactly how Registry hives are stored in memory and how they are accessed and used. It gives us a great way of getting ‘hands-on’ with the problem.

These debugging tools are free and available for download from Microsoft. They need installing and the symbol files access needs setting up properly to get best use of this technique. A small amount of knowledge is needed to actually gain access to the Kernel of the current machine.

How to install the debugging tools is explained in Appendix 4.

3.3 Special Tools

During the course of the Project some special tools were used or developed. Some of these, it is hoped, will be of use to others in their research or investigations.

Part of the aim of this project was initially to provide such a special tool. It is perhaps to be expected that as the problem space is explored that solution space ideas will emerge.

WARNING: The programs are presented in all good faith and are believed to be correct and fault-free. However a sensible degree of caution should be taken in their use and results verified.

3.3.1 Filter Symbols VBScript

A VBScript program called “Filter Symbols.vbs” was written to filter the list of possible symbols obtained from the ntoskrnl.pdb files. This method is explained later but suffice to say it produced a large list of potential names for internal structures.

A characteristic of these symbol names is that they all consist of just the characters A-Z (upper case) the digits 0-9 and the underscore character. The “Filter Symbols” script was written to filter the list of possible structure names into a list which met the above criteria. The script is quite simple, it always takes its input from a file called “a.txt” and always puts its output in a file called “b.txt”.

The script is shown as Supplement 1.

3.3.2 Hex2Binary Program

This program, written in Visual Basic 6, takes a text hex output, such as shown by WinHex, and converts it back to a binary file.

The motive for doing this is that it allows WinDbg to be used to show sections of computer memory in a hex list, like WinHex. This can then be cut and pasted into a text file and then, from that a binary file can be created.

In this way a registry hive can be exported directly from memory and turned into a file. Practically this can only be done manually for fairly small hives but the process could be automated for larger hives.

Once a hive has been reconstructed like this it is then amenable to analysis by any tools designed to look at registry files.

The source code for this utility is shown in Supplement 2.

Use of this script to extract Hives from memory is discussed in the next chapter.

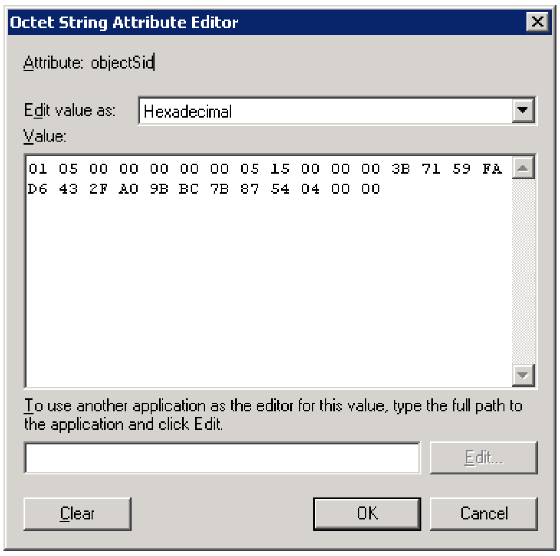
3.3.3 Hex2SID VBScript

Use was made of a VBScript program that the author wrote some years ago.

This takes a SID expressed as a text hex string and converts it to a SID in the familiar format that starts, normally, “S-1-5-“. The method for doing this is explained in the MSDN blog “The Old New Thing” (Chen, 2004).

This was originally written to take SIDs as shown by the Active Directory ADSIEdit utility. ADSIEdit allows Active Directory to be examined in close detail. It is perhaps the regedit of the Active Directory world (Microsoft, 2003b).

User Account objects have a number of attributes as do all Active Directory objects. One of these is the objectSID which is the SID of that user. In ADSIEdit this is shown as in this picture.



Attribute: objectSid

Edit value as: Hexadecimal

Value:

```
01 05 00 00 00 00 00 05 15 00 00 00 3B 71 59 FA
D6 43 2F A0 9B BC 7B 87 54 04 00 00
```

To use another application as the editor for this value, type the full path to the application and click Edit.

The hex bytes shown in text can be cut and pasted into Hex2SID which then converts it into the familiar SID format. In this case “S-1-5-21-4200165691-2687452118-2273033371-1108”

This script was of great help in unravelling the structure of the “sk” cell type and the NT Security Descriptor it holds, as described later.

The script is shown in Supplement 3.

3.3.4 RegHoover and HooverLoad Programs

It was determined that a tool was needed to examine the large amount of data that was available in the sum of all the collected Registry Files. Although there is a place for detailed examination of small quantities of data it is also necessary to be able to examine large quantities of data in order to see patterns, anomalies and differences as well as to validate what understanding has been gained.

To this end it was proposed to create a program called RegExploder that would process a Registry File and turn each cell or record into an HTML page. A specification was drawn up and is shown as Supplement 4.

This was considered too complex and time consuming and so a simpler program was written to process a Registry File as records and to load the data into a database for further use. The program was named RegHoover.

RegHoover is a “Ripper” type program as opposed to a “Traverser” type program in that it simply processes a file as a concatenation of cells or records. It makes no attempt to link the records together or to follow the interlinked structure of the records.

The Language that was chosen for this was plain ANSI-C using Microsoft Visual Studio 6. The reasons for this choice were the authors skill level, the ability of C to deal easily with binary data and the availability and capability of the development tools.

Microsoft Access was chosen as the database as it is sufficiently powerful, widely used and understood and (run-time at least) built into modern versions of Windows.

It is not possible (at least easily) to interface to Microsoft Access from ANSI-C and so a two stage process was adopted. The C program RegHoover.exe analyses the Registry File or files and produces an intermediate text file (called RegHoover.txt). The VBScript HooverLoad.vbs then takes this file and loads it into an Access Database.

RegHoover was able to process all of the Registry files collected including those from Windows NT4, 2000, Server 2003 and Vista.

Descriptions of how to run these program can be found in Supplements 5 and 6. The source code can be found in Supplements 7 and 8.

3.3.5 RegHoover Database

A database was developed to hold the information produced by the RegHoover program.

The ‘clean’ empty version of this database is held in a file called “Clean RegHoover.mdb” which is copied to a filename of RegHoover.mdb when a new database is being made.

The schema for this database can be found in Supplement 9.

Some twenty-five queries were developed to aid navigating the Registry data in the database. These are documented in Supplement 21.

3.3.6 BulkHoover and BulkHoover2 Scripts

It was necessary to find a way of analysing large numbers of files automatically.

The BulkHoover.vbs script was developed to do this. It takes as its starting point the drive X: and then processes all the folders under that drive, going down one level only.

For each folder it looks for a fixed set of Registry file names and for each one it finds it runs RegHoover and HooverLoad to produce a RegHoover.mdb file. This file is then copied back into the source folder with the name of the Registry File prepended.

The use of the X: drive is to ease configuration. The intention is that the Windows command line utility subst be used to substitute X: for the path where the Registry File folders are located.

A variation of this script was written called BulkHoover2.vbs. This does not produce a database file for each Registry File but instead loads all the data into one single database.

Both of these scripts can be run at the same time but from different folders.

These scripts can be found in Supplements 10 and 11.

3.3.7 Daily Analysis Cycle

The programs to analyse Registry Files, BulkHoover, BulkHoover2, RegHoover and HooverLoad, were run, repeatedly, overnight on a large set of Registry Files. The set eventually consisted of some 174 files obtained from 18 Computers and 40 users. The process hence yielded 174 database files for individual Registry files and one other for all the files.

Each morning the results and the source files were copied onto DVD to fix the data and free the computer and disk space. Eventually this was enough to fill a DVD, that is about 4.5GB of data. During the day the data was analysed, discoveries made and errors and avenues of investigation or improvement identified. The programs used were then improved or Registry files added in time for the following nights run.

Each run took over 12 hours and 16 cycles were run so amounting to perhaps 200 hours of processing. To allow more precision in discussing experiments and their results each overnight run was given a consecutive run number as in “Run 1”, “Run 2” and so on.

It is hoped that this approach will be of use to other researchers who can use these tools to build their own databases of Registry data.

3.3.8 PrintSchema Script

To help document the RegHoover database in a way that fitted with the needs of this project, a script was developed to extract the Schema information and print it in a simple and easy to understand way.

This was of great help in making sure that all tables had appropriate Primary Keys and Indices.

This script is shown in Supplement 12.

3.3.9 Other Scripts and Programs

A number of other scripts and programs were written to examine various aspects and theories about the data. They proved to be useful experimental tools as the scripts were modified and refined in the light of increased understanding and knowledge.

They are listed in this table; the source code is in the Supplement shown.

Script Name	Purpose	Supplement
CheckMaxLengths.vbs	Check theory about the ‘Max’ values in nk records	13
HashCheck.exe	Check Hash function used in lh records	14
TraverseData.vbs	Check that all parts of the analysed Hive in the database are reachable	15
Number.vbs	Used to Number Source Code files as a step in documenting them.	16

3.3.10 Documenting Source Code and Scripts

A method was needed to document Source Code and Scripts for this report. The method used is explained in Supplement 20.

3.4 In Memory Examinations

The blog “Push the Red Button” article “Enumerating Registry Hives” (moyix, 2008a) introduces the idea of using the Windows Debugger WinDbg with the Debugging Symbols for ntoskrl.exe to explore internal memory structures used by windows.

The blog focuses on understanding live memory structures in order to find them more easily in memory dumps collected from a Live Forensics intervention. The focus of this project is with traditional offline forensic examinations but the same rationales hold. It is also true that the pagefile will contain data that used to be in memory.

The next few sections described the methods that were used.

3.4.1 Displaying Kernel Structures in Memory

The blog (moyix, 2008a) describes how the dt command can be used to show the Microsoft definition of internal data structures using information from the appropriate symbols file.

An example command to do this is "`dt nt!_CM_HIVE`". This will display the structure of the `_CM_HIVE` object using information from the symbols file.

An obvious restriction of this approach is that you need to know the name of the objects before you can display them. Some names are given in the blog article and from displaying those structures the names of other structures can be seen and so displayed.

3.4.2 Discovering Kernel Structures

The Registry is organised by the Configuration Manager "The Windows kernel-mode configuration manager manages the registry" (Microsoft, 2008h).

The Kernel is implemented in ntoskrnl.exe, this means that the symbol file for the Kernel, and hence for the Configuration Manager is called ntoskrnl.pdb.

Attempts were made to extract these symbols from the PDB file to see if they would reveal more about the Registry structures. Full details of these attempts are provided later in Chapter 4.

3.4.3 The !reg Extension

The debugger (WinDbg and kd.exe) allow debugger extensions. These all start with an exclamation mark (!) and then the name of the extension.

There is an extension specifically for the Registry and it is called "!reg".

Issuing the command !reg to the debugger will list all of the commands available. These allow a list of all the Registry hives currently loaded in memory to be displayed and for a number of other useful commands.

This feature was first learnt about in the article "Challenges in Carving Registry Hives from Memory" (moyix, 2007).

3.4.4 Other Debugging Commands

Some common debugging commands (Microsoft, 2008i) have been found to be useful.

dd <address>	Display in Hex as Dwords (32 bits)
dw <address>	Display in Hex as Words (16 bits)
db <address>	Display in Hex as Bytes (8 bits)

db <address> L<n>	Display n elements eg db e1234567 L100 to display 256 (decimal) byes
eb <address> <value>	Edit Byte. Set byte at <address> to <value>
dt nt!<structname>	Display structure information
dt nt!<structname> <address>	Display structure information with data from <address>
lml	Lists which Symbols are loaded , gives location of each Symbol file

3.5 Code Examination

An obvious and valuable way of understanding the structure of the Registry is to understand those programs and utilities that directly access Registry files.

All the utilities that were found are written in either C, Perl or Python. Both Perl and Python are somewhat C like in their syntax although both are at a much higher level of abstraction.

These were unravelled by straightforward code examination. Paying attention to the comments was useful as was paying attention to discrepancies between the comments and the code.

To make sure that the code was understood the utilities were run and the code was examined to see how the output was produced. In this way the code examination was confirmed.

To aid this process Windows Visual Studio (Microsoft, 2008g) was used to trace through the execution of some parts of C code.

ActivePerl from ActiveState was used as the Perl interepter/compiler (ActiveState, 2008a).

A free Perl IDE (Integrated Development Environment) called Perl Express (Perl Express Group, 2007) was used for some Perl code examination.

Better results were obtained with the Komodo IDE (ActiveState, 2008b) perhaps as it comes from the same stable as ActivePerl.

The Python utilities were not looked at closely as they were small in number and were assessed as being either peripheral to the main interest or re-writes of other utilities.

The starting point for this work was to run the regdump.pl script from Parse-Win32Registry (Macfarlane, 2008) against a copy of a SAM Registry file. The code was then traced while looking at the registry file with WinHex to see how the program produced its output.

3.6 Record Templates

Simple record templates were produced using Word. These were each laid out as a table with columns for element sizes, offsets, hex bytes and a final column for the interpretation of the data. The final versions of these are shown in Appendix 5.

These template sheets were completed manually while working through a SAM Registry file using WinHex.

Once a few of these had been produced they were physically laid out on the floor to visually show how the Registry records fit together.

During the initial stages of understanding the Registry structure these templates were found to be very useful as a method for building knowledge and allowing visualisation. They provided statements of knowledge that could be improved and refined and they were a useful way of exploring new parts of the Registry structure.

Later in the project they were valuable as a reference source and as a manual method of decomposing an area of data.

3.7 Visio Diagrams

Microsoft Visio was used to produce diagrams of how data structures fit together and for other explanations.

Producing relevant diagrams was found to be great help towards understanding and a good way of cementing knowledge.

3.8 Anomaly Resolution

A specific effort was made during the project to identify any discrepancies anomalies or missing pieces of understanding which emerged from the Literature Search or any of the examination methods used.

The first priority was to identify such items of interest. A list of these was maintained during the project. The second priority was to attempt to resolve these items.

In this way a reliable and refined definition of the Registry structure, and how it is managed, was obtained.

Not all items could be resolved. Those that were not are left as items for further work and described later in the Conclusions.

3.9 Summary

In this chapter the methods used to exploit the information discovered during the Literature Search phase were examined as were other methods of gaining knowledge. These methods included collecting and examining Registry files and using the Windows Debugging Tools. Various custom programs written for the project were explained.

It was shown that these methods could be classified as follows.

Data Collection
Standard Tools
Special Tools
In Memory explorations

Chapter 4 – Experiments

In this chapter various experiments are described.

These use the methods described in “Chapter 3 – Methods” and are all designed to in some way improve either knowledge, facts, or understanding.

Some of the experiments are specifically targeted at resolving anomalies, discrepancies and gaps in knowledge.

The aims of the experiments are twofold

1. Improve understanding
2. Look for opportunities for producing programs that will be of forensic benefit.

This Chapter provides enough detail to allow these experiments to be repeated and/or verified. It is not sufficient merely to assert that something is true, it must be shown to be so.

As far as is possible much of the detail of these experiments has been put into Appendices (for smaller and more valuable information) or into Supplements (for larger amounts of information of less value). The Appendices form part of the printed report, the Supplements are only available on the companion CD.

4.1 Examining Parse-Win32Registry

The Parse-Win32Registry module is a library module created by James Macfarlane (Macfarlane, 2008). It is mentioned by Carvey in many of his blog postings (Carvey, 2008a) and used in some of his contributions such as the RegRipper utility (Carvey, 2008b) and the SAMParse utility (Carvey, 2007a). It is referenced by Dolan-Gavitt (Dolan-Gavitt, 2008).

The benefit of understanding how this code analyses the Registry is that it is a mature and well used piece of software which is in contemporary use. This gives it authority and authenticity. If it was getting it wrong then it would not be so widely used and its deficiencies would be commented on. The tools that have been developed using this module as a basis would not be viable. This is not to say that this module must therefore be perfect, merely that it must make a reasonable job of understanding the structure of the Registry files.

It comes with some utility scripts which make use of the module and these can be run against Registry files. We can see for ourselves whether or not the output is plausible and verify portions of it. Complete, 100% verification is not easy due to the sheer volume of data. Comparison with the output from other tools is possible although this does raise the problem of reconciling different output formats.

Code Examination of this module was done by a multi way approach.

1. Manually tracing a hardcopy print out of the code
2. Examining the output of a run of the regdump.pl Perl script against a SAM Registry file
3. Using WinHex to examine the hex of the SAM Registry file
4. Using RegEdit to examine the SAM Registry file
5. Completing Registry Record template sheets

All these sources of data were used concurrently in order to build a picture of how the records sit together. The template sheets were both an output and an input as they were used to build understanding.

The intent was to cross-check understanding and gain a consistent view and hence gain confidence that the view was valid.

The regdump.pl source code and the library modules that it used were traced to understand the path and method of execution and to pick up any relevant comments. This code uses the Perl unpack() function to extract values from blocks of data (Allen, 2008).

Initial work was done on version 0.30 of this module. Version 0.40 was released in September 2008. Version 0.40 was examined and nothing was found which would add to or change the understanding of the Registry structure found from version 0.30.

The structure of the modules is shown in Appendix 6 for version 0.30 and 0.40.

A detailed analysis of this module, how it works and the information that can be extracted is shown in Appendix 7.

4.2 Template Sheets

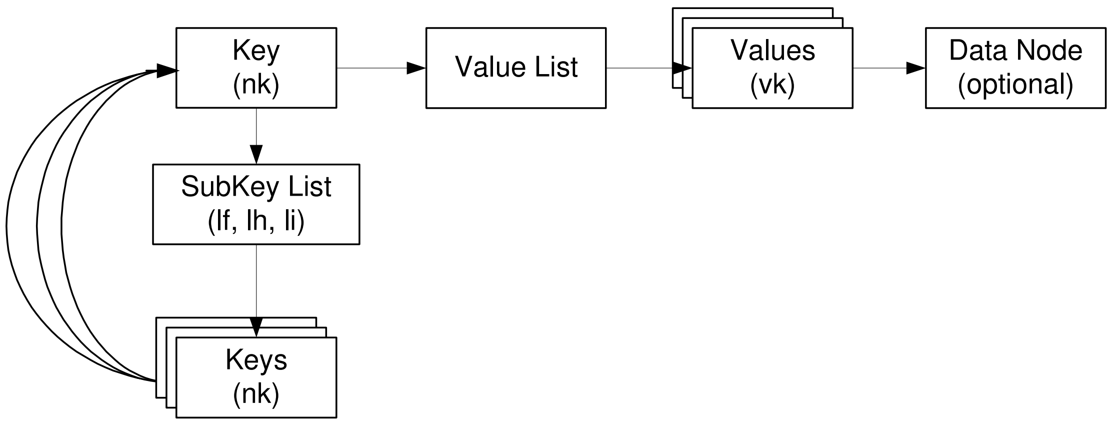
The information gained from the above studies was used to create a set of Template Sheets for the known record types.

As the execution of regdump.pl was traced a Template Sheet was completed for each record. This enabled a pattern to emerge.

Once about thirty of these had been completed they were laid out on a large enough piece of floor and from this the structure of the Registry became easily understood.

This was found to be as shown here.

Basic Diagram of the Registry



The above shows that each registry key record ("nk") has the following.

- If it has any values, a pointer to a Value List. The Value List in turn points to a number of Value records ("vk"). If the Value is more than 4 bytes in length then a Data Node is used merely to hold the data.
- If it has any Sub Keys, a pointer to a SubKey List. The SubKey List in turn points to a number of SubKey records which are of course key records ("nk").
- Each subkey record has a pointer that points back to its Parent key record.

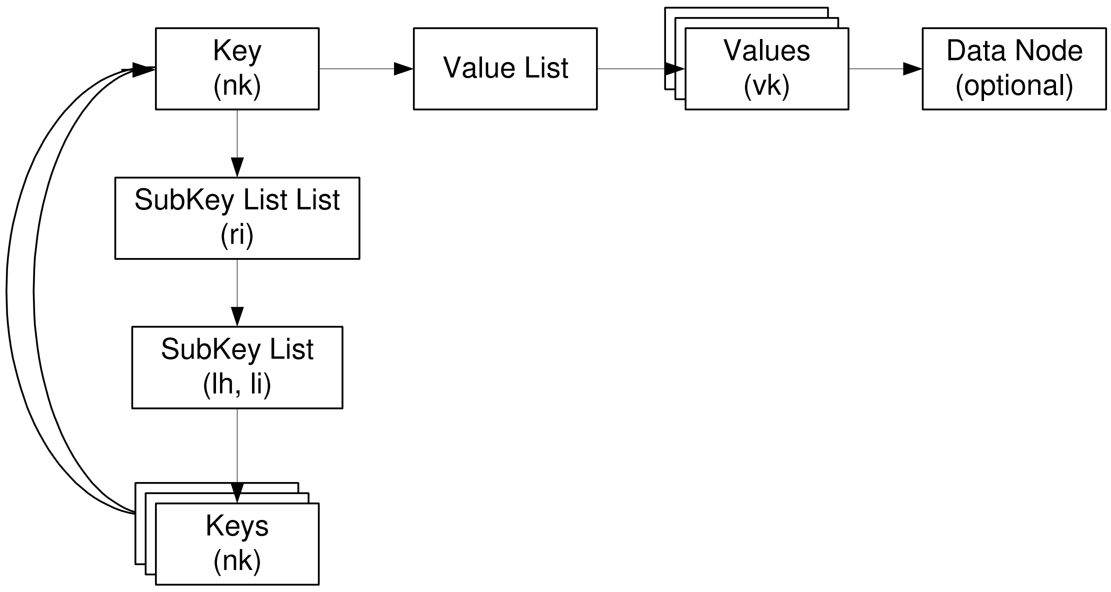
Notes

1. Class and "sk" links not shown
2. The count of the number of Subkeys in the Subkey List is held in both the "nk" record and the Subkey List. The count of the number of Values is only held in the "nk" record.
3. "ri" Subkey List not shown. These are lists of Subkey Lists.

From analysis of the Parse-Win32Registry module it would seem that the "ri" SubKey List is in fact a list of SubKey Lists. It may be that these are recursive in that an "ri" cell may point to another "ri" cell but this has not been verified.

With the "ri" records, the structure looks like this.

Basic Diagram of the Registry (with “ri” SubKey List)



It is shown later that “ri” lists only point to “li” or “lh” lists.

4.3 Record Structures

From the above analysis the following definitions of the various records can be deduced. We can expect these to be initially incomplete both in terms of the fields in the records and the number of record types. It is however a firm starting point.

In these tables the Name is normally the variable name used in the Parse::Win32Registry code and the Notes is usually the relevant comment in the code.

There are 10 record types, all but two have a signature, the two that do not have been named as “Value List” and “Data Node”.

Three record types are not seen here. These are the “sk” record type for holding security information, whatever data structure holds the Class Name information and the “lk” records which are rarely reported on. More on these later.

“regf”			
Size	**Offset**	**Name**	**Notes**
4	0x00	regf_sig	“regf”
8	0x0C	timestamp	Windows FILETIME
4	0x24	offset_to_first_key	aka the Root Cell
64	0x30	embedded_filename	Zero Terminated Unicode

“hbin”			
Size	**Offset**	**Name**	**Notes**
4	0x00	sig	“hbin”
4	0x04	offset_from_first_hbin	
4	0x08	size_of_hbin	

“nk”			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	Always “nk”
2	0x06	node_type	
8	0x08	timestamp	Windows FILETIME
4	0x14	offset_to_parent	
4	0x18	num_subkeys	
4	0x20	offset_to_subkey_list	lf, lh, ri, li
4	0x28	num_values	
4	0x2C	offset_to_value_list	
2	0x4C	name_length	key name length
var	0x50	name	

“lf”			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	“lf”
2	0x06	num_entries	number of entries
4	0x08	offset	offset to 1st subkey
4	0x0c	str	first four characters of the key name
	“offset” and “str” are repeated to make up num_entries sets of entries		

“lh”			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	“lh”
2	0x06	num_entries	number of entries
4	0x08	offset	offset to 1st subkey
4	0x0c	hash	hash of the key name
	“offset” and “hash” are repeated to make up num_entries sets of entries		

“ri”			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	“ri”
2	0x06	num_entries	number of entries
4	0x08	offset	offset to lf/lh/li record
	“offset” is repeated to make up num_entries entries		

"li"			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	"li"
2	0x06	num_entries	number of entries
4	0x08	offset	offset to 1st subkey
	"offset" is repeated to make up num_entries entries		

Value List			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
4	0x08	offset	offset to 1st value
	"offset" is repeated to make up the number of entries shown in the "nk" record.		

"vk"			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	sig	"vk"
2	0x06	name_length	value name length
2	0x08	data_length	length of data
4	0x0c	offset_to_data	offset of data
4	0x10	type	type of data
2	0x14	name_present_flag	flag
2	0x16	?	
var	0x18	name	value name
	Note 1	If LSB of name_present_flag is set to 1 then the Value has a Name.	
	Note 2	If the MSB of data_length is set to 1 then the offset_to_data is the data. This is used to store values of up to 4 bytes in length inline instead of using a Data Node.	

Data Node			
Size	**Offset**	**Name**	**Notes**
4	0x00	size	size (as negative number)
2	0x04	data	may not run to end of cell

4.4 HBIN Structure

The term hbin is surely an abbreviation of “Hive Bin”. An hbin is merely a container, like a bucket, for the cells within it.

The structure of HBIN records is given in “Windows Kernel Internals” p10 (Probert, undated).

It is shown here as a table with comments.

Size	Offset	Name	Comment
4	0x00	Signature	“hbin”
4	0x04	FileOffset	Offset from start of hbins to this hbin
4	0x08	Size	Size of this hbin (not offset to next)
8	0x0C	Reserved	
8	0x14	Timestamp	Windows FILETIME
4	0x1C	Spare	

Probert also describes a Bin as being a “Collection of cells”. This implies that cells do not cross hbin boundaries and from observation that appears to be the case.

It is a very common misconception that the hbins are chained or linked and that the size field is the offset to the next hbin. Probably from this error comes the incorrect statement that the last ‘offset’ is set to zero.

Titheridge (Titheridge, 2008) correctly states that the last hbin also has a value in the size field which is correct as it is a size not an offset.

Of course this is, at one level, a minor semantic difference as an offset would be the same value as the size except for the last one.

Probert adds to this confusion when he states (p8) that the Registry File header is “Followed by chained Bins” although he might have meant that they were contiguous. Winreg.txt (B.D., undated) states at line 429 that this field (the size field) is “Offset to the next hbin-Block” and this may be the source of all the subsequent confusion.

If these are not chained then how do we know when we have reached the end of the hbins? The way to tell is by using the Length field contained in the Registry File header (“regf” block) which is also known as the BaseBlock.

The other common misconception about hbins is that they are all 4K in size. They are in fact a multiple of 4K in size. “Windows Kernel Internals” p10 (Probert, undated) states “Size is increment of 4K” (he must mean multiple when he says increment). It is true that the vast majority of hbins are 4K but if that was a limit then that would limit the size of Key names, the size of Value names, the size of Values and the number of Subkeys and Values that a Key could have. (Key names are limited to 255 characters so that would not be an issue. (Microsoft, 2008a)).

By observation, only the first hbin has the Timestamp set.

This is a partial description of this structure, a complete analysis is given later.

4.5 Making Test Registry Files

To enable the working of the Registry to be investigated it was thought useful to have some very simple test Registry files. In particular a file with an hbin of more than one 4K block was sought in order to be able to investigate the internal workings of the registry when in memory.

Four test files were created

TEST	A basic Registry File with one Key Node (“nk”) record, one Key Security (“sk”) record and one free cell.
TEST2	A simple Registry file that has an hbin bigger than 4K
TEST3	Same as above but a cleaner version.
TEST4	A Registry File with a Key Security (“sk”) record that has some Auditing set.

Appendix 8 explains in detail how each of these test Registry files was constructed.

4.6 The Registry in Memory

It will be recalled from Chapter 2 that methods were discovered from the blog article "Enumerating Registry Hives" (moyix, 2008a) of using the Windows Debugger WinDbg (Microsoft, 2008k) to examine the Registry in memory.

Various experiments were tried to get information about the Registry, some were repeats or extensions of experiments in the blog "Push The Red Button" (moyix, 2008f) and some were new and of the authors contrivance.

To carry out this work the Windows Debugger tools were downloaded and installed. Full details are given in Appendix 4.

The Registry holds configuration information and so it is completely natural that the part of Windows that has responsibility for the Registry is called the "Configuration Manager", Windows Internals p197 (Russinovich, 2005a).

A number of in-memory experiments were carried out on a standard Windows XP SP2 desktop.

Information discovered about structures was verified by comparing it with other sources. In particular this information was compared with

- what was learnt from the analysis of the Parse-Win32Registry module
- "Windows Internals" (Russinovich, 2005a)
- WinReg.txt (B.D., undated)
- dosreg.c (Williams, 2000)
- the web page "Beginning to See the Light" (Clark, 2005)
- "Windows Kernel Internals" (Probert, undated).

4.7 The structure of the Base Block

The following command, from the article "Enumerating Registry Hives" (moyix, 2008a) , was run.

```
dt nt!_CMHIVE
```

The output of this can be seen in Appendix 9.

Some of this is quite obscure but it can be seen that the first 0x210 bytes of the _CM_HHIVE structure is made up of the _HHIVE structure. This was similarly displayed with the command

```
dt nt!_HHIVE
```

The output of this can be seen in Appendix 10.

Again some of this is quite obscure but it can be seen that at offset 0x24 is a pointer to a structure of type _HBASE_BLOCK. This can be similarly displayed with the command

```
dt nt!_HBASE_BLOCK
```

Which produces this output

```
   +0x000 Signature        : Uint4B
   +0x004 Sequence1        : Uint4B
   +0x008 Sequence2        : Uint4B
   +0x00c TimeStamp        : _LARGE_INTEGER
   +0x014 Major            : Uint4B
   +0x018 Minor            : Uint4B
   +0x01c Type             : Uint4B
   +0x020 Format           : Uint4B
   +0x024 RootCell         : Uint4B
   +0x028 Length           : Uint4B
   +0x02c Cluster          : Uint4B
   +0x030 FileName         : [64] UChar
   +0x070 Reserved1        : [99] Uint4B
   +0x1fc CheckSum         : Uint4B
   +0x200 Reserved2        : [894] Uint4B
   +0xff8 BootType         : Uint4B
   +0xffc BootRecover      : Uint4B
```

"Windows Internals" p200 (Russinovich, 2005a) states

> "The first block of a hive is the *base block*. The base block includes global information about the hive, including a signature – *regf* – that identifies the file as a hive, updated sequence numbers, a time stamp that shows the last time a write operation was initiated on the hives, the hive format version number, a checksum and the hive file's internal file name…"

All of these elements can be seen in the above data structure.

Probert p9 (Probert, undated) has a diagram of a Registry File which calls the first 4K block of the file the "HIVE HEADER (HBASE_BLOCK)". This page also gives an abbreviated structure for the header which matches, as far as it goes, what we have above.

The "regf" definition extracted from the Parse-Win32Registry matches the above structure as far as those items that are present. There are no conflicts.

So it seems we have the definitive structure of the Registry File header.

4.8 Base Block values

4.8.1 Version Number

The BaseBlock has a Major and a Minor data element which is surely the Version Number. The only values seen in these items were "1.3" and "1.5". This was exhaustively checked with the following query against the RegHoover database of all files.

```
        SELECT DISTINCT Major, Minor FROM BaseBlock
```

Only the following four file names were V1.5 and then only in Windows XP and above. In Windows NT4 and 2000 these files are version 1.3.

```
        default
        software
        system
        userdiff
```

The new Registry files introduced with Vista are all V1.3.

4.8.2 Type and Format

The values of Type and Format were similarly investigated and the only values found in the database were 0 and 1 respectively.

An experiment was carried out to determine the possible values and names for those values, details of this experiment can be found in Appendix 26.

The following were found to be valid names and values.

Type

Name	Value
HFILE_TYPE_PRIMARY	0
HFILE_TYPE_LOG	1
HFILE_TYPE_EXTERNAL	2

Format

Name	Value
HBASE_FORMAT_MEMORY	1

4.8.3 Sequence Numbers

The BaseBlock sequence numbers are used to make updates to the Registry file into transactions which can be reapplied in the event of failure as explained in Section 4.31 and Appendix 30.

A side-effect of this is that they show how many updates have been applied to the Registry file.

Of the 165 files in the test set that have BaseBlocks (some files were volatile parts of hives) there were 149 which had a sequence number of less than 10,000 and 106 which had a sequence number of less than 1,000. Only 2 BaseBlocks had a sequence number of more than 1,000,000.

This demonstrates the relatively juvenile nature of these files which is a weakness in this project.

4.8.4 RootCell

This gives the Cell Index of the first or Root Cell of the Hive. That is the Key cell or Node (“nk”) that is at the top of the tree structure.

In the data examined this was invariably set to 0x20.

4.8.5 Length

From work which is detailed in Appendix 35 it is plain that it is the sum of the sizes of all the currently used hbins. In other words the length of the hbin part of the file when the hive is in a Registry hive.

Some files were found with valid hbins in the file beyond this Length value. This is both an opportunity and a risk. If you are looking for deleted data then you will want to search these, even the cells that are not apparently free. On the other hand if you want to restrict the bounds of your search to the currently valid data then the Length value must be used.

4.8.6 Other BaseBlock Values

The Cluster value was always set to 1. It is not known what this is used for.

The BootType and BootRecover values were always zero. These may only ever exist in memory as they are not in the first 0x200 bytes of the BaseBlock. BootType may be to do with abnormal startup such as booting into Windows Recovery Mode. BootRecover may show when a registry LOG file has been replayed or some other recovery action taken on Hive load.

4.9 Registry File Sizes

In “Windows Kernel Internals” p8 (Probert, undated) it states that file sizes grow in 256K increments to avoid fragmentation of the file.

Therefore you would expect that the minimum size of a Registry file would be 256K. This is not the case with NT4 and Windows 2000 which suggests this change was introduced in XP. It is not true of “`ntuser.dat`” Registry Files.

In one, possibly anomalous, case an XP SP2 “`default`” Registry file was found to be less than 256K in size.

Oddly, the new BCD Registry file in Vista does not comply being 24K in size although the BCD-Template Registry file which presumably is used to build it does comply.

It was found that all Registry files, including from NT4 and Windows 2000, which were more than 256K in size had a size that was a multiple of 256K so in that sense it is true.

4.10 The Hive Linked List

The in memory linked list of Hives was explored. From “Enumerating Registry Hives” (moyix, 2008a) this starts with a variable called “CmpHiveListHead” which can be interrogated in WinDbg by giving the command

```
? CmpHiveListHead
```

The value given here can then be dumped from memory using a command such as

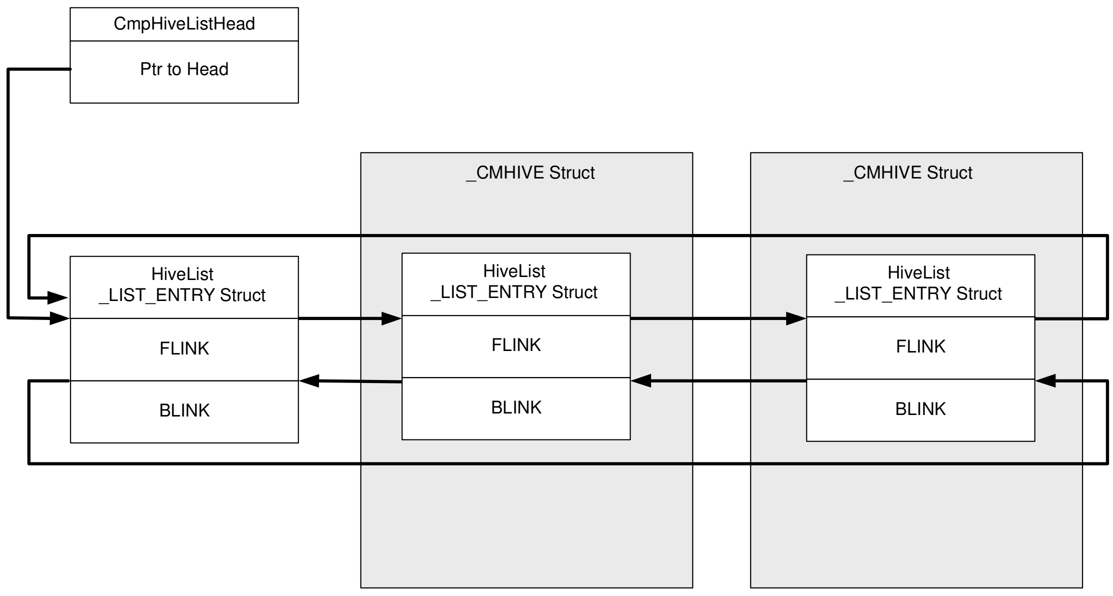
```
dt nt!_LIST_ENTRY <addr>
```

This will show the _LIST_ENTRY structure and the values it contains. It can be seen that we have two elements called FLink and Blink. It seems plain that these are “Forward Link” and “Back Link”.

The forward links were followed until it repeated itself showing that all elements of the list had been visited. The details of this experiment are given in Appendix 11.

The structure discovered can be seen in this simplified diagram which only shows two Hives.

<u>Registry Hives in Memory</u>



The CmpHiveListHead variable is simply a pointer to the List Head _LIST_ENTRY structure. In a sense the job of the List Head is to point to itself when the list is empty.

From the List Head all the other nodes are linked both forward and back.

Each _LIST_ENTRY structure that is embedded in a _CMHIVE structure will be at offset 0x224 from the beginning of the _CMHIVE structure. Therefore the address of the _CMHIVE structure can be obtained by subtracting 0x224 from the address of the _LIST_ENTRY structure.

4.11 The _CM_KEY_NODE structure

From “Cell Index Translation” (moyix, 2008d) comes this extract.

> “…large amount of information on internal registry structures already exists in the public symbols distributed by Microsoft. Here are the types I know of that correspond to hive data structures:
>
> - _CM_KEY_NODE
> - _CM_KEY_VALUE
> - _CHILD_LIST
> - _CM_KEY_INDEX
> - _CM_KEY_SECURITY
> - _CM_BIG_DATA”

It should be noted that these mainly start _CM which surely stands for Configuration Manager.

The first item looked at was the _CM_KEY_NODE. The structure of this was extracted by issuing the WinDbg command

```
dt nt!_CM_KEY_NODE
```

This is the resultant output

```
+0x000 Signature          : Uint2B
+0x002 Flags              : Uint2B
+0x004 LastWriteTime      : _LARGE_INTEGER
+0x00c Spare              : Uint4B
+0x010 Parent             : Uint4B
+0x014 SubKeyCounts       : [2] Uint4B
+0x01c SubKeyLists        : [2] Uint4B
+0x024 ValueList          : _CHILD_LIST
+0x01c ChildHiveReference  : _CM_KEY_REFERENCE
+0x02c Security           : Uint4B
+0x030 Class              : Uint4B
+0x034 MaxNameLen         : Pos 0, 16 Bits
+0x034 UserFlags          : Pos 16, 4 Bits
+0x034 VirtControlFlags   : Pos 20, 4 Bits
+0x034 Debug              : Pos 24, 8 Bits
+0x038 MaxClassLen        : Uint4B
+0x03c MaxValueNameLen    : Uint4B
```

```
   +0x040 MaxValueDataLen  : Uint4B
   +0x044 WorkVar          : Uint4B
   +0x048 NameLength       : Uint2B
   +0x04a ClassLength      : Uint2B
   +0x04c Name             : [1] Uint2B
```

If this is compared with our current knowledge of the "nk" record structure we can see, with the exception of the cell size element, that this is a good fit in that nothing shown here conflicts. There are some extra puzzles as there is more to work out.

One answer is given by Dolan-Gavitt (aka moyix) in the paper "Forensic Analysis of the Windows registry in memory" (Dolan-Gavitt, 2008) where he says.

> "One crucial difference, however, is the existence of volatile keys and values in in-memory hives. The `_CM_KEY_NODE` structure has two members, `SubKeyCounts` and `SubKeyLists`, that give the number of subkeys and a pointer to the subkey list, respectively. Each member, however, is actually an array of length two: the first entry in the array refers to the *stable* keys, while the second refers to the *volatile* keys."

This provides two insights. The first rather obviously fills in some otherwise unknown values in the "nk" record. The second more importantly shows us that there are volatile parts to Registry Hives that have an external, in file, existence. In other words non-volatile hives can have volatile Keys.

There are two peculiarities about this structure definition. The first is the use of the _CHILD_LIST structure. The structure of this can be seen by the usual method by using a command as follows.

```
          dt nt!_CHILD_LIST
```

This produces this output

```
   +0x000 Count            : Uint4B
   +0x004 List             : Uint4B
```

Why this should be done, rather than have scalar values, is not immediately obvious.

The other oddity is that there are two alternative definitions for part of the structure from offset 0x1c. The second alternative is that it is for a `_CM_KEY_REFERENCE` object. The structure of this was extracted

```
          dt nt!_CM_KEY_REFERENCE
```

This produces this output

```
   +0x000 KeyCell          : Uint4B
   +0x004 KeyHive          : Ptr32 _HHIVE
```

The use of this was not immediately apparent but was discovered by further work which is explained below.

We have the definitive structure of a record, in this case the “nk” record.

4.12 The _CM_KEY_VALUE structure

In a similar way to the exploration of the _CM_KEY_NODE structure the _CM_KEY_VALUE structure was examined. The structure of this was extracted by issuing the WinDbg command

```
          dt nt!_CM_KEY_VALUE
```

This is the resultant output

```
   +0x000 Signature        : Uint2B
   +0x002 NameLength       : Uint2B
   +0x004 DataLength       : Uint4B
   +0x008 Data             : Uint4B
   +0x00c Type             : Uint4B
   +0x010 Flags            : Uint2B
   +0x012 Spare            : Uint2B
   +0x014 Name             : [1] Uint2B
```

Once again, allowing for the omission of the initial size element, we can compare this with our current knowledge of the “vk” record structure and we can see very good agreement, there are no conflicts or unexplained parts.

We now have the definitive structure of a “vk” record.

4.13 The _CM_KEY_INDEX structure

Issuing the Windbg command.

```
          dt nt!_CM_KEY_INDEX
```

Produces this output.

```
   +0x000 Signature        : Uint2B
   +0x002 Count            : Uint2B
   +0x004 List             : [1] Uint4B
```

This seems to be only a generic header for all the subkey list types, that is “lf”, “lh”, “li” and “ri” and so does not add much to our knowledge. It does give us the proper, Microsoft, names for these elements.

4.14 The _CM_BIG_DATA structure

The structure of this is exactly the same as for _CM_KEY_INDEX which makes it something of a mystery. It is not known what this might be used for.

4.15 The _CM_KEY_SECURITY structure

Work on this produced some very valuable results. It is the most complicated record structure held in a Registry File.

This can be approached in the normal way by displaying the structure of the data type using this command.

```
          dt nt!_CM_KEY_SECURITY
```

This gives us

```
   +0x000 Signature        : Uint2B
   +0x002 Reserved         : Uint2B
   +0x004 Flink            : Uint4B
   +0x008 Blink            : Uint4B
   +0x00c ReferenceCount   : Uint4B
   +0x010 DescriptorLength : Uint4B
   +0x014 Descriptor       : _SECURITY_DESCRIPTOR_RELATIVE
```

WinReg.txt (B.D., undated) has a neat, text, diagram which shows the “sk” records being chained together. This matches with the Flink and Blink items above.

On line 581 it states that “The usage counter counts the number of references to this “sk”-record.”. On line 575, part of a definition of the “sk” structure, defines the D-Word at 0x000C as being the usage-counter. This matches the ReferenceCount above.

This source states, line 468, that the sk record “is the ACL of the registry”.

“Windows Internals” p203, Table 4-6 states “Security-descriptor cells include … a reference count that records the number of key nodes that share the security descriptor”. (Russinovich, 2005a).

“Beginning to see the Light” (Clark, 2005) was instrumental in unravelling this structure. In the section headed “Audit and permissions:” are a number of useful snippets and some detailed work matching security settings on keys (set via regedit) with hex values seen in this record. This source states, of the value at offset 0x14 in the cell (including the cell size).

> "Length of entry (not surplus) from offset 18 (which is right after this dword) All offset here after are with respect to offset 18"

Allowing for the initial four byte cell size this places this value at 0x10 which is named above as DescriptorLength. The data starting immediately after is named as Descriptor and has a type of `_SECURITY_DESCRIPTOR_RELATIVE`.

What we have here is an NT Security Descriptor of the type used for files, file shares, in Active Directory and other places. The term Security Descriptor is often abbreviated to SD.

Decoding this is of significant forensic interest as it is has applications outside just the Registry.

The _CM_KEY_SECURITY structure is acting merely as a container for this Security Descriptor which can be described as its payload.

The Security Descriptor was decoded in its entirety. Initially this was done using barely more than the description from "Beginning to See the Light" and a hex dump of the data. Subsequently a very useful description of these structures was found in the C header file winnt.h.

This work is too lengthy to describe here and is explained in detail in Appendix 12.

The result is a complete description of all the bytes that make up a Windows NT Registry Security Descriptor.

4.16 Registry Values Types

The "vk" record contains a Data Type field which defines the type of data.

The following Value Types and their code values were taken from winnt.h. The definitions of these values in the Visual Studio .NET 2003 winnt.h file was the most up to date found in that it includes the value types REG_QWORD and REG_QWORD_LITTLE_ENDIAN.

REG_NONE	0	No value type
REG_SZ	1	Unicode nul terminated string
REG_EXPAND_SZ	2	Unicode nul terminated string (with environment variable references)
REG_BINARY	3	Free form binary

REG_DWORD	4	32-bit number
REG_DWORD_LITTLE_ENDIAN	4	32-bit number (same as REG_DWORD)
REG_DWORD_BIG_ENDIAN	5	32-bit number
REG_LINK	6	Symbolic Link (unicode)
REG_MULTI_SZ	7	Multiple Unicode strings
REG_RESOURCE_LIST	8	Resource list in the resource map
REG_FULL_RESOURCE_DESCRIPTOR	9	Resource list in the hardware description
REG_RESOURCE_REQUIREMENTS_LIST	10	
REG_QWORD	11	64-bit number
REG_QWORD_LITTLE_ENDIAN	11	64-bit number (same as REG_QWORD)

These values fit well with the Parse-Win32Registry values which can be found in Win32Registry.pm at lines 38-53 and are as follows.

REG_NONE	0
REG_SZ	1
REG_EXPAND_SZ	2
REG_BINARY	3
REG_DWORD	4
REG_DWORD_BIG_ENDIAN	5
REG_LINK	6
REG_MULTI_SZ	7
REG_RESOURCE_LIST	8
REG_FULL_RESOURCE_DESCRIPTOR	9
REG_RESOURCE_REQUIREMENTS_LIST	10
REG_QWORD	11

Definitions from Microsoft can be found in “Registry Value Types” (Microsoft, 2008ac).

4.17 Navigating Registry Structures in Memory

At this point we have reached a stage were it would be very helpful to be able to navigate the Registry structures in memory in much the same way as we can when they are in Registry Files.

To do this we have to know where to find the hbins that make up the Registry whether it is in File or in Memory.

When in a file it is easy, they are simply contiguously laid out in the file. This is not necessarily true in memory. Although Windows can (and does) make an initial contiguous allocation it is not possible to make sure that any extra hbins that might be needed will have contiguous memory allocated.

Another complication is that we also have volatile keys attached to otherwise stable Registry Hives. The stable parts of these Hives are kept safe in Registry files so they can be reloaded, the volatile parts are not and are kept in a parallel set of volatile hives. Somehow they must be able to be found by some sensible mechanism.

The way this is done is explained in “Windows Internals” p 203-207 (Russinovich, 2005a) and in “Cell Index Translation” (moyix, 2008d). Much of this section is taken from these references. Russinovich makes no mention of volatile cells, only of volatile hives which have stable (and volatile) cells but no associated file.

This mechanism may seem over complex but does have some reason to it.

It all centres on the Cell Index (or CellIndex). We know that when the Hive is in a file the Cell Index is the offset from the beginning of the first hbin. Since this is always at offset 0x1000 in the file we know that to get the File Offset from the Cell Index we add 0x1000.

In memory we have to make more sophisticated use of the Cell Index. It does still have the same conceptual meaning in that it an offset from the first hbin and if they were in contiguous memory that would be enough.

We know, as a Block is 4K in size, that the last 12 bits, or last three Hexadecimal digits, will determine the offset in the relevant 4K block of the cell we are trying to find.

The other 20 bits of the Cell index are split into fields which allow us to find both the address of the Block and the address of the hbin which might not be the same.

The reason why the block address and the hbin address might be different is because of hbins that are more than 4K in size. If an hbin consists of one 4K block then the address of that block will be the same as the address of the hbin. If the hbin consists of multiple 4K blocks then only the first block will have the same address as the hbin. The other blocks must have addresses in increments of 4K. That is hbins must be in contiguous memory as otherwise cells that crossed a 4K boundary would become split in memory.

So we need a data structure that will allow us to lookup the address of the hbin and the address of the block given the Cell Index. The structure that allows this is the “Cell Map Table”. The internal structure name for this is the _HMAP_TABLE. We can see the structure in the normal way using WinDbg.

```
          dt nt! _HMAP_TABLE
```

what we get is this

```
   +0x000 Table            : [512] _HMAP_ENTRY
```

If we pursue this, we can find out what an _HMAP_ENTRY looks like with this command.

```
          dt nt!_HMAP_ENTRY
```

We get this

```
   +0x000 BlockAddress     : Uint4B
   +0x004 BinAddress       : Uint4B
   +0x008 CmView           : Ptr32 _CM_VIEW_OF_FILE
   +0x00c MemAlloc         : Uint4B
```

Clearly we have our BlockAddress and BinAddress. The other two entries are the Configuration Manager View of File and the size of memory allocated to this object.

We can see that each _HMAP_ENTRY object is 16 bytes long and that there are 512 of them in a Table which neatly comes to 8K or 2 x 4K of memory.

This is all very well but what happens if you have more than 512 blocks of Registry which is not unusual? We only need 9 bits to reference the 512 entries in this table. Using the higher order bits we can select which table by looking it up in a Cell Map Directory. This is of type _HMAP_DIRECTORY and so the structure can be seen with the following command.

```
          dt nt!_HMAP_DIRECTORY
```

Which gives us this

```
   +0x000 Directory        : [1024] Ptr32 _HMAP_TABLE
```

This data structure is 4K in size (1024 x 4 byte entries). It is referenced by the next higher 10 bits from the Cell Index.

The advantage of this two stage data structure is that the detailed parts of the table only need be in memory when wanted and, as long as the Registry is not too fragmented, should reduce memory usage and page thrashing.

This is significantly more flexible than having one table of 2^{19} _HMAP_ENTRY elements. It would not be feasible to have a full size table in memory and if you have part-sized table then how would you shrink or expand it?

There is one final piece of the puzzle left. We have only accounted for 31 of the 32 bits of the Cell Index (12 + 9 + 10). The highest order bit is used to select between the Stable and the Volatile sets of hbins. If it is set to zero then we are looking for a stable cell in a Stable hbin, if it is set to 1 we are looking for a volatile cell in a Volatile hbin.

The selection here is from the two element Storage item which is a part of the _HHIVE structure. These are of type _DUAL and their structure can be found with this command.

```
dt nt!_DUAL
```

This gives the following.

```
+0x000 Length           : Uint4B
+0x004 Map              : Ptr32 _HMAP_DIRECTORY
+0x008 SmallDir         : Ptr32 _HMAP_TABLE
+0x00c Guard            : Uint4B
+0x010 FreeDisplay      : [24] _RTL_BITMAP
+0x0d0 FreeSummary      : Uint4B
+0x0d4 FreeBins         : _LIST_ENTRY
```

We can see in this structure an _HMAP_DIRECTORY element and an _HMAP_TABLE element.

This latter element is of some surprise as we would have expected to find that this was an array of 1024 such items but what we have is only one entry (which will have 512 entries in it). The reason for this is apparently an optimisation for small hives which have no more than 512 blocks. This will be explained further later on.

We could reasonably expect that the Length element would be the mirror of the length element in the BaseBlock (“regf” header). Without this entry here there would not be one for the volatile part of a hive.

The FreeDisplay element is used for managing the free cells in memory “Windows Kernel Internals” p10 (Probert, undated).

It is not known what the Guard, FreeSummary or FreeBins elements are used for.

4.17.1 Finding HBINS – Small Hive

Appendix 13 shows a worked example of finding an hbin in memory for the case where the total size of the hbins is no more than 512 blocks.

This is quite an easy process. Understanding it is fundamental to understanding how this mechanism works.

4.17.2 Finding HBINS – Large Hive

Now that we have established a method we can apply it to another Hive which is larger, and which has a volatile part.

In this case the Stable part is more than 512 blocks in size.

This was done and is explained in some detail in Appendix 14.

4.17.3 Finding HBINS – Large HBINS

The work in this section was completed before the worked example using the SYSTEM Hive shown in Appendix 14 was completed in detail.

The purpose of this work was to show the impact on the internal structures of having an hbin of more than one 4K block. This is something which the above example fortuitously came across.

This experiment is reproduced anyway as it neatly illustrates some points.

Full details can be found in Appendix 15.

4.17.4 Finding HBINS – Another Way

In his web article “Internal structures of the Windows Registry” (Anand, 2008) explains the following process which is equivalent to what we have been doing.

If we start from “`!reg hivelist`” command as before we can get the following data for the SYSTEM Hive.

Data Item	**Value**
HiveAddr	e1036008
Stable Length	5d8000
Stable Map	e1038000
Volatile Length	24000
Volatile Map	e1036144
BaseBlock	e1037000
Filename	SYSTEM

The “cellindex” command of the reg extension will allow us to find the block location in memory from the address of the Hive and the Cell index. For example, if we want to find where a cell index of 0x20 is in the SYSTEM hive above we can do so with this command.

```
!reg cellindex e1036008 20
```

Which produces this output.

```
Map = e1038000 Type = 0 Table = 0 Block = 0 Offset = 20
MapTable     = e1039000
BlockAddress = c7901000

pcell:  c7901024
```

As can be seen this has picked out the Cell Map Directory (0xe1038000) and the Cell Map Table (0xe1039000) and used that to look up the Block Address (0xc7901000). The Cell index has then been added and as a final touch another 4 has been added to step over the initial size element of the cell.

This method works (unsurprisingly) and is quite easy. However it is obscure and lacks the transparency needed for forensic analysis. It also obviously will only work on a live system that has the debugging tools installed.

It does provide a valuable means of verifying our manual method.

4.17.5 Bin Address Flag Values

As has been commented on earlier in the Appendices, the BinAddress values in the Cell Map Table entries have some of the low order bits set. A small number of values have been seen and it is now possible to speculate what they might be.

This analysis is purely observational and coincidental, and may be completely wrong. It is however plausible.

Value	Analysis
0x01	This entry is for an hbin
0x02	not seen
0x04	seen but no explanation
0x08	This entry is for a Volatile block

4.18 Verifying ACL and ACE values

Earlier we were able to derive the structure of the Security Key (“sk”) record and show the components down to ACL and ACE level.

We ought to be able to verify that certain bit values have certain meanings with respect to the settings in the regedit program Permissions GUI.

This can be done by manipulating the Permissions using the regedit GUI and then seeing what values appear in the data, the actual registry bytes.

One technique would be to start regedit, load a hive, make a change, unload the hive (so that file is not locked) and look at the data in WinHex. This would be quite laborious. With the knowledge gained so far about finding the Registry objects in memory a better alternative is to find the object in memory and then observe these values before and after changes..

A risk is that if the modified Security Key (“sk” record) becomes larger then it will be moved to a new location. We could find ourselves looking at now unused bytes.

4.18.1 Preparation

A copy of the test Registry File TEST4 was prepared by adding a new trustee. This was needed as all the standard ACEs are inherited and so cannot be manipulated.

Full details of the procedure is given in Appendix 16.

4.18.2 Permissions Settings

It was now possible to step through all the possible values of the Permissions, as set using the regedit GUI, and to see what values this produced in the relevant bytes of the ACE.

This work is explained in detail in Appendix 17.

This confirmed the following table.

Permission Setting	**Value of Permissions DWORD**
Query Value	0x00000001
Set Value	0x00000002
Create Subkey	0x00000004
Enumerate Subkeys	0x00000008
Notify	0x00000010

`Create Link`	`0x00000020`
`Delete`	`0x00010000`
`Write DAC`	`0x00040000`
`Write Owner`	`0x00080000`
`Read Control`	`0x00020000`
`Full Control`	`0x000F003F`

4.18.3 ACE Inheritance Settings

We can apply the same method to unravelling what Inheritance permissions set on a Trustee in the GUI cause what data changes.

The GUI settings fall into two categories.

1. A drop down list called “Apply onto:” which has the following three options
   - This key and subkeys
   - This key only
   - Subkeys only

2. A tick box labelled “Apply these permissions to objects and/or containers within this container only”

The details of how this was done can be found in Appendix 18.

From this work the following table was derived.

“Apply onto:”	**“Apply these…”**	**AceFlags Value**	**AceFlag Names**
This key and subkeys	Not Set	0x02	`CONTAINER_INHERIT_ACE`
This key only	Not Set	0x00	
Subkeys only	Not Set	0x0A	`CONTAINER_INHERIT_ACE` `INHERIT_ONLY_ACE`
This key and subkeys	Set	0x06	`CONTAINER_INHERIT_ACE` `NO_PROPAGATE_INHERIT_ACE`
This key only	<cannot set>	No Result	No Result
Subkeys only	Set	0x0E	`CONTAINER_INHERIT_ACE` `NO_PROPAGATE_INHERIT_ACE` `INHERIT_ONLY_ACE`

4.18.4 Security Descriptor Inheritance Settings

The “Advanced Security Settings..” dialogue box has two tick boxes on the Permissions tab and the Auditing tab. These read.

> “Inherit from parent the permission entries that apply to child objects. Include these with entries explicitly defined here.”
>
> “Replace permission entries on all child objects with entries shown here that apply to child objects”

The first of these controls whether or not parent properties are allowed to propagate into this object. By default this is selected.

The second of these is merely a request to re-apply settings to child objects in case they have not propagated properly. It is a setting that calls for an action rather than a setting that changes the configuration.

The possible values were examined as described in detail in Appendix 19.

The conclusion reached is that setting “Inherit from parent…” in either the Permissions or the Auditing tab clears a bit in the Control variable of the Security Descriptor that inhibits inheritance.

The bit values are as follows, from winnt.h.

Permissions	SE_DACL_PROTECTED	0x1000
Auditing	SE_SACL_PROTECTED	0x2000

Each of these bits, when set, prevents otherwise inheritable ACEs from being inherited.

More information about the Control item can be found in this Microsoft web page “SECURITY_DESCRIPTOR_CONTROL Data Type”, (Microsoft, 2008n).

4.19 The Registry Namespace

It is familiar to refer to keys by their pathname as shown in the regedit program. For example

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet.

This is not how Windows refers to these keys internally. Windows converts all such key references to names that start with the term “REGISTRY” and so the above key would be known as follows

\Registry\Machine\System\CurrentControlSet

(Upper or lower case is not relevant).

When Windows is asked to open an object with a name that starts "\Registry\" it knows to delegate this task to the Configuration Manager.

This is explained in "Windows Internals" p 207 (Russinovich, 2005a). which states.

> "Regedit shows key names in the form HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet but the Windows subsystem translates such names into their object namespace form (for example, \Registry\Machine\System\CurrentControlSet)".

4.20 Key Node flag values

The Key Node record ("nk") contains an element calls Flags.

An experiment was conducted to determine the proper internal name for each of the bits in this value.

Details of this experiment are described in Appendix 20. The results of this were as follows.

Bit	Hex	Decode	
1	0x0001	Unused	
2	0x0002	HiveExit	
3	0x0004	HiveEntry	Root Key
4	0x0008	NoDelete	Deletion not allowed
5	0x0010	SymbolicLink	
6	0x0020	CompressedName	Name in ASCII not Unicode
7	0x0040	PredefinedHandle	
8	0x0080	VirtMirrored	
9	0x0100	VirtTarget	
10	0x0200	VirtualStore	
11	0x0400	Unused	
12	0x0800	Unused	
13	0x1000	Unused	
14	0x2000	Unused	
15	0x4000	Unused	
16	0x8000	Unused	

With this information we can decode the two most commonly seen Flag values in nk records. These are

0x2C	CompressedName, NoDelete, HiveEntry
0x20	CompressedName

This explains why the Root Key normally has the flags value set to 0x2C and most other keys have 0x20.

It is apparent that “CompressedName” means that the key name is in ASCII rather than in Unicode.

The NoDelete bit prevents regedit from deleting the key as a simple experiment shows (set this bit on a key in memory and then try to delete it via regedit).

HiveEntry appears to be set on the Root Key of each Hive, including that of the Registry or Root Hive. This may be to show code that is navigating up the chain of Keys from Child to Parent that it has reached the top of the tree. Note that this is not shown by having any ‘end of chain’ marker in the Parent field of a Hives Root Key. More on the Parent field of Root Keys later.

The “All Files” database (run 15) was checked to see what values of this Flags field could be found using this query.

```
SELECT DISTINCT Flags, Count(*) AS [Count]
FROM nk GROUP BY Flags

UNION

SELECT DISTINCT Flags, Count(*) AS [Count]
FROM lk GROUP BY Flags
```

The results, with bit names, were as follows.

Flags	**Count**	**Bits**	**Bit Names**
0x00	6		
0x20	1,130,093	0x20	CompressedName
0x2A	12	0x20, 0x08, 0x02	CompressedName, NoDelete, HiveExit
0x2C	165	0x20, 0x08, 0x04	CompressedName, NoDelete, HiveEntry
0x30	8	0x20, 0x10	CompressedName, SymbolicLink
0x60	1	0x40, 0x20	PredefinedHandle, CompressedName
0x1020	53,709	0x1000, 0x20	<Unused>, CompressedName

This was from a total of 1,183,994 “nk” and “lk” records.

The following conclusions were drawn.

- Values of 0x20 and 0x00 are for ordinary keys the difference between them being that 0x20 indicates that the Key Name is held in Compressed form, that is in ASCII. Another way of looking at this is that we have just 6 Key names in this set of data that are in UniCode.
- The 165 x 0x2C values are the Root Cells of the 165 full Registry files in the data set.
- The 8 x 0x30 values are all in the Volatile parts of Hives (one of which is Volatile itself). These are explained below in the section on Symbolic Links.
- The 12 x 0x2A values are the 12 “lk” or Key Link cells that link from the Registry hive to the other Hives.

  HiveExit and HiveEntry can be seen here as the two ends of the link relationship. The “lk node having the HiveExit bit set and which points to the Root Cell of the Hive which has its HiveEntry bit set.
- The one “nk” record that has the PredefinedHandle bit set is quite unusual. It is to do with holding Performance Data and it refers to data held externally in some files.

  Please see Appendix 36 for more information about this key type.
- The bit value 0x1000 which is set in 53,709 nk records are all from Vista files so it seems that this is a new value introduced with Vista. Further work needed.

4.21 Extracting the Registry Hive from Memory

If we know where to find the hbins in memory we should be able to extract them and make a Registry File.

There is not a lot of value in doing this for the Hives that have a Registry file as they are flushed out to disk regularly (or at least the Stable parts of them are).

This is of more interest for the Volatile Hives that do not normally exist outside memory or for the Volatile hbins of Hives that do have a Stable part.

The sizes of all the hives can be seen with the “`!reg hivelist`” command. It has already been shown that this list is in reverse order of loading, that is the most recently loaded is at the top and the hive that was loaded first is at the bottom.

That bottom hive typically has these values, taken from a run of "`!reg hivelist`".

Name	Value
HiveAddr	0xe102d008
StableLength	0x1000
StableMap	0xe102d068
VolatileLength	0x 1000
VolatileMap	0xe102d144
BaseBlock	0xe102e000

It can be seen that this hive only has 2 hbins, one stable and one volatile. (It is a puzzle as to why a volatile hive has a volatile hbin since by definition it is all volatile anyway).

The BaseBlock and the Stable hbin were extracted and converted to a binary file. The Volatile hbin was also extracted.

The details of how this was done and an analysis of the results can be seen in Appendix 21.

It is plain from this analysis that the purpose of this Registry hive which we have unravelled is to act as the entry point for all the Registry Namespace. That is to allow namespace names like "\Registry\Machine\SAM" to be mapped to the SAM Hive.

A vital part of this mechanism is the discovery of a new record type called "lk" which perhaps stands for "Key Link". It is this node that links from this root Registry hive to all the other hives in the Registry.

The structure of the Registry Hive is diagrammed in Appendix 22. The original template sheets used are shown in Supplement 17 as a worked example of how a manual dissection can be done.

As explained in Appendix 21 we can now map everything that is seen in regedit to real actual data.

As a further refinement a method was developed that allows all the hbins of a registry hive to be easily extracted from memory. This more advanced method is explained in detail in Appendix 23.

Using this advanced method both the Volatile Hives (Registry and Machine) and all the Volatile hbins of all hives that have Volatile hbins were extracted and added to the pool of Registry files that were routinely analysed overnight.

4.22 Symbolic Links

Eight “nk” records were found which had the SymbolicLink bit set in their Flags value.

All of these were in the Volatile parts of Hives, in one case (Registry) it was in the Volatile part of a Volatile Hive which seems a little unnecessary.

In all cases the “nk” record had no SubKeys and just one Value.

In all cases the Value had a name of “SymbolicLinkValue”. The value of the Value was the full Registry namespace path of the Key that the Link pointed to.

The eight Symbolic Links were found in the following Hives.

Name	Count
Registry	1
Security	1
System	3
ntuser.dat (Current User)	1
ntuser.dat (Local Service User)	1
ntuser.dat (Network Service User)	1

The links were as follows (expressed in Registry namespace).

Registry:	
Link Name:	`\Registry\User\S-1-5-18`
Links To:	`\Registry\User\.DEFAULT`

Security	
Link:	`\Registry\Machine\Security\SAM`
Link To:	`\Registry\Machine\SAM\SAM`

System:	
Link:	`\Registry\Machine\System\ControlSet001\Control\Print\Printers`
Link To:	`\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Print\Printers`
Link:	`\Registry\Machine\System\ControlSet001\Hardware Profiles\Current`
Link To:	`\Registry\Machine\System\CurrentControlSet\`

	`Hardware Profiles\0001`
Link:	`\Registry\Machine\System\CurrentControlSet`
Link To:	`\Registry\Machine\System\ControlSet001`

ntuser.dat (Network Service)	
Link:	`\Registry\User\S-1-5-20\Software\Classes`
Link To:	`\Registry\User\S-1-5-20_Classes`

ntuser.dat (Local Service)	
Link:	`\Registry\User\S-1-5-19\Software\Classes`
Link To:	`\Registry\User\S-1-5-19_Classes`

ntuser.dat (Current User)	
Link:	`\Registry\User\S-1-5-21-4200165691-2687452118-2273033371-1108\Software\Classes`
Link To:	`\Registry\User\S-1-5-21-4200165691-2687452118-2273033371-1108_Classes`

4.23 Key Node Parent Values

If a Key does not have any SubKeys then the data item in the “nk” record that would normally point to the SubKey List will be set to -1 (0xFFFFFFFF).

As has been previously shown, each Key Node has a value called Parent which contains the Cell Index of the Key Node that is the Parent to that Key.

By traversing the Parent links the full path name of a Key can be determined.

However there are two puzzles. First of all this mechanism surely limits the reverse path to the particular hive, how do we know the name of the hive?

The second is that even the Root Key of a Hive has a sensible value in the Parent field when surely this should be some sort of terminating value such as 0 or -1. The values found make no sense within that hive.

It was speculated that the Parent value in the Root Cell of each hive might in fact be the Cell Index of the “lk” record in the Registry or Root Hive.

This theory was checked by examining all the hives and checking the Parent value in each of their Root Cells against the Cell Indices of the relevant “lk” records in the Registry hive. There was a complete 100% match.

On checking the Root Hive the Parent Value of its Root Key was found to be set to -1 (0xFFFFFFFF) which plainly marks the termination of the Parent paths for all Keys in the entire Registry.

All hive Root Cells have in their Flags value the HiveEntry bit set. When navigating up through the Keys it seems that this may be used to indicate that the path now passes to the Registry hive.

4.24 Extracting ntoskrnl Symbols

The use of internal symbols from the ntoskrnl module (the NT Kernel) proved to be a very good way of understanding the Registry structures.

It was thought a useful approach to attempt to get a comprehensive list of all these to see what else might be of interest.

The Microsoft debugger, WinDbg used to allow a wildcard syntax for symbols which was as follows

```
dt nt!*
```

The above command would list all symbols. This form has been discontinued and no longer works on the current version of the debugger.

The symbols for the Kernel are contained in a file called ntoskrnl.pdb which can by found in the Symbols folder after the symbols have been downloaded from Microsoft or in the cache if the Symbols server has been used as recommended. See Appendix 4 for details of how to install the Windows Debugger.

The debugger command "`lml`" will show which symbols have been loaded and gives the path to where the PDB file can be found.

PDB stands for Program DataBase (Microsoft, 2008ad).

An attempt was made to extract the synbols from the Symbol file by using an example program supplied by Microsoft with Visual Studio, which is Microsoft's development environment (Microsoft, 2008g). This program is called "diadump". It uses the Microsoft "Debug Interface Access SDK" (Microsoft, 2008h) to dump the contents of a .pdb file.

This proved to be fruitless, the program ran for a very long time and produced an obviously erroneously large output.

The second attempt took a more empirical approach and simply used a Windows version of the well known "strings" program to extract plausible strings from the ntoskrnl.pdb file. These were then filtered with the "Filter Symbols" script. The list produced from that was then used to create a script file for kd.exe which is the command line version of the Kernel Debugger. This script file used the Kernel Debugger command “dt” which displays the layout of a structure. In this way a complete list of all Kernel structures with a name of three or more characters was found.

Appendix 24 explains the process in detail.

This was successful in producing a plausible list of symbols and these were of some use. Notably the _ACL and _SECURITY_DESCRIPTOR_RELATIVE.

A third method was tried. Paul Schreider (Schrieder, 2008) has produced a PDB Parser program called Windows PDB Exploder, the executable of which is called win_pdbx.exe.

This was run against the ntoskrnl.pdb file. The output is somewhat crude which is in keeping with the kind of tool that it is. The output was filtered to produce an alternative list of Kernel structures.

Full details of this attempt is given in Appendix 25.

The result had 413 entries, a lot less than had been obtained using the Strings approach. It did not reveal any new structures that had not been seen before.

4.25 How Dirty pages are flushed out

According to “Windows Kernel Internals” p12 (Probert, undated)…

> “Every time a cell is dirtied the whole page is marked dirty (in the Dirty Vector)”

“Windows Internals” p 209 (Russinovich, 2005a) suggests that this is not done by 4K block as stated above but by 512 byte sector.

> “…a bit array in which each bit represents a 512-byte portion, or *sector*, of the hive. …an *on* bit in the array means the system has modified the corresponding sector in the hive in memory and must write the sector back to the hive file.”

Probert states at p14 that the Kernel has a bitmap of type RTL_BITMAP which has a bit set for each 4K block that has been altered (“one bit per page”) and so needs to be flushed out to disk. It would appear that this is the DirtyVector element of the _HHIVE data structure.

An experiment was conducted to see if the data would be written out in 512 byte sectors or in 4K blocks. The details of this experiment are given in Appendix 27.

It was clearly shown that that the whole 4K block is written out.

A possible explanation for the above apparent discrepancy between these two authoritative authors is that it used to be done in 512 byte sectors but is now done in 4K blocks. Possible reasons are that the recovery process using .LOG files works better with 4K blocks or it may be that it was realised that as the disk system has to write a minimum of 4K anyway then it could be less efficient to write amounts smaller than that as the disk system will need to read 4K in, patch in the changed 512 byte sector and then write it out again. In other words it may be that what Russinovich wrote was correct at the time it was written.

However there is another mystery.

As the experiment in Appendix 28 shows, the bitmap which shows which part of the Hive is dirty and needs to be flushed is sized at eight times what would be needed for 4K blocks. That is it is sized as would be needed for 512 byte sectors.

As described by Probert p16 (Probert, undated) there is a normally 5 second delay before the “Lazy Flush” is initiated which then checks all the hives and flushes out any dirty pages.

As a result these bitmaps will nearly always be all zeroes. The only time that any part of any of them will be non-zero is during the short period between a change having been made and the data being written out.

Using this information all the bitmaps were examined for all non-volatile hives and this clearly showed that the bitmap size was the number of bits and not the number of bytes.

Details of the method used are in Appendix 29.

The last question to answer is whether the bitmaps are marked up on a bitwise basis. It is obviously quite hard to spot a change that might only exist for no more than 5 seconds.

Windows 2003 Server does have a registry value which can be changed to alter this period. This value is

```
    HKLM\System\CurrentControlSet\Control\
    Session Manager\Configuration Manager\
    RegistryLazyFlushInterval
```

”Windows Internals” p209 (Russinovich, 2005a). (In the book the element “Control” is missing from the path).

A Microsoft article hints that this may also work on XP SP2 (Microsoft, 2006a).

This was attempted on an XP SP2 computer, setting the lazy flush interval to 60 seconds. A small hive was loaded (the one used in the earlier experiment detailed in Appendix 27). The location of this hives dirty bitmap was determined in the debugger. The only value in this hive was then altered using regedit and the dirty bitmap examined.

It was found that the relevant byte in the dirty bitmap was set to 0xFF, a setting which lasted about 5 seconds showing that the registry change did not have any impact.

The computer was rebooted, the experiment repeated and it was found that the lazy flush interval had been changed to about 90 seconds. This time is probably a reflection of the approximate nature of the setting. So the Registry key change does work but unsurprisingly needed a reboot.

The fact that all 8 bits of the bitmap for that block were set explains why all eight 512 byte sectors were written out. It is possible that the ability to only flush out specific 512 byte sectors exists in the kernel code but that that capability is masked by all eight bits being set. This capability may be used for some hives or in some circumstances.

4.26 LOG files

Every Registry File has a .LOG file associated with it named as the Registry file name with ".LOG" appended. For example, the Registry file "SAM" has an associated log file called "SAM.LOG". If a Registry file is loaded, perhaps by regedit, and it does not have a .LOG file then one will be created.

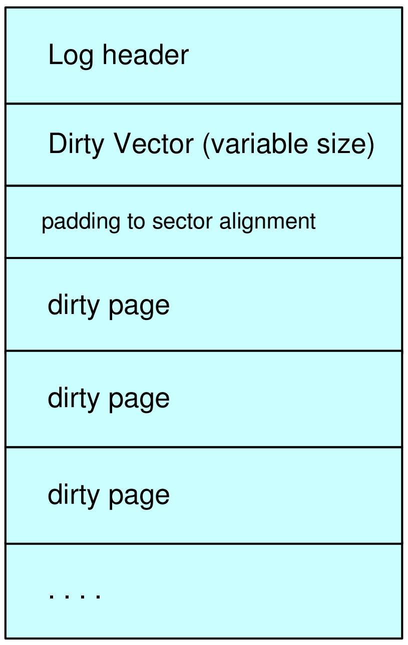
The purpose of these .LOG files is to provide a safe and recoverable mechanism for updating the Registry file.

Any updates are first written to the .LOG file and only when safely flushed to that file are the changes made to the main file. In this context the main file is called the "PRIMARY" file and the .LOG file is called the "LOG".

How this works is explained in more detail in Appendix 30.

The structure of the .LOG files is explained in "Windows Kernel Internals" p15 (Probert, undated). The following diagram is taken from there.

.LOG File



The Log Header is the same basic format as the “regf” header and has the same signature.

The Dirty Vector is a copy of the internal data structure and shows where the following blocks come from.

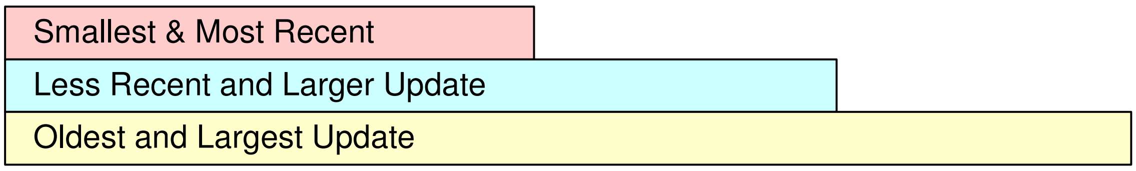
The dirty pages are 4K Blocks, that is only those parts of the hbins that need flushing out.

A simple experiment was carried out on .LOG files as follows. The experiment was carried out on a Windows XP SP2 computer with a large number of windows open.

1. The registry key to lengthen the LAZY_FLUSH time had previously been set so that flushes happened 60-90 seconds after the first change. This allows changes to happen slowly enough to be easily seen.
2. Windows Explorer was opened to the %windir%\system32\config folder.
3. The system was left for a couple of minutes to make sure that any outstanding Registry flushes were carried out.
4. The size of the .LOG files was noted. They were all 1K.
5. All but the Windows Explorer window was closed.
6. It was observed that the sizes of some of the .LOG file swelled by a large amount at the expected interval after the changes started.
7. It was observed that at the next ‘tick’ the .LOG files went back to their default or quiescent size of 1K.

It was deduced from this that .LOG files only contain update data for the LAZY_FLUSH interval which is five seconds by default.

However such information will have a high probability of showing in unallocated space and would seem to be one of the mechanisms by which Registry data can appear in unallocated space. It is to be expected that such data would show in layers with older, bigger, sets of update data showing but masked by more recent updates, something like this diagram.



The structure of the first few bytes of a .LOG file (a SAM.LOG file) was decomposed using the BaseBlock Record template (Appendix 5) and compared with the data in the corresponding SAM Registry file.

The data was found to be very similar between the two files. The following were noted.

- The Sequence numbers in the PRIMARY file were one more than in the LOG file.
- The Type data item was set to 0x01 (HFILE_TYPE_LOG) in the LOG file and 0x00 (HFILE_TYPE_PRIMARY) in the PRIMARY file.
- The Timestamps were identical down to the fraction of a second.

4.27 The “regf” Checksum

The regf header contains a field called “Checksum”.

Work was done to investigate this value and the following was discovered.

The Checksum is a DWORD by DWORD XOR of all the dwords in the first 0x200 bytes except the last dword, which contains the Checksum. No other bytes affect it

The utility xorcheck (Clark, 2005) does not appear to work.

A program, regfXOR.exe, was written to investigate how the Checksum was calculated. This can be used to calculate the Checksum for manually adding to a Registry file.

Full details of this work can be found in Appendix 31. The source code for the regfXOR.exe program can be found in Supplement 19.

4.28 Large hbin sizes

It used to be thought that hbins were all 4K in size. It soon became plain that this is not so and a query was run to see what hbin sizes might be found.

The following query was run against the “All Files” database (run 15).

```
SELECT Size, Count(*) AS [Count]
FROM hbin
GROUP BY Size
ORDER BY SIZE;
```

The following table was constructed from the output.

Size	Num 4K Blocks	Count
4,096	1	81,196
8,192	2	1,602
12,288	3	558
16,384	4	531
36,864	9	55
69,632	17	24
135,168	33	25
266,240	65	6
528,384	129	6
1,052,672	257	3

This is a total of 84,006 hbins of which 2,810 are more than 4K in size.

The three largest (257 x 4K each) are all in `ntuser.dat` files. No further investigation was done of what cells these large hbins contain.

4.29 Hash Values in lh SubKey Lists

An “lh” SubKey list differs from an “lf” SubKey list in the form of the Hash. In an “lf” list this is the first four characters of the SubKey name (right padded with nulls). In an “lh” list the hash is a numeric value. It is plausible that in each case the hash value is used as a consistency check.

Values of the hash and the SubKey name it refers to were extracted from a RegHoover database that was known to contain “lh” records. From that data the algorithm was deduced.

A program was written, HashCalc.exe which when given a file of SubKey Names and Hash values will report on any that do not fit the algorithm found. This was used to test the theory against practice.

The algorithm is:

1. First set hash value to zero
2. Then, working from left to right through the letters of the SubKey name, for each one, multiply Hash by 37 and then add the ASCII value of that letter

There is one complication

Any lower case letters will be changed to upper case. This also happens with characters whose ASCII code is above 127 but which would be a lower case letter if their most significant bit is ignored.

It is not clear how Unicode SubKey names are treated. This was not investigated.

Full details of how this work was done can be found in Appendix 31.

The source code for the HashCheck program can be found in Supplement 14.

4.30 About ri and li SubKey Lists

The "ri" and "li" SubKey Lists are two of the four kinds of SubKey lists.

Much work was done on the data in the "All Files" database to characterise the use of these cells. This work is detailed in Appendix 37.

From this work the following conclusions were drawn about V1.3 Registry files.

- "li" records are only used in V1.3 Registry Files
- "ri" records always point to "li" records.
- "li" records can be pointed to directly by an "nk" record (one in the data examined).
- "li" records are used in preference to "lh" records above a certain number of subkeys which is between 420 and 904.
- "ri" records are used to further sub-divide "li" records above a certain number of subkeys which is between 905 and 1,027.

From this work the following conclusions were drawn about V1.5 Registry files.

- "ri" records always point to "lh" records.
- "ri" records are used to sub-divide "lh" records above a certain number of subkeys which is between 1,010 and 1,256.

- About 98% of “lh” records that are pointed to by “ri” records have a number of entries in the range 506-1,010, the rest are less than 506.
- Only about 0.13% of “lh” records had a number of entries greater than or equal to 506.
- Only about 2% of the “lh” records with 506 or more entries were pointed to directly by “nk” records.

It seems plain from this work that in version 1.3 Registry files, first “li” and then a combination of “ri” and “li” records are used to control the size of SubKey lists in place of the normal “lf” records.

In version 1.5 files “ri” records are used to split up overly large “lh” records.

No instances were found of an “ri” record pointing to another “ri” record and it is hard to see why this would be useful.

It was also discovered that for large SubKey lists a small number of cell sizes seem to be used with large amounts of free space in them. This may be to prevent fragmentation.

4.31 SubKey Lists and Registry File Versions

In version 1.3 Registry files SubKey lists are “lf”, “li” and “ri”.

In version 1.5 Registry files SubKey lists are “lh” and “ri”.

The only Registry file types that are version 1.5 are “`default`”, “`software`”, “`system`” and “`userdiff`” from Windows XP onwards (that is also Windows Server 2003 and Vista).

The above was determined by examination of the “All Files” database (Run 16).

4.32 Flag Values in vk Records

Values that do not have a Name are the Default value for that Key and are shown as such in regedit.

All the information that has been researched states that the “vk” record has a “Name Present” flag (bit zero of the Word at offset 0x14). That is, when the Value has a Name this flag is set, when it does not have a Name then this flag is not set.

This can be observed by looking at the “vk” records. The default value does not have a Name and the Name Present flag is clear, other Values that have a Name also have the Name Present flag set so it must be the case that this bit is the Name Present flag.

It is known that the CompressedName flag in the “nk” record shows, when set, that the Key Name is in ASCII rather than Unicode. No value that could control this behaviour in a “vk” record was known about.

A theory was developed that this flag in the “nk” record might also control Value Names, that is when a Key Name is in Unicode then so must be all the Value Names or conversely if the Key Name is in ASCII then so to must be the Value Names.

A Key, which had some values, had its’ Name changed to contain a Unicode character and the Value Names were examined. They were still in ASCII. This raised the possibility that Value Names could not be in Unicode.

A direct attempt was then made to rename a key by adding a Unicode character and it succeeded. On examination it was found that we had a Value with a name but with the Name Present flag cleared.

Finally it was realised that the Name Present flag is no such thing. It is the CompressedName flag. The reason why it appears when a name is present is because the name is in ASCII which will nearly always be the case if using an English version of Windows where ASCII is quite good enough.

This was tested by looking at a “vk” record in WinDbg where, when the name was ASCII, it was followed by the word “compressed” in brackets.

All the bits of the Flags value were checked and only “compressed” could be detected as being a valid bit setting.

Details of this work can be found in Appendix 32.

4.33 Data Type as Data

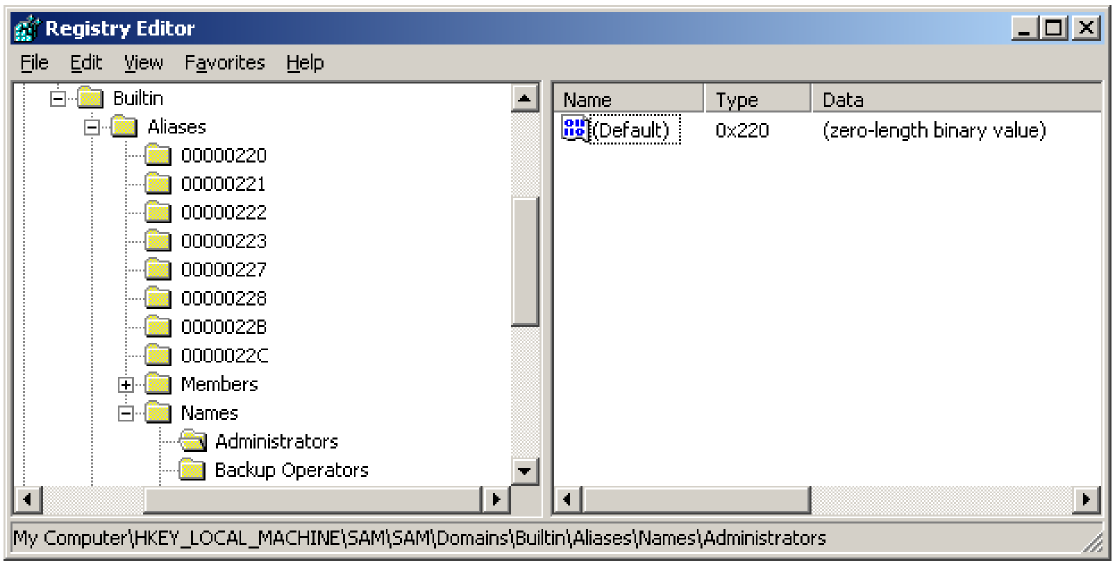
Occasionally data is held in a “vk” record as the Data Type. That is the data is blank but the value held in the Data Type field is out of range and is in fact the data.

An example of this can be found at

```
\Registry\Machine\SAM\SAM\
Domains\Builtin\Aliases\Names\Administrators
```

This has a Default Value with no data but a data type of 0x220 (544) which is the RID of the Administrators Group.

This shows in regedit as follows.



```
Builtin
  Aliases
    00000220
    00000221
    00000222
    00000223
    00000227
    00000228
    0000022B
    0000022C
    Members
    Names
      Administrators
      Backup Operators
```

Name	Type	Data
(Default)	0x220	(zero-length binary value)

My Computer\HKEY_LOCAL_MACHINE\SAM\SAM\Domains\Builtin\Aliases\Names\Administrators

Thomassen (Thomassen, 2008) states that this feature is common in Values that are under `\Registry\Machine\SAM\SAM\Domains\`.

4.34 Max Lengths in nk Records

The "nk" records contain the following four variables which all have names starting "Max".

MaxNameLen
MaxClassLen
MaxValueNameLen
MaxValueDataLen

The "lk" records contain the same elements but these do not appear to be used and were all set to zero in the twelve "lk" records that were extracted into the database (run 16).

From the source code for the reglookup utility (Morgan, 2008b), in particular the header file regfi.h, comes the following fragment. This is part of the definition of a struct called "`REGF_NK_REC`" which must be for the "nk" record.

```
/* max lengths */
uint32 max_bytes_subkeyname;       /* max subkey name * 2 */
uint32 max_bytes_subkeyclassname;  /* max subkey classname length (as if) */
uint32 max_bytes_valuename;        /* max valuename * 2 */
uint32 max_bytes_value;            /* max value data size */
```

This suggests that these values are to hold the maximums found in either SubKeys (for `MaxNameLen` and `MaxClassName`) or Values (for `MaxValueNameLen` and `MaxValueDataNameLen`).

It is possible that this is an optimisation so that when processing a Key the code knows what size of buffers to allocate for the data it will encounter when processing the SubKeys or Values that belong to that key.

Of note is that the name elements (MaxNameLen and MaxValueNameLen) are apparently at twice the actual name lengths. This may be to allow for the Names to be expanded to Unicode if they are in ASCII.

To test the theory as outlined above a script was written to traverse all the keys in a hive in the database and for each one check the max values by scanning all the Values and the SubKeys of each key. The script called CheckMaxLengths was written to do this and, due to the volume of data to process, to report exceptions only.

This listing for this script is shown in Supplement 13.

It was found that the theory was broadly correct with the detail that the Max Name lengths were twice the largest ASCII (Compressed) Name or one times the largest UniCode (Uncompressed) Name whichever was the largest.

It was found that in a small number of cases the Max values were bigger than the underlying data would suggest but in no cases smaller. It was speculated that this was a benign side effect of the internal code used to calculate these values in that perhaps these values ratchet up more easily than they contract. The most important issue is that they are big enough.

4.35 Big SIDs

A small curiosity has emerged from the “All Files” database.

In 311 cases a SID was found that was smaller by four bytes than the space allocated to it in the Security Descriptor (that was contained in an “sk” record). In all cases the SID was either 12 or 16 bytes and has either 16 or 20 bytes allocated to it.

This was found in two files, both of them a “system” Registry file, one each from NT4 and Windows 2000 (W2K). The collection of files only had one set each from NT4 and W2K.

There were 7 occurrences out of 608 ACE records in the NT4 system file and 304 out of 1,145 ACE records in the W2K system file.

Speculation is that it is a fault that was corrected in Windows versions after 2000. This is a benign fault in that all it does is to waste a little space.

4.36 Too Small Data Nodes

The “All Files” database (run 16) showed that in 25 cases the Data Node was too small to hold the mount of data that the “vk” node specified was present.

This happened in 13 files, all but one of which was a software Registry file, the other was a system Registry file (from Vista). In all cases the files were from XP or Vista, that is none were from NT4, Windows 2000 or Windows Server 2003.

It is not certain if this is a fault, an error or weakness in RegHoover or a lack of understanding. More work is needed.

4.37 TimeStamps

There are four Registry records that contain TimeStamps, “regf” or BaseBlock, hbin, “nk” and “lk”.

Only the first hbin in a Registry file contains a TimeStamp.

Of the 174 files in the “All Files” database (run 16) some 15 did not have an hbin with a Timestamp. Ten of these were volatile hives that had been extracted from memory. Oddly 5 were ordinary Registry Files and this was confirmed by examining the original files using WinHex. For the record the files were as follows (the three ntuser.dat files were all for the “Default User”).

Source	**File**	**Operating System**
Computer 13	userdiff	NT4
Computer 14	system	Windows 2000
User 1	ntuser.dat	XP SP2
User 28	ntuser.dat	NT4
User 38	ntuser.dat	Windows 2000

The LastWriteTime from the “nk” records for each of the 159 files was extracted and compared to the TimeStamps extracted. In every case it was found that these did not match. In all but five case the “nk” LastWriteTime was before the TimeStamp date/times, in some cases by over 12 years (NT4).

In the five files that had the more recent “nk” records (that were more recent than the hbin TimeStamps) the discrepancy was between 17 and 56 minutes. This raised the possibility that this was a Daylight Saving Time (DST) issue which was plausible as 3 of the dates were in August and one in September. However one date was in November which is outside the DST period (National Maritime Museum, 2008).

An experiment was devised and conducted to determine if DST would affect the TimeStamps in this way. Details are in Appendix 34.

A summary of the result of this experiment is that the nk LastWriteTime was in all cases 5 seconds or slightly more before the TimeStamp that was present in the hbin and the BaseBlock. This is exactly as expected with a 5 second Lazy Flush interval.

In the absence of any DST issue can only assume that these cases where the “nk” time was prior to the BaseBlock/hbin time were caused by altering the date/time on the machines.

The fact the the hbin TimeStamps had been updated, hence showing that the files had been altered, without updating the “nk” time suggests that changes that do not directly affect the “nk” record do not affect the “nk” LastWriteTime. This might include changes to the data held in values or changes to a keys Security settings.

This could be of forensic significance.

This needs more work.

4.38 ClassNames

Until the database solution could be used no ClassNames where found (although no great effort was made to find them).

Once found their structure was the same very simple structure as used by Data Nodes. That is a cell which consists only of a Cell Size and then the Data. The Cell being padded out to an 8 byte boundary if needed.

A ClassName cannot be seen or edited using regedit.

It is known that Microsoft puts some security information into ClassNames possibly because they are hard to access. This includes the SysKey salt used to obscure password hashes in the SAM.

It was thought possible that a ClassName could be seen by exporting it as a text (.txt) file from Regedit. However when this was tried with a key that was known to have a ClassName (Computer 13, .Default file \Software \Microsoft \Windows NT\CurrentVersion\Program Manager\Common Groups) the text export still showed as “<NO CLASS>”.

On speculation that this might be a change in regedit or in the Windows operating system the same file was loaded into regedt32 running under NT4. The key was navigated to and saved using the menu option Registry->Save Subtree As… option and saved as a text file. The ClassName was correctly shown as “progman”.

This shows that the handling of ClassNames by the regedit (regedt32) programs has altered.

The values held in the ClassName records were briefly looked at. In all cases it was text as Unicode. There were 62,741 ClassName values which consisted of only 97 unique values. Of these 40 were text, 56 were 8 digit hex (lower case letters) and one was a simple zero (“0”).

No further analysis of the data held in ClassNames was attempted.

4.39 Record Signatures and Names

It is possible to try to assign names to records based on their Signatures.

This is not without its problems as it may lead to misleading names being given which will only serve to perpetuate misunderstandings. Some records can be given appropriate names with more confidence than others. Some records do not have signatures and so can be given any suitable name.

In the table below an attempt has been made to assign record names to the thirteen known record types. The names assigned to the SubKey lists ("lf", "lh", "li" and "ri") are quite speculative.

Signature	**Name**
regf	Registry File
hbin	Hive Bin
nk	Key Node
lk	Key Link
vk	Key Value
sk	Key Security
lf	Folder List
lh	Hash List
li	Index List
ri	Index Recursive
<no signature>	Value List
<no signature>	Data Node
<no signature>	Class Name

4.40 Pattern Matching

The original purpose of this Project was to see if it was possible to find Registry records in an arbitrary block of data such as might be recovered from unallocated space in the file system or from a pagefile.

This section discusses these possibilities now that a good understanding of the structure of Registry data has been gained.

4.40.1 “nk” Records

If we are looking for “nk” records we can bring to bear the following that we know about Registry records (or cells) and about the “nk” records in particular.

- All Registry cells start on an 8 byte boundary.
- All Registry cell sizes are a multiple of 8 bytes.
- The minimum size of an “nk” record is 0x50 plus the name (all keys have a name). That is 0x58 or 88 bytes minimum. We know that the maximum size of a Key Name is 255 characters (Microsoft, 2008a), if this is Unicode then that is 510 bytes. Therefore the maximum size of an “nk” record is 510 + 88 which is 600 bytes when rounded to next 8 byte boundary. From this we know that the first four bytes of an “nk” cell must be between “A8 FF FF FF” and “A8 FD FF FF”.
- The next two bytes must be “6E 6B” (“nk” in ASCII).
- The next two bytes are the Flags value in which only certain bits can be set. On up to Windows Server 2003 these are the bits set in the number 0x037E or 0x137E for Vista.

The above facts will find every “nk” record and it is plausible that there will be a low rate of false positives.

The above test can be extended by looking for plausible values in the known fields of the “nk” record. For example, the NameLength value should give, when added to 0x50 and rounded to the next 8 byte boundary, the Cell size (be aware of the risk that “nk” records that have been shrunk by having their name reduced may not be allocated a new cell). There are other tests that could be devised.

Of note is that the first test assumes that the sample data you are testing is still aligned on the same byte boundaries as was originally the case. It is expected that this will be so in most cases.

4.40.2 “vk” Records

Much the same as applies to “nk” records also applies to “vk” records.

A major difference is that “vk” records can be much bigger than “nk” records as the maximum allowed size of a Value name is nearly 16K (Microsoft, 2008a). This also increases the risk that a “vk” record will span more than one 4K block and so might not all be available.

This risk is mitigated by the fact that most Value names are of a more reasonable size. Looking at the “All Files” database (run 16) the largest Value name in that set of data was 276 bytes long. This is out of 2,241,475 Values. From this it would seem reasonable to set a limit of, say, 500 or 600 bytes which would then only exclude any “vk” records with the more outrageous Name lengths.

The “vk” record will normally hold the data if it is no more then four bytes long as is the case with DWORDS. This is likely to be useful as otherwise the Data Node has to be found to get the value held by the Value record.

4.40.3 ValueList Records

These records are going to be hard to find. They do not have a signature and can be any length (although the cell must be a multiple of 8 bytes in length). It has been observed that these records often have abandoned entries at the end of the cell.

The best way of finding these is probably going to be matching their Cell Index with an “nk” record and Value count and matching its entries with “vk” records. A tricky and error prone method when working with partial data.

Value Lists are particularly important as they link Keys and Values. They are needed to start to put a Key name to a Value and to find the Values belonging to a Key.

4.40.4 SubKey Lists

These do all have the same format of header which includes a signature and a count of the number of entries.

We do also know that the sizes of SubKey lists are limited by splitting them up using “ri” records and that “lf” records are changed to “li” records above a certain size.

“lf” records can probably be further identified by their bodies consisting of a pattern of pairs of Cell Index and up to four ASCII chars.

Larger SubKey lists do have a lot of empty space at the end of the cells, that is the cells are commonly much bigger than is needed to hold that number of entries.

Finally we know that large SubKey lists seem to come in a small number of standard sizes.

4.40.5 Data Nodes and Class Names

Both of these record types have the same simple format which is of the cell size followed by data. Class names are always Unicode as are text Data Nodes. This may assist their identification.

4.40.6 Determining Cell Indices

If a fragment of data is found with Registry cells in it then a basic problem is how to determine the Cell Index of each of these records.

If the associated hbin header can be found then it is easy as the "FileOffset" value in the header gives the base Cell Index. The Cell Index of an individual cell is then that FileOffset value plus the offset from the hbin header.

If we do not have the hbin header then we need to fall back on two other possible strategies. First of all that of locality, many cells refer to other nearby cells. The second possibility is that we can match this section of data with other section of data by processing the offset of each cell from the 4K block it is in (which might not be the first in the hbin) and the values of the Cell Indices that point to other cells. In other words Import and Export Cell Index values. This would be akin to piecing together a Ming vase that had been shattered and might be both time consuming and unreliable.

4.40.7 Conclusion

For all of these options more work is needed. What would also be needed is for any investigator tackling problems in these ways to judge carefully whether or not the amount of effort would be proportionate to the value of the evidence produced.

4.41 How Free Cells are Managed

It is to be expected that Registry cells will fall into disuse and so become free or deleted. This will happen sometimes when a Key or Value is modified.

A detailed discussion of how this area was investigated is contained in Appendix 38. The main conclusions are given here.

- There is a bit map for each of a range of cell sizes. Exact sizes (multiples of 8) up to 128 and then powers of 2 up to 2048 and then 2048 and above in one map.

- Free Cells are tracked by using the FreeDisplay element of the Storage element of the _HHIVE structure.
- This item is 24 elements long but calculations show only 20 elements are needed.
- Probert (Probert, 2005) states that free cells are chained together but evidence was found that this is not the case.

It is apparent that the management of free cells happens 'in memory' and not 'in file'. This is the clearest possible evidence that the Registry files are not the Registry but are merely a persistent backing store for the Registry which only really exists, in a live working state, in memory.

More work is needed in this area.

4.42 Jolanta Thomassen MSc Dissertation

In December 2008 the authorI was lucky enough to get a copy of the MSc Dissertation of Jolanta Thomassen (Thomassen, 2008) thanks to the authors' generosity. The subject of this is "Forensic Analysis of Unallocated Space in Windows Registry Hive Files".

A summary of this is shown in Appendix 39. Main points were.

- The focus was on finding deleted data
- She was hampered by not knowing about free cells having a positive cell size and by not knowing about the Length field in the BaseBlock.
- She has done some good work on what constitutes a valid Key Node record ("nk") or Key Value record ("vk").

An enjoyable read and recommended.

4.43 Summary

In this chapter the Experiments that were done have been explained and the results interpreted.

It has been shown how the level of knowledge and understanding of the Registry has been steadily improved through a logical exploitation of the information both available and discovered as the project progressed.

Enough detail has been provided to allow these experiments to be repeated and so allow for independent verification or challenge.

Chapter 5 – Results

In this Chapter a summary of the Results will be presented and it will be claimed that the results of this project can be considered to be reliable.

5.1 Introduction

The results can be split into two main types – facts or information and methods.

Facts or information can be further sub-divided into known facts that have been confirmed, disputed or contradictory explanations that have been resolved, errors or misunderstandings that have been corrected and new information that was not known before.

It is as well at this stage to say that even the newest information that has been discovered is not new at all. At the very least someone, somewhere within Microsoft must know all that is said in this entire thesis and more besides. This is in the nature of a reverse-engineering project such as this is. Much of this information must also be understood by the developers who have worked on the WINE, SAMBA and ReactOS projects (WINE, 2008b), (SAMBA, 2008a) and (ReactOS, 2008). I hope I have been able to make a contribution.

This is a good time to repeat that as this is a reverse-engineering process it is reliant on drawing assumptions from the observed data and its use. It is entirely possible that ranges of values and uses that have not been encountered are within the design of the Registry but as they have not been encountered it is not possible to consider them.

Finally, it is completely possible, and has been done with respect to the Registry in the past, to produce completely plausible and reasonable explanations that are quite simply wrong. The “Name Present” flag in the Key Value record (“vk”) being a perfect example of such a plausible but wrong explanation.

5.2 Achievements

This project has succeeded in the following ways.

5.2.1 Facts and Information

- Mapping every single used byte in Windows Registry files (Record Templates, Appendix 5).
- Putting a proper Microsoft name to nearly every field in Registry files (sections 4.4, 4.7, 4.11 to 4.15 and Appendix 12).
- Giving a good explanation of the purpose of nearly every field in Registry files (multiple places, mainly in chapter 4 and the Appendices).

- Understanding many of the more important internal memory Registry structures (sections 4.7, 4.10-4.15, 4.17, 4.21, 4.24, 4.25, 4.41).
- Showing how the Registry can be navigated in memory (section 4.17 and others).
- Showing how the “Registry” hive is used to map the windows Registry namespace to Hives (section 4.19, 4.21).
- Discovery of the “lk” record and its structure (section 4.21).
- Complete decomposition of the “sk” record and the Security Descriptor it contains (section 4.15).
- Understanding the Parent value in Root Cells (section 4.23).
- The proper meaning of the Flags field in “nk” (and “lk”) records and the derivation of the proper names for the bits used (section 4.20).
- Showing what the four “max” values in”nk” records are for (section 4.34).
- Deriving the hash used in “lh” records (section 4.29).
- Showing what “ri” and “li” records are used for (section 4.30).
- Showing how Unicode Value Names are stored (section 4.32).
- Showing the structure of Class Names (section 4.38).
- Explaining how dirty pages are flushed out and the use of the .LOG files section 4.26).
- Showing that very large hbins do occur (section 4.28)..
- Showing that “regf” and first “hbin” timestamps are the same (section 4.37).
- Pattern matching possibilities for finding cells in an arbitrary block of data (section 4.40).
- Showing how free cells are managed (section 4.41).
- Discovery and attempted partial decode of ‘little bits’ in Cell Map Table entries in memory (section 4.17.5 and Appendix 14).
- Working out how the “regf” checksum is calculated and what it covers (section 4.27).
- That Registry files have Major and Minor version numbers (sections 4.7 and 4.8.1).
- Which cell types can be found in which file versions (section 4.31).
- The Length field in Registry file headers that allows the ‘in use’ part of the Registry file to be determined (section 4.4).
- Confirmation that Registry Timestamps and LastWriteTimes are in UTC (Appendix 34).

- Discovery that the new Vista Registry files are version 1.3 and so fully compatible (in structure) with previously used Registry files (section 4.8.1).
- Discovery of Type 17 ACEs in Vista Registry Files. These are part of the new Microsoft "Mandatory Integrity Control" feature (Appendix 12 section 5).
- Discovery that Vista file "nk" records have a new flag bit value of 0x1000 (section 4.20).

5.2.2 Methods

- Production of examples and step-by-step guides to using the Windows Debugging Tools to examine and navigate Registry structures in memory.
- Development of paper Templates to assist manual decomposition.
- Development of the RegHoover program and Database and its associated scripts HooverLoad, BulkHoover and BulkHoover2. These allowed bulk collection and analysis of large quantities of data.
- Producing methods to extract Volatile hives and parts of hives from memory and into files.
- Providing the ability, through the RegHoover program, to analyse fragments of the Registry providing they start with a valid hbin header. This includes truncated hbins.
- The theoretical ability to analyse fragments of Registry data isolated from their hbin header is discussed but not implemented.

5.2.3 Programs

Some 13 programs and scripts were written in ANSI-C, VBScript and Visual Basic. These amounted, in volume, to over 8,000 lines including comments and blank lines.

5.2.4 Errors

The following errors, some common, were found in the work of others.

Error	Researcher (not exhaustive)
That the Size field in an hbin header is an offset with a zero in the last hbin. That hbins are so chained together,	This very common misconception seems to have originated with B.D. but has been widely promulgated by many including Williams, Probert, Jenkinson, Morgan and others.
That cell sizes are a multiple of 4 bytes. They are multiples of 8 bytes.	B.D and Williams.
That the “vk” record has a “Name Present” bit when the Value has a Name. It is in fact a “Compressed” bit meaning that the Name is ASCII and only occurs when there is a Name and it is ASCII	B.D., Williams, Macfarlane, Jenkinson and others
That dirty 512 byte sectors are flushed out to Registry Files. In fact it is 4K blocks that are flushed.	Russinovich
That the “regf” or BaseBlock Checksum is a Sum. It is an XOR.	B.D.
That the Flags value in “nk” records is a ‘Type’ value set to 0x2C for Root Cells and mainly 0x20 for others. It is in fact a bit-field with bit meanings as shown by this paper.	B.D, Williams, Macfarlane
That all cells start (after the size) with a signature. This is true of 8 cell types but not of three (ValueList, DataNode and ClassName)	Russinovich

5.2.5 Exceptions

The following fields in Registry files have not had their ‘proper’ Microsoft names assigned.

Record Type	Field
“lf”, “lh”	Hash
“lf”, “lh”, “ri”, “li”, ValueList	Offset
Data Node, Class Name	Data

It has not been possible to understand the purpose of the following Registry file fields.

Record Type	Field	Size
“nk”, “vk”	UserFlags	4 bits
	VirtControlFlags	4 bits
	Debug	1 byte
	WorkVar	4 bytes
“regf” or BaseBlock	Cluster	4 bytes
	BootType	4 bytes
	BootRecover	4 bytes

NB: BootType and BootRecover possibly never appear in a Registry file.

5.3 Reliability

The work produced here is designed to be completely reproducible.

The methods are accessible to anyone who wants to repeat or extend these experiments. Sufficient detail, including step-by-step guides and source code has been provided. The work is solidly based on previous, referenced, work by others and detailed experiments. As such is transparent, open to challenge and hence reliable.

The intention was to produce knowledge and understanding that forensic analysts could rely on due to being traceable back to original sources and reproducible when the result of experimentation. It is believed that this aim has been met.

5.4 Summary

In this chapter we set out to give an overview of what has been achieved.

A summary list was shown of the major advances in terms of facts and information. Another list showed the major methodological advances made.

Finally the reasons why the data produced by this project should be considered reliable were discussed.

Chapter 6 – Critical Analysis

In this Chapter the project will be reviewed in terms of what went right and what did not. What lessons could be learnt? How might it be done differently? How was the project managed? How useful that was?

6.1 Methodology Overview

The overall method was to proceed in the following steps

1. Carry out a comprehensive literature survey to capture and identify sources of information about the structure of the Registry. No attempt at this stage to review and understand the information except to assess for relevance.
2. Review the captured material in order to build an initial understanding and to find a route into the subject area.
3. The initial route chosen was to deeply analyse the Parse::Win32Registry module and how it parsed a SAM Registry file. This module was chosen as it was in contemporary use by other researchers and hence could be considered at least partly reliable – if it was getting things horribly wrong then this would be known about. The SAM Registry file was chosen as it was small enough to avoid the risk of becoming flooded with too much data and was sufficient for the purpose.
4. The paper Templates for each record type were developed. These allowed the current understanding to be captured and used and so form part of a learning circle of define, test, refine.
5. The excellent work by moyix on using the Windows Kernel Debugging tools to discover internal structures used for the Registry and to navigate them was exploited and refined. This produced most of the specific definitions of the Registry structures which was helped by already having a partial understanding of them.
6. A period then followed of understanding as much as possible by the detailed examination of relatively small amounts of data. A low volume high intensity detailed approach. This was fruitful but did eventually start to run dry. For example no Class Names had been encountered.
7. The development of RegHoover and its' associated programs. This was a deliberate attempt to move away from detailed examination of small quantities of data to spreading a wide net, collect together as much data as possible and then analyse it in order to see patterns and to detect previously rare data. This was very successful and led directly to many valuable discoveries.

On reflection the above direction seemed to work very well for this project.

6.2 Use of Literature Survey data

It is possible that more benefit could have been gained by studying more closely the information collected during the Literature Review.

This is true of the information collected from the large collaborative development projects such as SAMBA, WINE and ReactOS (SAMBA, 2008a), (WINE, 2008b), (ReactOS, 2008). This information would need to be treated with caution as they are all aiming, to a greater or lesser extent, for an emulation of Windows at the API level rather than at the file or internal structure level which is what would be useful for this project.

The analysis of the various public domain utilities collected was sparse and this again may have missed opportunities.

It is known that the Volatility project (Volatility Systems, 2008), which is devoted to analysing memory dumps, has a thread to do with finding Registry data and it is very likely that this contains valuable information. None of this material was collected or reviewed.

The reasons for the lack of exploitation were twofold. On the one hand the approach used rapidly produced good and authentic results. Further analysis of the existing work may well have not been of great benefit to the main thrust of the project although it might have revealed further misunderstanding to correct or puzzles to attempt to solve. The other reason was simply lack of time or the need to spend the available time wisely.

6.3 Other Researchers

Attempts were made to engage with other researchers but with little success. Early on there was little to share or ask about and later in the project the pace was too hectic. This may represent a missed opportunity.

6.4 Software Development Methodology

The software development methodology was an ad-hoc approach. At the level of complexity of these programs this is towards the limit of what should be attempted with this approach.

Program Specifications were produced for two programs that were not then written. No specifications or design documents or test plans were produced for any of the programs that have been written.

In part this can be explained on the basis that some of these were simple programs and others were experimental in nature and hence were moving targets in the sense that the specification was changing to match current understanding which was in turn driven by the output from the programs.

Guides to using RegHoover and HooverLoad have been produced and can be found at Supplements 5 and 6.

6.5 Study Methods

A review of the methods used for studying is given here.

1. A project folder was established on the authors desktop computers both at home and at work. Sub folders within this folder were established for important categories such as WebPages, Meetings, Write-Up, Source Code, Programs and so on.

   A USB stick was used for communication between these two project work locations. Synchronisation scripts were written that would synchronise the contents of the work folder on the USB stick with the desktop computers work folder in both directions. In this way work was able to progress at two locations and there were always three copies of the work available.

   Additionally a proper incremental (rather than merely snapshot) backup was made available at work (by dint of the normal enterprise backup system) and also at home due to use of a File Server and backup mechanism there. This meant that in the event of files being overwritten it was possible to step back to previous versions.

   By the end of the project the folder was over 12GB in size.

2. A document organiser program called Maple (Crystal Office, 2009) was used to organise contemporaneous notes. This provides hierarchical folders and a search mechanism. Most notes where held in a reverse date format with a brief name of date and subject as in “2008-11-01 sk cells”. These were collected into major folders named with just the month and year as in “2008-11”. Folders were also created in Maple to facilitate the management of the project such as follows.

   - Possible Experiments (for puzzles and items that need resolving)
   - Questions for project supervisor (Prep for next project meeting)
   - Don’t Forget (short reminders)
   - Apparent Errors (in others work)
   - Results (items for Results chapter)
   - Critical Review (items for Critical Review chapter)
   - Further Work (items for Further Work chapter)

3. Wherever possible notes were entered into Maple as it was done so as to avoid loss. Attempts were made to collect references as they were discovered, in case they might prove useful. Failing to always make such immediate notes was a weakness and time was wasted either repeating work or searching for elusive references. This is probably not a unique experience.
4. When doing research on the internet, useful pages were saved in a folder reserved for that purpose, in both HTML and MHT format if possible. A shortcut to the web page was also created and saved in that folder. This proved a very useful way of quickly finding and using relevant material as well as a way of capturing it. Material on the internet can disappear or change. This method was found to be very beneficial, particularly for the few resources that were heavily used.
5. A program called WebSite Extractor (InternetSoft Corporation, 2008) was used to extract the contents of web sites where there were a number of pages or where it proved difficult to save them from the browser. This worked well.
6. References were collected using a Web based service called RefWorks (RefWorks, 2008). This allows a common format to be enforced and for references to be sorted into user specified folders. This was not a very easy product to use. One criticism is that the entry forms had too many fields catering to too many options.
7. To aid concentration playing music, often loudly, was found to help in two ways. One was that it could provide stimulation and energy at lacklustre times. The other was that at times of higher than usual distractibility it seemed to aid concentration by filling in a void that might otherwise be filled with something (anything) interesting. Alternatively it helps by making increased effort on focusing on the intended subject essential.
8. At times of high attempted effort struggling against high distractibility a kitchen timer was found useful. A “Count Down / Up Timer”, product code RJ82D from Maplin (Maplin, 2008) was just right for the task. It was used to time sessions of work with the aim of making them at least 40 minutes long. It was sometimes found easy to think it was time for a break and to see from the timer that nowhere near enough time had passed to make this appropriate. This was a way of keeping the authors nose to the grindstone as well as making sure that breaks were taken at good intervals which aided concentration and learning.
9. The concept of Quality Time or uninterrupted hours was found important. When doing work that requires high amounts of intellectual effort it will take some time to ‘get into’ the subject and to build concentration to the level needed to produce good quality work. Even a small interruption can break concentration and put the researcher ‘back to square one’. The following is from the book “Controlling Software Projects” (DeMarco, 1982) in a section on how to count costs.

> "The basic unit of cost measurement is the *uninterrupted* hour. (This idea has a sobering implication for work environments in which there is no such thing as an uninterrupted hour – so be it.) The nature of systems development work is such that restart time is long, about twenty-five minutes after each interruption."

To mitigate this risk the authors email client was normally not run during work periods (and is always run without an audible announcement of new emails) and the phone was often unplugged.

6.6 Sample Registry Files

It was a clear weakness of the project that most of the example Registry files were relatively juvenile and so did not have the range of values and depth of data that might be expected. They did not have a representative amount of 'wear and tear' and usage that might be expected in real world data. This problem, of collecting Registry files, was also encountered by at least one other researcher (Thomassen, 2008).

6.7 Project Timing

The timing of this project was good. A year earlier and there would have been much less material to work on and it is doubtful that as much could have been achieved. In another year perhaps much of this material will have been discovered by others.

6.8 Project Management

The project was managed by a series of meetings between the Student and the Supervisor.

These were initially monthly which meant after about every 90-100 hours of work. When the project went 'full-time' these meetings were held more often, about every two weeks, so that the amount of work hours between meetings was still the same. Towards the end of the project it was sufficient to relax to about every three weeks.

The student always travelled to the Supervisor which was seen by the Student as making best use of the time of the Supervisor. It was felt best that the Supervisors effort be focussed on delivering valuable feedback and advice.

Notes were produced for each meeting and agreed at the next, such agreement being nearly always a formality. After a few meetings it was agreed to also carry a list of outstanding agreed Actions with the Notes.

As the project progressed and written work started to be produced review of this became an essential and valuable part of the meetings. There was some slight difficulty here as often material took a while to reach the Supervisor which meant on some occasions work submitted for one meeting being properly discussed at the one after. A sort of overlap of submission and review. This was not a major problem.

It is the authors view that such project meetings are an essential aid to the student providing as they do an interested but detached sounding board and guidance as to what aspects need attention at various stages of the project. It is thought to be important that such meetings are held every 80-100 hours through the project.

Managing and getting the most benefit out of such meetings is seen as a major factor that would determine the outcome of a project such as this.

6.9 Project Planning

Project planning was done by producing Hours Plans, Gantt Charts, To Do lists and keeping lists of Actions from the meetings.

6.9.1 Hours Plans

The Hours Plans were a narrative description and a spreadsheet with week-by-week hours totals on showing how the number of raw hours that the project needed was going to be made available and delivered.

Of course the project is about what is delivered and mere hours consumed do not of itself constitute a deliverable. However trying to do too much in not enough time is surely a plan for difficulty if not failure.

The number of Credits needed for the project is set by the University (Cranfield, 2008) at 80 (Cranfield, 2005) and the notional number of hours per Credit is stated as ten per credit by the Quality Assurance Agency for Higher Education (QAA, undated).

From the document "Academic credit in Higher Education in England" (QAA, 2006).

> "UK HEIs that use credit have agreed that one credit represents 10 notional hours of learning."

So the project part of this MSc therefore has a notional duration of 800 hours of learning.

Hours plans were developed which showed how, allowing for holidays and other interruptions, this amount of time could be delivered. This showed that at the work rate chosen the hours target could be met by Wednesday 10th December 2008 and had within it some 420 hours of slack before the project deadline. This hours plan also showed how, in a normal week, these hours would be delivered while still leaving one whole day a week free.

The Hours Plan was revised in mid-September, at which point only 200 hours of effort had been delivered. The end date was moved back to 19th December with 351 hours of slack available to the hand-in date.

The author considers this hours-delivered based approach to have been a vital factor contributing to the successful management of the project.

The project is believed to have had no more than the notional 800 hours of effort applied to it. A rough break-down is as follows.

Component	**Approx Hours**
Literature Search	200
Experiments	250
Write-Up	300
Meetings	50
TOTAL	**800**

The biggest variance was on the writing up which was budgeted at 20 hours per chapter for a total of 140 hours. Some chapters probably did take no more than 20 hours but some were very much more.

If this project was run again the author would reduce the number of days worked on the project and increase the size of the work blocks. That is instead of, say, 4 nights a week at 2 hours a night reduce that to three nights a week at 2½ hours a night or even two nights at 4 hours a night. This is to have a smaller number of occasions to focus effort and to provide more uninterrupted hours.

6.9.2 Quality Time

This project did suffer from difficulties 'getting started' to the extent that by August the project was in some trouble.

It was eventually accepted that the major cause of this was starting the project at the same time as starting a new job which led to too many demands on the author. Fortunately it was possible to work full time on the project from mid-September onwards and this proved very beneficial in three ways.

- It proved possible to devote time in large blocks, in other words to have a lot of uninterrupted hours.
- The time that was spent was good quality time in that it was the prime time of the working week. It was not 20 hours on top of 40 hours already spent. Working long hours leads to fatigue and loss of productivity and concentration – work that makes intellectual demands is the first to suffer.

- The simpler lifestyle meant that there were less distractions and other calls on time and effort.

6.9.3 Gantt Charts

A list of tasks was developed and a rough time in hours assigned to each. They were grouped into phases and converted into a Gantt Chart using Microsoft Project.

These Gantt Charts were produced initially and at the stage when the project went full-time.

They were of most use at the beginning of the project when the ‘finish line’ was over the horizon and so a good road map was needed to plan how to get started.

Initially, with few tasks running, it was easy to track progress against the plan. As the project progressed and multiple parallel tasks came on stream the amount of effort needed, and the accuracy of any possible results, made this technique harder to use and less effective.

6.9.4 To Do Lists

These were simple lists of short term tasks. As the complexity of the project increased it was increasingly used to manage the day-to-day complexity of the project and to try to prevent items of work from being missed.

They became more useful as the project progressed.

6.9.5 Action Lists

The meetings naturally started to produce Actions and these were not always done by the next meeting.

To manage this Action lists were maintained with the Notes of each meeting. Actions were numbered and a the date of the meeting they were raised at recorded along with the date of the meeting at which it was resolved. Also recorded was the person whose action it was although this was of course rather one-sided.

Towards the end of the project it became a task to clear the actions and this was achieved by either completing the action, deciding it was no longer relevant or by simply abandoning it.

This was an inevitable and useful adjunct to having project meetings.

6.10 Difficulty, Enjoyment and Value

Just a quick section on approximate relative amounts of difficulty, enjoyment and value.

Component	Difficulty	Enjoyment	Value
Literature Search	Medium	Medium	High
Experiments	High	High	High
Write-up	High	Low while doing it	High
Project Planning	Medium	Low	High
Project Meeting Admin	Low	Low	Medium
Project Meetings	Low	High	High

6.11 Doing too much

This project has suffered from tackling too much. This has resulted in a heavy burden of writing up and reduced the time and space available to more thoroughly discuss the results and the project.

The only realistic option for curtailing the scope would have been to have not developed the RegHoover set of programs or else not to have followed up so many of the leads and opportunities that was so produced. This would have been a large missed opportunity and in the view of the author regrettable.

The project has been delivered with an acceptable quality of presentation and write-up and within the time budget but significantly larger than was expected.

6.12 Summary

In this chapter we have examined the factors which helped and those which have hindered this project.

We saw the importance of good Methodology, Project Planning and Project Management as well as a range of other challenges and solutions.

Chapter 7 – Conclusion

In this chapter a brief summary of the conclusions is given and opportunities for further work will be outlined.

7.1 Conclusions

- The main conclusion that can be drawn from this work is that the Registry does not exist in the Registry files but only really exists in memory. This is evidenced by the management of free cells being entirely in memory and also by the fact that crucial data structures such as the Registry hive only exist in memory. The conclusion is that the Registry files are merely backing store to provide the Registry with persistence. This is not to say that they are not valid sources of data as they are normally flushed out every five seconds.
- The method of taking a contemporary and trusted analysis tool and deconstructing it to provide a launch platform for further research has shown itself to have been valid.
- The method of using the Windows Debugging Tools to unravel Windows Data structures has been successfully exploited to provide a complete mapping of Registry file data structures.
- The use of a custom program to dissect Registry files and store the data in a database coupled with custom programs and queries to analyse that data has proven a valid and powerful method of generalising a basic knowledge and allowing a new perspective.
- It has been plausibly postulated that Registry Keys and Key Values and possibly other records can be found in any arbitrary block of data.
- Significant lessons have been learnt about the organisation and management of such a relatively large project.

7.2 Further Work

This has been a productive project and many aspects of the subject area have been laid bare. There is always more that can be done and some suggestions for further work follows.

1. The RegHoover program can analyse whole Registry files or whole hbins. It can also analyse hbins which have been truncated at the end, that is which still have their header intact. It may be feasible to extend RegHoover to analyse hbins fragments that do not have a header. This might happen where the header has become overwritten by a file (data

recovered from file slack) or if the 4K block is other than the first one of the hbin.

2. Patterns were defined for searching for Key Node and Key Value cells in an arbitrary block of data. This work needs to be practically tested and other strategies tried to identify Registry data wherever it might be found.
3. The programs could be improved. The HooverLoad script (which has quite a long run time) could be converted to a .exe file, perhaps as Visual Basic. This should yield a performance improvement. Other scripts that are long running such as TraverseData could also be converted to .exe.
4. The database could be converted to SQL Server or some other 'large' database application. This might allow load time speed improvements and would allow more data to be held.
5. The RegHoover/HooverLoad process can process volatile hives that have been extracted from memory. This process could be improved to hold Volatile Cell Indices as positive values by having for each Cell Index a flag that states that it was originally Volatile. Could also have a single row table to give the creation date of the database.
6. During this work a new "nk" record Flags bit of 0x1000 was found. What is this for? Its name could probably be determined by running the Kernel Debugger on a Vista machine and repeating the experiment to determine flag values.
7. It was discovered that sometimes hbins can be very large. Why is this? What do they contain? How do they come about? Is it a side-effect of merely concatenating adjacent free hbins and so a result of churn?
8. It was determined how the "lh" hash value is calculated for ASCII Key names. How is the Hash value calculated in "lh" records when the SubKey Name is in Unicode?
9. It was found that in version 1.3 registry files that "li" and "ri" records start to be used when the number of subkeys becomes large. At what values do these records start to be used?
10. It was found that in version 1.5 registry files that "ri" records start to be used when the number of subkeys becomes large. At what value does this record start to be used?
11. It was discovered that "li" and large "lh" cells have a small number of sizes. Do cells get allocated in fixed step sizes above a certain size?
12. RegHoover reports that some Data Nodes are apparently too small? Why is this? Is it a fault in RegHoover or an unexplained feature or anomaly.
13. The 010 Template Editor is designed to allow Hex records to be parsed and displayed. A useful task would be to develop templates for use with this tool.

14. It was found that normally Key Node time stamps (Last Write Time) are before the times in the BaseBlock/first hbin. However some are not. Why is this?
15. Pattern definitions were defined. More work could be done on these to see what patterns can be found in SubKey lists to allow them to be searched for. More work in general on successful pattern matching.
16. How can we determine the Cell Index of cells found in blocks of data with no hbin header? Normally we can use the Offset field in the hbin header to act as a base Cell Index for that hbin.
17. Much work was done on how Free Cells are managed in memory. It remains a puzzle as to whether there should be 20 or 24 elements in a FreeDisplay bitmap? How are free cells managed?
18. Many of the important ‘in memory’ data structures were unravelled and explained. What are the others for?
19. The proper Microsoft names were found for nearly all the fields of Registry files. What are the ‘proper’ Microsoft names for the Hash and Offset values found in SubKey Lists and the Value List?
20. The field names for all the fields in “nk”, “lk” and the BaseBlock records were found. The use of nearly all of these was also deduced. What are the few unknown fields found in “nk”/”lk” records and the Cluster value in the BaseBlock used for?
21. A fairly large set of Registry files were used in this project but they were not intended to be a statistically valid sample. It was also not easy to get well-used Registry files. It would be of interest to find a better, more typical, set of Registry files to examine. (Units and Organisations with significant caseload may be able to use Registry files from previous cases with an obvious confidentiality restriction).
22. Probert refers to a No Lazy Flush setting for a hive. Where is this?
23. BaseBlock Type settings were identified and found to be attributable to the PRIMARY file or the LOG file. What file or structure uses the BaseBlock type HFILE_TYPE_EXTERNAL?

7.3 Summary

In this chapter the main conclusions were presented and a list of possible areas of further work have been outlined.

References

ActiveState (2008a), *ActivePerl - The complete and ready-to-install Perl distribution.*, available at: http://www.activestate.com/Products/activeperl/index.mhtml (accessed 2008/11/20).

ActiveState (2008b), *Komodo IDE 5.0 Overview*, available at: http://www.activestate.com/Products/komodo_ide/index.mhtml (accessed 2008/11/20).

Allen, J. (2008a), *Perl 5.10.0 documentation, unpack()*, available at: http://perldoc.perl.org/functions/unpack.html (accessed 2008/10/14).

Allen, J. (2008b), *Perl 5.10.0 documentation, encode()*, available at: http://perldoc.perl.org/Encode.html (accessed 2008/10/25).

Anand, G. (2008), *Internal structures of the Windows Registry*, available at: http://blogs.technet.com/ganand/archive/2008/01/05/internal-structures-of-the-windows-registry.aspx (accessed 2008/10/31).

B.D. , *WinReg.txt*, available at: http://home.eunet.no/~pnordahl/ntpasswd/WinReg.txt (accessed 2008/7/10).

Boling, D. (2001), *Programming Windows CE,* 2nd ed, Pages 998, Microsoft Press , ISBN:0735614431.

Carvey, H. (2005), "The Windows Registry as a forensic resource", *Digital Investigation,* vol. 2, no. 3, pp. 201-205.

Carvey, H. (2006a), *New Today*, available at: http://windowsir.blogspot.com/2006/10/new-today.html (accessed 2008/10/13).

Carvey, H. (2006b), *Parsing Raw Registry Files*, available at: http://windowsir.blogspot.com/2006/11/parsing-raw-registry-files.html (accessed 2008/10/13).

Carvey, H. (2007a), *SAM Parse*, available at: http://windowsir.blogspot.com/2007/01/samparse.html (accessed 2008/10/13).

Carvey, H. (2007b), *Scripts for parsing the Registry*, available at: http://windowsir.blogspot.com/2007/01/scripts-for-parsing-registry.html (accessed 2008/10/13).

Carvey, H. (2007c), "Registry Analysis", in *Windows Forensic Analysis,* Syngress, Rockland, pp. 125-189.

Carvey, H. (2008a), *Windows Incident Response Blog*, available at: http://windowsir.blogspot.com/ (accessed 2008/10/9).

Carvey, H. (2008b), *RegRipper Update*, available at: http://windowsir.blogspot.com/2008_05_01_archive.html (accessed 2008/10/9).

Chen, R. (2004), *The Old New Thing*, available at: http://blogs.msdn.com/oldnewthing/archive/2004/03/15/89753.aspx (accessed 2008/10/11).

Clark (2005), *Security Accounts Manager - Registry Structure*, available at: http://www.beginningtoseethelight.org/ntsecurity/ (accessed 2008/10/9).

CPAN *, Comprehensive Perl Archive Network*, available at: http://search.cpan.org/.

Cranfield. (2005), *Msc, PGDip & PGCert in Forensic Computing, Student Handbook, Section Three* .

Cranfield (2007), *Cranfield University*, available at: http://www.cranfield.ac.uk/ (accessed 2009/1/3).

Crystal Office (2009), *Maple*, available at: http://www.crystaloffice.com/maple/ (accessed 2009/1/3).

Cuomo, N. *, Personal web page*, available at: http://www.studenti.unina.it/~ncuomo/syskey/ (accessed 2008/10/8).

DeMarco, T. (1982), *Controlling Software Projects,* Pages 284, Englewood Cliffs , New Jersey, ISBN:0-13-171711-1.

DFRW "Digital Forensic Research Workshop 2008", August 2008, Baltimore, MD, .

Dolan-Gavitt, B. (2008), "Forensic analysis of the Windows registry in memory", *Digital Investigation,* vol. 5, no. Supplement 1, pp. S26-S32.

GTK+ *, The GTK+ Project*, available at: http://www.gtk.org/ (accessed 2008/10/9).

Guidance (2008), *Guidance Software Inc.*, available at: http://www.guidancesoftware.com (accessed 2088/10/9).

Hague, S. and Wood, C. *, Designing a Hard Disk Driver for Windows CE 2.12*, available at: http://users.ece.gatech.edu/~hamblen/489X/projects/disk/index.html (accessed 2008/10/9).

Helix (2008), *e-fence Inc.*, available at: http://helix.e-fense.com/ (accessed 2008/10/9).

InternetSoft Corporation (2008), *Website Extractor*, available at: http://www.internet-soft.com/extractor.htm (accessed 2009/1/3).

ISO (2004), *International Organization for Standardization 8601:2004*, available at: http://www.iso.org/iso/catalogue_detail?csnumber=40874 (accessed 2008/10/24).

Jenkinson. (2008), *Advanced Forensics* (unpublished Short Course), Cranfield University, Shrivenham.

Macfarlane, J. (2008), *Parse-Win32Registry-0.40*, available at: http://search.cpan.org/~jmacfarla/Parse-Win32Registry-0.40/ (accessed 2008/10/9).

Maplin (2008), *Maplin Electronics Ltd*, available at: http://www.maplin.co.uk (accessed 2009/1/3).

Martelli, A. (2006), *Python in a nutshell,* 2nd ed, O'Reilly , Sebastopol CA, ISBN:0-596-10046-9.

Microsoft (2003a), *Windows Server 2003 Resource Kit Tools*, available at: http://www.microsoft.com/downloads/details.aspx?familyid=9d467a69-57ff-4ae7-96ee-b18c4790cffd&displaylang=en (accessed 2008/10/9).

Microsoft (2003b), *Adsiedit Overview*, available at: http://technet.microsoft.com/en-us/library/cc773354.aspx (accessed 2008/10/11).

Microsoft (2003c), *How Security Descriptors and Access Control Lists Work*, available at: http://technet.microsoft.com/en-us/library/cc781716.aspx (accessed 2008/10/31).

Microsoft (2003d), *Unicode and Keyboards on Windows, 23rd Internationalization and Unicode Conference, Prague, Czech Republic, March 2003*, available at: http://www.microsoft.com/globaldev/handson/dev/Unicode-KbdsonWindows.pdf (accessed 2008/12/04).

Microsoft (2006a), *FIX: Audio playback includes unwanted noises for a multimedia application*, available at: http://support.microsoft.com/kb/839562 (accessed 2008/11/18).

Microsoft (2006b), *Missing Objects and Counters in Performance Monitor*, available at: http://support.microsoft.com/kb/127207.

Microsoft (2007a), *Understanding Container Access Inheritance Flags in Windows 2000*, available at: http://support.microsoft.com/kb/220167 (accessed 2008/10/8).

Microsoft (2007b), *Debug Interface Access SDK*, available at: http://msdn.microsoft.com/en-us/library/x93ctkx8.aspx (accessed 2008/10/11).

Microsoft (2007c), *Strings v2.40*, available at: http://technet.microsoft.com/en-us/sysinternals/bb897439.aspx (accessed 2008/11/20).

Microsoft (2008a), *Registry Element Size Limits*, available at: http://msdn.microsoft.com/en-us/library/ms724872.aspx (accessed 2008/10/8).

Microsoft (2008aa), *HKEY_CURRENT_CONFIG*, available at: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/69675.mspx (accessed 11/3/.

Microsoft (2008ab), *HKEY_CLASSES_ROOT*, available at: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/69675.mspx (accessed 11/3/.

Microsoft (2008ac), *Registry Value Types*, available at: http://msdn.microsoft.com/en-us/library/ms724884(VS.85).aspx (accessed 2008/11/4).

Microsoft (2008ad), *PDB Files*, available at: http://msdn.microsoft.com/en-us/library/yd4f8bd1(VS.71).aspx (accessed 2008/11/18).

Microsoft (2008ae), *SYSTEM_MANDATORY_LABEL_ACE*, available at: http://msdn.microsoft.com/en-us/library/cc230379(PROT.10).aspx (accessed 2009/5/1).

Microsoft (2008af), *Mandatory Integrity Control*, available at: http://msdn.microsoft.com/en-us/library/bb648648(VS.85).aspx (accessed 2009/1/5).

Microsoft (2008b), *ACCESS_MASK Data Type*, available at: http://msdn.microsoft.com/en-gb/library/aa374892.aspx (accessed 2008/10/8).

Microsoft (2008c), *ADS_ACEFLAG_ENUM Enumeration*, available at: http://msdn.microsoft.com/en-us/library/aa772242.aspx (accessed 2008/10/8).

Microsoft (2008d), *Registry Key Security and Access Rights*, available at: http://msdn.microsoft.com/en-gb/library/ms724878.aspx (accessed 2008/10/8).

Microsoft (2008e), *Security identifiers*, available at: http://technet.microsoft.com/en-us/library/cc780850.aspx (accessed 2008/10/8).

Microsoft (2008f), *Windows Embedded CE*, available at: http://msdn.microsoft.com/en-us/library/bb847932.aspx (accessed 2008/10/9).

Microsoft (2008g), *About Visual Studio*, available at: http://msdn.microsoft.com/en-us/vstudio/products/bb931214.aspx (accessed 2008/10/11).

Microsoft (2008h), *Windows Kernel-Mode Configuration Manager*, available at: http://msdn.microsoft.com/en-us/library/cc264540.aspx (accessed 2008/10/11).

Microsoft (2008i), *Debugger Commands*, available at: http://msdn.microsoft.com/en-us/library/cc266538.aspx (accessed 2008/10/11).

Microsoft (2008j), *Windows registry information for advanced users*, available at: http://support.microsoft.com/kb/256986 (accessed 2008/10/13).

Microsoft (2008k), *Debugging Tools for Windows - Overview*, available at: http://www.microsoft.com/whdc/devtools/debugging (accessed 2008/10/27).

Microsoft (2008l), *Absolute and Self-Relative Security Descriptors*, available at: http://msdn.microsoft.com/en-us/library/aa374807.aspx (accessed 2008/10/31).

Microsoft (2008m), *SECURITY_DESCRIPTOR Structure*, available at: http://msdn.microsoft.com/en-us/library/aa379561(VS.85).aspx (accessed 2008/10/31).

Microsoft (2008n), *SECURITY_DESCRIPTOR_CONTROL Data Type*, available at: http://msdn.microsoft.com/en-us/library/aa379566(VS.85).aspx (accessed 2008/10/31).

Microsoft (2008o), *ACL Structure*, available at: http://msdn.microsoft.com/en-us/library/aa374931(VS.85).aspx (accessed 2008/10/31).

Microsoft (2008p), *ACE Data Type*, available at: http://msdn.microsoft.com/en-us/library/aa374912(VS.85).aspx (accessed 2008/10/31).

Microsoft (2008q), *ACE_HEADER Structure*, available at: http://msdn.microsoft.com/en-us/library/aa374919(VS.85).aspx (accessed 2008/11/31).

Microsoft (2008r), *ACCESS_ALLOWED_ACE Structure*, available at: http://msdn.microsoft.com/en-us/library/aa374847(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008s), *ACCESS_DENIED_ACE Structure*, available at: http://msdn.microsoft.com/en-us/library/aa374879(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008t), *SYSTEM_AUDIT_ACE Structure*, available at: http://msdn.microsoft.com/en-us/library/aa379616(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008u), *ACCESS_MASK Data Type*, available at: http://msdn.microsoft.com/en-us/library/aa374892(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008v), *Security Identifiers*, available at: http://msdn.microsoft.com/en-us/library/aa379571(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008w), *Well-known SIDs*, available at: http://msdn.microsoft.com/en-us/library/aa379649(VS.85).aspx (accessed 2008/11/1).

Microsoft (2008x), *Parts of a Security Descriptor*, available at: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/distrib/dsce_ctl_dbvr.mspx (accessed 2008/11/1).

Microsoft (2008y), *Microsoft Most Valuable Professional*, available at: http://mvp.support.microsoft.com/ (accessed 200811/2/.

Microsoft (2008z), *HKEY_CURRENT_USER*, available at: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/51211.mspx?mfr=true (accessed 2009/5/1).

Morgan, T. D. (2008a), "Recovering deleted data from the Windows registry", *Digital Investigation,* vol. 5, no. Supplement 1, pp. S33-S41.

Morgan, T. D. (2008b), *Reglookup*, available at: http://projects.sentinelchicken.org/reglookup/ (accessed 2008/6/18).

moyix (2007), *Challenges in Carving Registry Hives from Memory*, available at: http://moyix.blogspot.com/2007/09/challenges-in-carving-registry-hives.html (accessed 2008/8/10).

moyix (2008a), *Enumerating Registry Hives*, available at: http://moyix.blogspot.com/2008/02/enumerating-registry-hives.html (accessed 2008/10/8).

moyix (2008b), *Reading OpenKeys*, available at: http://moyix.blogspot.com/2008/02/reading-open-keys.html (accessed 2008/10/8).

moyix (2008c), *CredDump: Extract Credentials from Windows Registry Hives*, available at: http://moyix.blogspot.com/2008/02/creddump-extract-credentials-from.html (accessed 2008/10/8).

moyix (2008d), *Cell Index Translation*, available at: http://moyix.blogspot.com/2008/02/cell-index-translation.html (accessed 2008/10/8).

moyix (2008e), *CredDump project*, available at: http://code.google.com/p/creddump (accessed 2008/10/8).

moyix (2008f), *Push The Red Button*, available at: http://moyix.blogspot.com (accessed 2008/10/27).

National Maritime Museum (2008), *British Summer Time (BST)*, available at: http://www.nmm.ac.uk/explore/astronomy-and-time/time-facts/british-summer-time-(bst) (accessed 2008/12/27).

Nordahl-Hagen, P. , *Offline NT Password & Registry Editor*, available at: http://home.eunet.no/~pnordahl/ntpasswd/ (accessed 2008/7/10).

NTFS4DOS , *NTFS4DOS Boot Disk*, available at: http://www.bootdisk.com/ntfs.htm (accessed 2008/11/20).

Perl Express Group (2007), *Perl Express: A Free Perl IDE/Editor for Windows*, available at: http://www.perl-express.com/ (accessed 2008/10/13).

Probert, D. B. , *Windows Kernel Internals*, available at: http://www.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Lectures/09-Registry/Registry.pdf (accessed 2008/10/9).

QAA , *The Quality Assurance Agency for Higher Education*, available at: http://www.qaa.ac.uk (accessed 2009/1/3).

QAA (2006), *Academic credit in Higher Education in England*, available at: http://www.qaa.ac.uk/academicinfrastructure/FHEQ/academicCredit/AcademicCredit.pdf (accessed 2009/1/3).

ReactOS (2008), *ReactOS Project*, available at: http://www.reactos.org (accessed 2008/10/8).

RefWorks (2008), *RefWorks -- an online research management, writing and collaboration tool*, available at: http://www.refworks.com/ (accessed 2009/1/3).

Rendell, M. , *regutils - win9x registry & ini file manipulation tools for unix*, available at: http://www.cs.mun.ca/~michael/regutils/ (accessed 2008/10/9).

Richards, J. (2007), *joeware.net*, available at: http://www.joeware.net (accessed 2008/11/2).

Riley, S. (2006)*, Mandatory integrity control in Windows Vista*, available at: http://blogs.technet.com/steriley/archive/2006/07/21/442870.aspx (accessed 2009/1/5).

Russinovich, M. E. *, Inside the Registry*, available at: http://www.microsoft.com/technet/archive/winntas/tips/winntmag/inreg.mspx.

Russinovich, M. E. and Solomon, D. (2005a), "Registry Internals", in *Microsoft Windows Internals,* 4th ed, pp. 197-211.

SAMBA (2005)*, Status of editreg*, available at: http://lists.samba.org/archive/samba-technical/2005-May/040851.html (accessed 2008/11/20).

SAMBA (2008a)*, Samba Home Page*, available at: http://us1.samba.org/samba/ (accessed 2008/10/8).

SAMBA (2008b)*, SAMBA download*, available at: http://us1.samba.org/samba/ftp/samba-latest.tar.gz (accessed 2008/10/8).

Schwartz, R. L., Phoenix, T. and foy, b. d. (2005), *Learning Perl,* 4th ed, O'Reilly , Sebastopol, CA, ISBN:0-596-10105-8.

Screiber, S. B. (2001), *Undocumented Windows 2000 Secrets,* Pages 563, Addison-Wesley , ISBN:0-201-72187-2.

Screiber, S. B. (2008)*, Updated PDB Exploder*, available at: http://undocumented.rawol.com/win_pdbx.zip (accessed 2008/10/13).

Sourceforge (2008a)*, Sourceforge WINE project*, available at: http://sourceforge.net/projects/wine (accessed 2008/10/8).

Sourceforge (2008b)*, Sourceforge ReactOS project*, available at: http://sourceforge.net/projects/reactos (accessed 2008/10/8).

Thomassen, J. (2008), *Forensic Analysis of Unallocated Space in Windows Registry Hive Files,* The University of Liverpool.

Titheridge, D. A. (2008), *Microsoft Windows Vista* (unpublished MSc thesis), Cranfield University.

Tittel, E., Stewart, J. M. and Chapple, M. (2004), "Access Control Techniques", in *CISSP Study Guide,* , pp. 16.

Volatile Systems (2008)*, The Volatility Framework: Volatile memory artifact extraction utility framework*, available at: https://www.volatilesystems.com/default/volatility (accessed 2009/1/3).

Wall, L., Christianson, T. and Orwant, J. (2000a), *Programming Perl,* 3rd ed, O'Reilly , Sebastopol, CA, ISBN:0-596-00027-8.

Wall, L., Christianson, T. and Orwant, J. (2000b), "CPAN", in *Programming Perl,* 3rd ed, O'Reilly, Sebastopol, CA, pp. 547-556.

Williams, N. (2000)*, dosreg.c*, available at: http://www.wednesday.demon.co.uk/dosreg.html (accessed 2008/17/10).

WINE (2008a)*, World Wine News Issue #348*, available at: http://www.winehq.org/site?issue=348#News:%20Wine%201.0! (accessed 2008/10/8).

WINE (2008b)*, Wine HQ*, available at: http://www.winehq.org/ (accessed 2008/10/8).

X-Ways (2008)*, Winhex: Hex Editor and Disk Editor*, available at: http://www.x-ways.net/winhex/ (accessed 2008/10/9).

Bibliography

Adelstein, F. and Joyce, R. A. (2007), "File Marshal: Automatic extraction of peer-to-peer data", *Digital Investigation,* vol. 4, no. Supplement 1, pp. 43-48.

Ahmed, M., Garrett, C., Faircloth, J., Payne, C., Lee, W. M. and Ortiz, J. (2002), "Configuring ASP.NET", in *ASP .NET Web Developer's Guide,* Syngress, Rockland, pp. 173-225.

Anderson, H. and Tooley, M. (1999), "Troubleshooting windows registry", in *Newnes PC Troubleshooting Pocket Book (Third edition),* Newnes, Oxford, pp. 155-158.

Bhardwaj, P. K. (2006), "Monitoring System Events, Processes, and Performance", in *How to Cheat at Windows System Administration Using Command Line Scripts,* Syngress, Burlington, pp. 241-272.

Bhardwaj, P. K. (2006), "System Services, Drivers, and the Registry", in *How to Cheat at Windows System Administration Using Command Line Scripts,* Syngress, Burlington, pp. 205-240.

Born, G. (1998), *Inside the Microsoft Windows 98 Registry,* Microsoft Press , Redmond, ISBN:1-57231-824-4.

Brown, A. (2000), *The How to Study Book,* Barricade Books , New York.

Carrier, B. (2005), *File System Forensic Analysis,* Pages 569, Pearson Education , Upper Saddle River NJ, ISBN:0-32-126817-2.

Carvey, H. (2005), *Windows Forensics and Incident Response,* Addison-Wesley, Boston, ISBN:0-321-20098-5.

Craft, M. and Llewellyn, J.,Thomas D. (2001), "Using active directory: A case study", in Melissa Craft, Thomas D. Llewellyn and Jr. (eds.) *Windows 2000 Active Directory (Second Edition),* Syngress, Rockland, pp. 501-519.

Dodge, R. C. "Skype Fingerprint", *Proceedings of the 41st Annual Hawaii International Conference on System Sciences,* 7-10 Jan. 2008, pp. 484.

Drew, S. and Bingham, R. (1997), *The Student Skills Guide,* Gower , Aldershot, ISBN:0-566-07857-3.

Ganapathi, A., Yi-Min, W., Ni, L. and Ji-Rong, W. (2004), "Why PCs are fragile and what we can do about it: a study of Windows registry problems", *2004 International Conference on Dependable Systems and Networks,* 28 June-1 July 2004, California Univ., Berkeley, CA, USA, pp. 561.

Hanner, K. and Hörmanseder, R. (1999), "Managing Windows NT®file system permissions— A security tool to master the complexity of Microsoft Windows NT®file system permissions", *Journal of Network and Computer Applications,* vol. 22, no. 2, pp. 119-131.

Hartman, T. (1997), *Attention Deficit Disorder - A Different Perception,* Second ed, Underwood Books , Grass Valley, California, ISBN:1-887424-14-8.

Highland, H. J. (1997), "Windows NT registry", *Computers & Security,* vol. 16, no. 2, pp. 88-90.

Hoffman, P. (2003), *Perl for Dummies,* 4th ed, Pages 381, Wiley , New York, ISBN:0-7645-3750-4.

Kelly, K. and Ramundo, P. (2006), *You Mean I'm Not Lazy, Stupid or Crazy?!* Pages 460, Scribner , New York, ISBN:0-7432-6448-7.

Kernigham, B. W. and Plauger, P. J. (1981), *Software Tools in Pascal,* Pages 366, Addison-Wesley , Reading MA, ISBN:0-201-10342-7.

Kernigham, B. W. and Ritchie, D. M. (1988), *The C Programming Language,* Second ed, Pages 272, Prentice Hall , Englewood Cliffs NJ, ISBN:0-13-110362-8.

Kisik, C., Gibum, K., Kwonyoup, K. and Woosuk, K. "Initial Case Analysis Using Windows Registry in Computer Forensics", *Future generation communication and networking,* vol. 1, no. 6-8, pp. 564-564 - 569.

Kokoreva, O. (2002), *Windows XP Registry,* Pages 530, A-List , Wayne PA, ISBN:1-931769-01-X.

Mee, V., Tryfonas, T. and Sutherland, I. (2006), "The Windows Registry as a forensic artefact: Illustrating evidence collection for Internet usage", *Digital Investigation,* vol. 3, no. 3, pp. 166-173.

Murray, R. (2003), *How to Survive your Viva,* Open University Press , Maidenhead, ISBN:0-335-21284-0.

Nikkel, B. J. (2007), "An introduction to investigating IPv6 networks", *Digital Investigation,* vol. 4, no. 2, pp. 59-67.

Robichaux, P. (1998), "Managing the windows NT registry", *Computers & Mathematics with Applications,* vol. 36, no. 3, pp. 125-125.

Robichaux, P. (2000), *Managing the Windows 2000 Registry,* O'Reilly & Associates , Sebastopol, CA, ISBN:1-56592-943-8.

Saito, Y., Bershad, B. N. and Levy, H. M. (2000), "Manageability, availability, and performance in porcupine: a highly scalable, cluster-based mail service", *ACM Trans.Comput.Syst.,* vol. 18, no. 3, pp. 298.

Sammes, T. and Jenkinson, B. (2007), *Forensic Computing - A Practitioners Guide,* Second ed, Pages 465, Springer , London, ISBN:1-84628-397-3.

Schultz, E. (2003), "Security Views", *Computers & Security,* vol. 22, no. 5, pp. 368-377.

Shinder, M. D. T. W., Shinder, D. L. and Grasdal, M. (2002), "Using the Security Configuration Tool Set", in *Dr. Tom Shinder's ISA Server and Beyond,* Syngress, Burlington, pp. 487-531.

Todd, C. and Johnson, J.,Norris L. (2001), "Default Access Control Settings", in Chad Todd, Norris L. Johnson and Jr. (eds.) *Hack Proofing Windows 2000 Server,* Syngress, Rockland, pp. 21-61.

Todd, C. and Johnson, J.,Norris L. (2001), "Security Configuration Tool Set", in Chad Todd, Norris L. Johnson and Jr. (eds.) *Hack Proofing Windows 2000 Server,* Syngress, Rockland, pp. 149-198.

Todd, C. and Johnson, J.,Norris L. (2001), "Using Security-Related Tools", in Chad Todd, Norris L. Johnson and Jr. (eds.) *Hack Proofing Windows 2000 Server,* Syngress, Rockland, pp. 535-616.

Topallar, M., Depren, M. O., Anarim, E. and Ciliz, K. "Host-based intrusion detection by monitoring Windows registry accesses", *2004. Proceedings of the IEEE 12th Signal Processing and Communications Applications Conference,* 28-30 April 2004, pp. 728.

Xueqin, Z., Chunhua, G. and Jiajun, L. (2006), "Support Vector Machines for Anomaly Detection", *The Sixth World Congress on Intelligent Control and Automation, 2006. WCICA 2006.* Vol. 1, pp. 2594.

Youngsoo, K. and Dowon, H. "Windows Registry and Hiding Suspects' Secret in Registry", *ISA 2008. International Conference on Information Security and Assurance,* 24-26 April 2008, pp. 393.

Zenkin, D. (2000), "Anti-virus software reports on Windows registry changes", *Computer Fraud & Security,* vol. 2000, no. 6, pp. 6-6.