



# 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>



**GIAC Certified Forensic Analyst  
Practical Assignment  
(GCFA)  
v1.5  
James D. Perry II**

Submitted: November 20, 2004

## Table of contents

<b>ABSTRACT</b>	<b>3</b>
<b>PART ONE: Analyze an Unknown Image</b>	<b>4</b>
Letter to Mr. David Keen of Ballard Industries	6
Forensic Analysis Report for Ballard Industries	6
<i>Executive Summary</i>	6
<i>Examination Details</i>	6
<i>Examination Conclusion</i>	27
<i>Appendix A-1: a:\_index.htm Strings Report</i>	32
<i>Appendix A-2: a:\CamShell.dll Strings Report</i>	33
Part One References	40
<b>PART TWO: Option 2 – Perform Forensic Tool Validation</b>	<b>41</b>
Validation of Compromised Computer Inventory System (CCIS)	42
<i>Executive Summary</i>	42
<i>Scope</i>	43
<i>Tool Description</i>	45
<i>Test Apparatus, Environmental Conditions, and Procedures</i>	70
<i>Criteria for Approval</i>	73
<i>Results, Analysis, and Presentation</i>	73
<i>Conclusion</i>	74
<i>Appendix A: ccis_collector script</i>	77
<i>Appendix B: ccis_sender script</i>	85
<i>Appendix C: ccis_sender.bat</i>	87

Part Two References

88

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

## ABSTRACT

The GIAC Certified Forensic Analyst (GCFA) certification requires that each candidate complete a comprehensive practical to demonstrate, as stated on the GIAC website (<http://www.giac.com/GCFA.php>), that the candidate has the “necessary knowledge, skills, and abilities to handle advanced incident handling scenarios, conduct formal incident investigations, and carry out forensic investigation of networks and hosts.” The purpose of this practical is to achieve these objectives, identified above, and to fulfill a requirement for the GCFA certifications.

Part one of this practical contains a forensic analysis report that was generated after analyzing an unknown image. The report outlines the methodology of investigation and a step-by-step outline of the tasks performed to identify the content and purpose of a floppy disk. The knowledge, skill, and ability of the forensic analyst are made clear in the analysis

Part two of this practical focuses on the validation of forensic tool. The Compromised Computer Inventory System (CCIS), created by the certification candidate, is a forensic evidence collection system that automates the acquisition of data. The CCIS forensic tool is described in detail and includes a discussion on how it benefits the analyst. Additionally, the complete code is provided for use by the forensic community.

© SANS Institute 2000-2005

# **PART ONE:**

## **Analyze an Unknown Image**

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



**ACME Security Services, Inc.**

1337 GIAC Certification Way  
Las Vegas, NV 31337

November 20, 2004

Mr. David Keen, Security Administrator  
Ballard Industries, Inc.  
6667 IRC Highway  
Las Vegas, NV 31337

Mr. Keen:

ACME Security Services, Inc. has completed our analysis of the floppy disk you provided. Per your request, a comprehensive review of the disk was performed with the specific tasks of identifying the contents and the associated potential uses. I have enclosed a comprehensive report with appendices for your review.

Thank you for the opportunity to assist Ballard Industries with this investigation. We appreciate your business. Should you need additional information, please feel free to contact me at (555) AUT-OPSY. Additionally, please contact me with instructions on returning the floppy disk and chain of custody form to you.

Regards,

James D. Perry II  
Senior Forensic Analyst

(Enclosure)

## **REPORT ON FORENSIC ANALYSIS OF FL-260404-RJL1.IMG.GZ**

### **EXECUTIVE SUMMARY:**

David Keen, Security Administrator for Ballard Industries, has retained ACME Security Services, Inc. to perform a forensic analysis of a floppy disk that was seized from an employee. Mr. Keen explained that Ballard Industries, a designer of fuel cell batteries, had recently noticed a decline in customer reorders for one of Ballard's unique lines of fuel cell batteries. An internal investigation has determined that one of Ballard's major competitors, Rift, Inc. has been receiving orders for the same fuel cell battery. Based on the recent decline in reorders, and the results of their continuing internal investigation, Ballard is fearful that the company's customer database and proprietary information has been disclosed to their competition.

Mr. Keen explained that a security guard had seized the floppy disk, mentioned above, from Robert J. Leszczynski Jr., Lead Process Control Engineer for Ballard Industries, on April 26, 2004 at approximately 4:45PM Mountain Standard Time as he was leaving the R&D labs. According to Mr. Keen, removing a floppy disk from these labs is against company policy.

Mr. Keen provided ACME Security Services, Inc. with a single floppy disk and chain of custody form that included the following information:

- TAG# fl-260404-RJL1
- 3.5 inch TDK floppy disk
- MD5: d7641eb4da871d980adbe4d371eda2ad (file: fl-260404-RLJ1.img)
- fl-260404-RJL1.img.gz

ACME Security Services' primary goal for this analysis was to provide a report to Mr. Keen regarding the contents of the floppy disk and how the contents might have been used by Mr. Leszczynski.

### **EXAMINATION DETAILS:**

Prior to performing the examination, a few critical actions were taken to ensure that the integrity of the floppy disk was maintained. The first action taken was to engage the write protection tab on the floppy disk. As described at <http://www.computerhope.com/help/floppy.htm>, this prohibits the floppy disk drive from writing any changes to the floppy disk. This was important to ensure the integrity of the floppy disk. Next, the floppy disk was inserted into the floppy drive of Jericho, one of ACME Security Services' forensic analysis stations (Jericho is a Dell Inspiron XPS Laptop running the Fedora Core 2 distribution of Linux). Once inserted, the floppy disk was mounted using the following command:



```
# mount -r -t vfat /dev/fd0 /mnt/floppy
```

This command mounts the floppy device (/dev/fd0) to the mount point of /mnt/floppy using the read-only (-r) option. This further protects the integrity of the floppy by not allowing the operating system to attempt writing to the disk (Mandia & Proise, Pg. 122).

Once the disk was protected from potential modification, forensic duplication of the floppy disk was performed. This process, described in the following paragraphs, creates an exact bit-by-bit copy, referred to as an image throughout this report, of the original floppy disk. This task was completed so that all analysis is performed on an image of the disk rather than the original disk, thus, further protecting the integrity of the original evidence and providing a scientific control. One would naturally question how an exact copy can be made of the disk and how can one objectively conclude that the image is exact. These questions will be answered in the following paragraphs.

As described earlier, the write-protected floppy disk was mounted read-only on one of ACME Security Services' forensic workstations. The built-in Unix command **dd** was then used to create the forensic image. **dd** is, according to [http://encyclopedia.thefreedictionary.com/Dd%20\(Unix\)](http://encyclopedia.thefreedictionary.com/Dd%20(Unix)), "a common Unix program whose primary purpose is the low-level copying of files." It is widely used in "computer forensics when the contents of a partition need to be preserved in a byte-exact copy." To perform the duplication the following command was used:

```
# dd if=/dev/fd0 of=/forensics/gcfa/v1_5.gz
```

This command instructs **dd** to make a bit-by-bit copy of the contents of /dev/fd0 (the protected floppy) and save the image in a file named v1\_5.gz in the /forensics/gcfa directory. This answers the question asked earlier regarding how an exact image can be obtained.

To answer the question concerning an objective method to insure the image is an exact duplication, I relied on the tool **md5sum**. Professor Ronald L. Rivest of MIT created the MD5 algorithm to take an input file of arbitrary length, process the file through an algorithm, outlined in RFC 1321, to create a 128-bit "fingerprint." As stated at <http://www.networksorcery.com/enp/data/md5.htm> "it is conjectured that it is computationally infeasible to product two [different] messages having the same" fingerprint. Because of this fact, MD5 fingerprints are used as a reliable method to insure that the contents of a file have not changed. The **md5sum** command computes an MD5 fingerprint for a file.

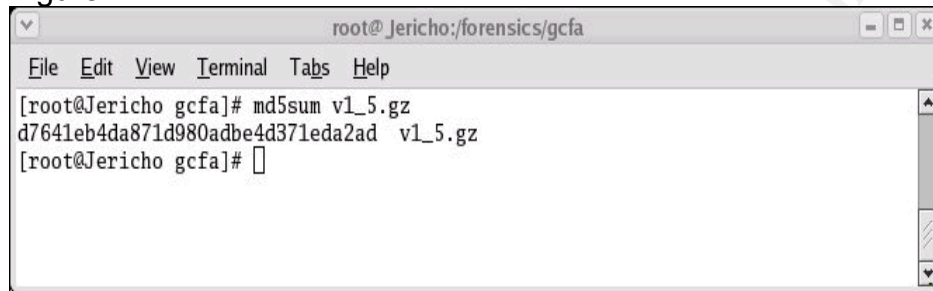
The chain-of-custody form provided to ACME Security Services by Mr. Keen included an MD5 fingerprint that was created for the floppy disk prior to

transferring the evidence to ACME.

- MD5: d7641eb4da871d980adbe4d371eda2ad (file: fl-260404-RLJ1.img)

Thus, the image, if it is an exact copy, should provide the exact same fingerprint as above when an MD5 sum is created for it. The following figure shows the results from the **md5sum** command:

Figure 1



```

root@Jericho:/forensics/gcfa
File Edit View Terminal Tabs Help
[root@Jericho gcfa]# md5sum v1_5.gz
d7641eb4da871d980adbe4d371eda2ad v1_5.gz
[root@Jericho gcfa]#

```

Figure1, above, illustrates that the MD5 fingerprint provided by Ballard Industries, on the chain-of-custody form, are identical to the fingerprint of my image, thus, I was assured that my image was an exact copy of the floppy disk.

The analysis of the image began by attempting to determine what type of computer used the floppy disk. To do this I used the Unix **file** command which performs an analysis of a file and outputs its type. The following figure shows the results on the image file:

Figure 2



```

root@Jericho:/forensics/gcfa
File Edit View Terminal Tabs Help
[root@Jericho gcfa]# file v1_5.gz
v1_5.gz: x86 boot sector, code offset 0x3c, OEM-ID "mkdosfs", root entries 224,
sectors 2872 (volumes <=32 MB) , sectors/FAT 9, serial number 0x408bed14, label
: "RJL", FAT (12 bit)
[root@Jericho gcfa]#

```

The output above provided some very useful information. First, I determined that the disk was formatted using FAT12. This indicates that the disk was *likely* formatted using a Microsoft Windows operating system. Second, I could infer that the disk was most likely formatted by Mr. Robert J. Leszczynski. This assumption is based on the disk's label "RJL" which are Mr. Leszczynski's initials. The end-user is prompted to create the label of choice when formatting a disk using Microsoft Windows.

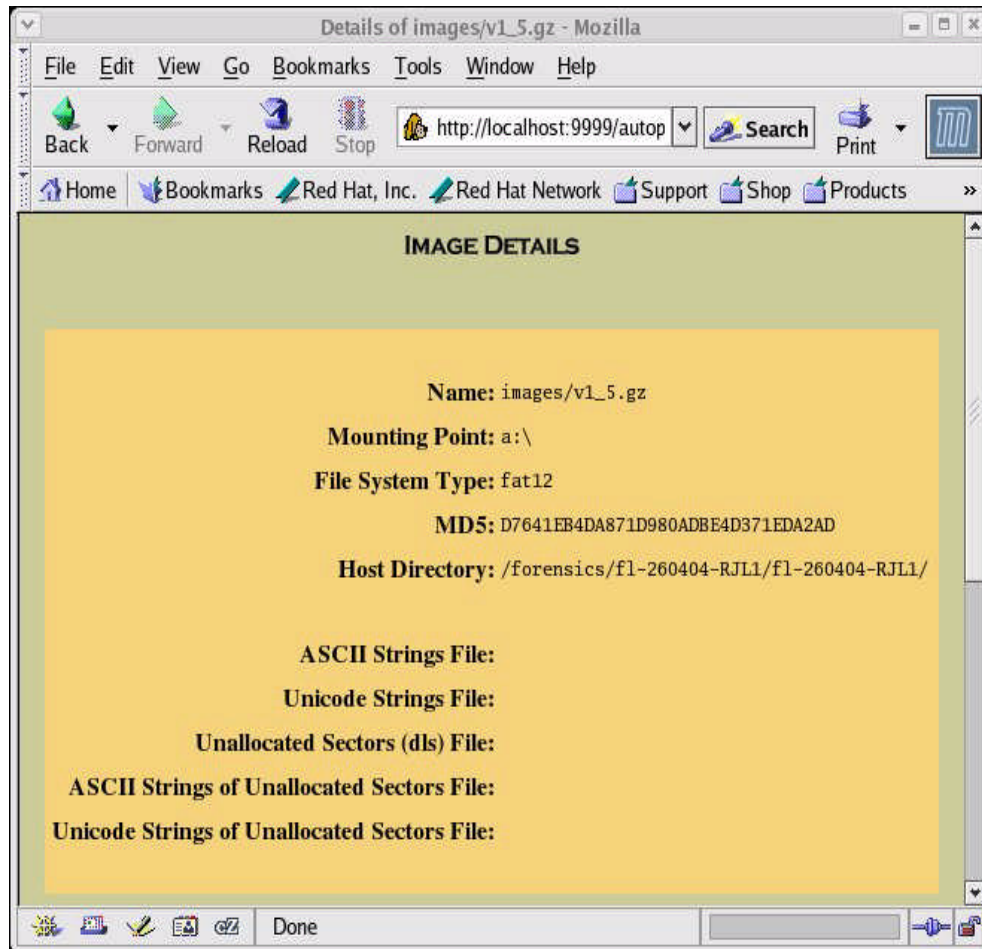
During a forensic analysis it is important to establish an investigative

foundation. To accomplish this, a few key pieces of information must be obtained. First, it is important to establish a timeline of when files that are contained within an image were last modified, last accessed, and last changed (referred to as MAC times). This information helps identify a chronological order of the activities performed on or by the device being reviewed. In this particular case, a floppy disk is being analyzed. Second, the analyst needs to identify the contents of the image. This includes identifying and recovering files that may have been deleted, modified, and/or partially overwritten. These tasks provide valuable information in establishing what is on the floppy disk and how it might have been used; both of which were identified as goals of this analysis by Mr. Keen.

My analysis used the Autopsy Forensic Browser V2.03 to establish a MAC timeline and an inventory of the image contents. Autopsy Forensic Browser, available at <http://www.sleuthkit.org/autopsy/desc.php>, is an Open Source tool built for Unix systems that automates several of the fundamental forensic analysis tasks. I began my analysis by importing the floppy image into the Autopsy Forensic Browser system. Autopsy Forensic Browser provides the ability to create a MD5 fingerprint for imported images. This provides additional assurance that the Autopsy Forensic Browser maintained the integrity of the image during the analysis. The following figure shows the MD5 fingerprint that was generated for the floppy disk image inside the Autopsy Forensic Browser:

Figure 3

© SANS Institute 2000 - 2005



The MD5 fingerprint shown above in Figure 3 matches the one provided on the chain of custody form.

Once I verified the MD5 fingerprint, Autopsy Forensic Browser was used to create a MAC timeline. The results of the timeline showed activity during February, 2001 and April, 2004. The following figures illustrate the results:

Figure 4: February 2001 Timeline Summary

© SANS Institute

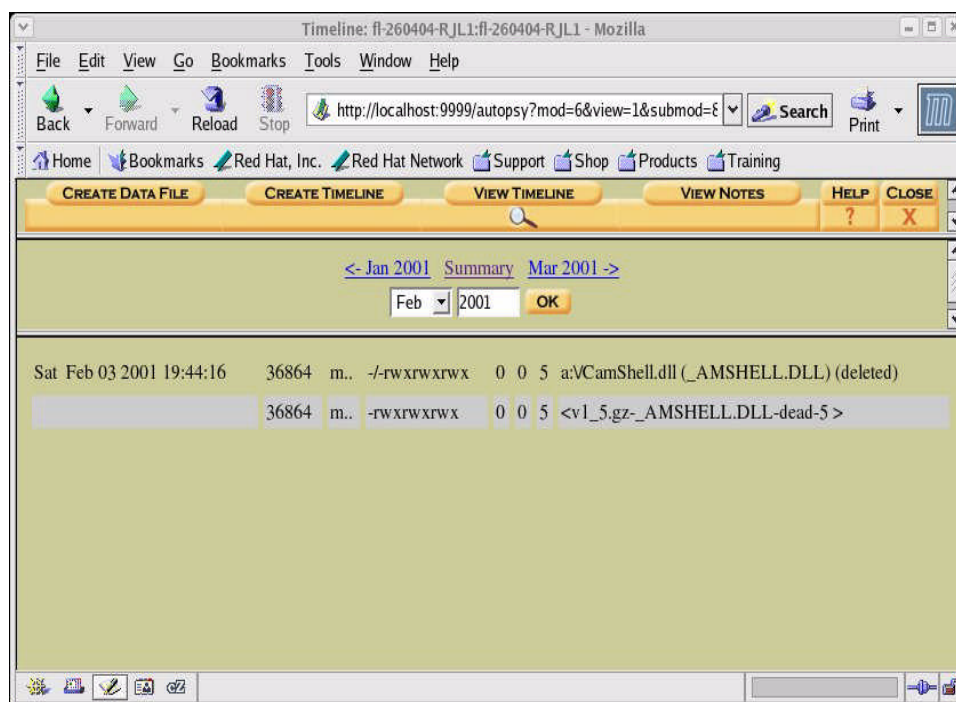


Figure 4, above, indicates that a file named `a:\CamShell.dll` was modified on Saturday, February 3, 2001 at 19:44:16. After the filename I noticed in parenthesis that the file was later deleted. Based on experience, recovering and analyzing deleted files usually provides an excellent source of possible leads. In this particular case, the file has an extension of `*.dll`. DLL stands for dynamic link library which is a “file containing a collection of Windows functions designed to perform a specific class of operations....Functions within DLLs are called (invoked) by applications as necessary to perform the desired operation” (<http://www.google.com/search?hl=en&lr=&oi=defmore&q=define:DLL>). This provided the first lead to investigate. Making note of the file name, I next looked at the timeline for April 2004 which is shown below in Figure 5.

Figure 5, below, indicated that a file name `a:\_ndex.htm` was modified on Friday, April 23, 2004 at 10:53:56. Again, I noticed that the file is later deleted. This time the file had an extension of `*.htm` which is the file extension for Hypertext Markup Language, HTML for short. HTML is “the coded format language used for creating hypertext documents on the World Wide Web and controlling how Web pages appear” (<http://www.getnetwise.org/glossary.php#H>). This provided another lead to investigate.

The April 2004 timeline in Figure 5, below, also lists several files with the `*.doc` file extension. The `*.DOC` extension is commonly known as that of Microsoft Word, a popular word processing software application. In general, Microsoft Word documents are normally small in size. With that information in mind, two particularly large files immediately caught my attention: `a:\Remote_Access_Policy.doc` and `a:\Password_Policy.doc`. These files were



noted for further investigation.

Figure 5: April 2004 Timeline Summary

Date	Time	Size	Type	Permissions	File Path
Thu Apr 22 2004	16:31:06	32256	m..	-rwxrwxrwx	0 0 13 a:\Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
		33423	m..	-rwxrwxrwx	0 0 17 a:\Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
Fri Apr 23 2004	10:53:56	727	m..	-rwxrwxrwx	0 0 28 a:\V_index.htm (deleted)
		727	m..	-rwxrwxrwx	0 0 28 <v1_5_gz-_index.htm-dead-28 >
Fri Apr 23 2004	11:54:32	215895	m..	-rwxrwxrwx	0 0 23 a:\Remote_Access_Policy.doc (REMOTE~1.DOC)
Fri Apr 23 2004	11:55:26	307935	m..	-rwxrwxrwx	0 0 20 a:\Password_Policy.doc (PASSWO~1.DOC)
Fri Apr 23 2004	14:10:50	22528	m..	-rwxrwxrwx	0 0 27 a:\Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
Fri Apr 23 2004	14:11:10	42496	m..	-rwxrwxrwx	0 0 9 a:\Information_Sensitivity_Policy.doc (INFORM~1.DOC)
Sun Apr 25 2004	00:00:00	0	.a.	-rwxrwxrwx	0 0 3 a:\VRJL (Volume Label Entry)
Sun Apr 25 2004	00:53:40	0	m.c	-rwxrwxrwx	0 0 3 a:\VRJL (Volume Label Entry)
Mon Apr 26 2004	00:00:00	215895	.a.	-rwxrwxrwx	0 0 23 a:\Remote_Access_Policy.doc (REMOTE~1.DOC)
		727	.a.	-rwxrwxrwx	0 0 28 <v1_5_gz-_index.htm-dead-28 >
		307935	.a.	-rwxrwxrwx	0 0 20 a:\Password_Policy.doc (PASSWO~1.DOC)
		42496	.a.	-rwxrwxrwx	0 0 9 a:\Information_Sensitivity_Policy.doc (INFORM~1.DOC)
		36864	.a.	-rwxrwxrwx	0 0 5 <v1_5_gz-_AMSHLL.DLL-dead-5 >
		727	.a.	-rwxrwxrwx	0 0 28 a:\V_index.htm (deleted)
		32256	.a.	-rwxrwxrwx	0 0 13 a:\Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
		33423	.a.	-rwxrwxrwx	0 0 17 a:\Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
		36864	.a.	-rwxrwxrwx	0 0 5 a:\CamShell.dll (_AMSHLL.DLL) (deleted)
		22528	.a.	-rwxrwxrwx	0 0 27 a:\Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
Mon Apr 26 2004	09:46:18	36864	.c	-rwxrwxrwx	0 0 5 a:\CamShell.dll (_AMSHLL.DLL) (deleted)
		36864	.c	-rwxrwxrwx	0 0 5 <v1_5_gz-_AMSHLL.DLL-dead-5 >
Mon Apr 26 2004	09:46:20	42496	.c	-rwxrwxrwx	0 0 9 a:\Information_Sensitivity_Policy.doc (INFORM~1.DOC)
Mon Apr 26 2004	09:46:22	32256	.c	-rwxrwxrwx	0 0 13 a:\Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
Mon Apr 26 2004	09:46:24	33423	.c	-rwxrwxrwx	0 0 17 a:\Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
Mon Apr 26 2004	09:46:26	307935	.c	-rwxrwxrwx	0 0 20 a:\Password_Policy.doc (PASSWO~1.DOC)
Mon Apr 26 2004	09:46:36	215895	.c	-rwxrwxrwx	0 0 23 a:\Remote_Access_Policy.doc (REMOTE~1.DOC)
Mon Apr 26 2004	09:46:44	22528	.c	-rwxrwxrwx	0 0 27 a:\Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
Mon Apr 26 2004	09:47:36	727	.c	-rwxrwxrwx	0 0 28 a:\V_index.htm (deleted)
		727	.c	-rwxrwxrwx	0 0 28 <v1_5_gz-_index.htm-dead-28 >

With the timeline established and a few items identified to further investigate, the next action performed was a review of the contents of the floppy disk. Figure 6, below, shows the contents of the floppy disk using the Autopsy Forensic Browser.

Figure 6

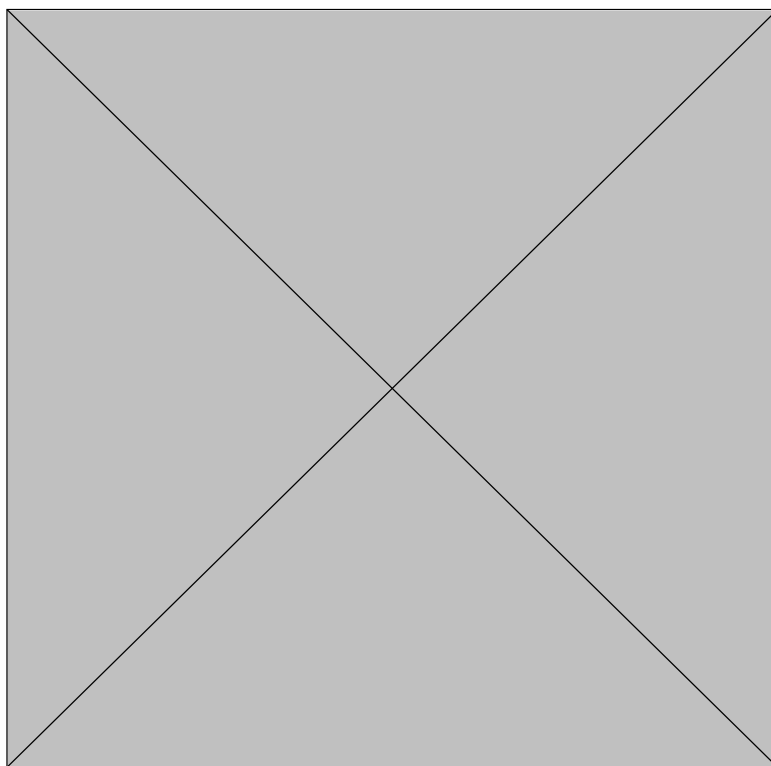


Figure 6 outlined a total of eight files on the floppy disk. Two of which have been deleted, and the rest are Microsoft Word documents.

Based on the analysis of the timeline and disk contents, I identified two deleted files and two Microsoft Word documents that initially required further

Autopsy Forensic Browser provides a strings reporting mechanism. The reporting tool was used to generate the strings reports that are found in the appendix of this document.

Appendix A-1 and Figure 7, below, provides information that the file a:\\_ndex.htm contained HTML code for a simple webpage. Within the code a few references to Ballard Industries are found providing a link to the purpose of the HTML code. Otherwise, the file didn't provide much additional information.

Figure 7: a:\\_ndex.htm **strings** command information

```
-----
                                CONTENT

<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDED"ED">
<center>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"

codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#vers
ion=6,0,0,0"
  WIDTH="800" HEIGHT="600" id="ballard" ALIGN="">
  <PARAM NAME=movie VALUE="ballard.swf"> <PARAM NAME=quality VALUE=high> <PARAM
NAME=bgcolor VALUE=#CCCCC> <EMBED src="ballard.swf" quality=high bgcolor=#CCCCC
WIDTH="800" HEIGHT="600" NAME="ballard" ALIGN=""
  TYPE="application/x-shockwave-flash"
  PLUGINSOURCE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</center>
</BODY>
</HTML>

-----
```

The file a:\CamShell.dll was analyzed next. Appendix A-2 contains the complete strings report and Figure 8, below, shows a key portion of the report.

Figure 8: a:\CamShell.dll **strings** command information

```
-----
                                CONTENT

<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDED"ED">
<center>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"

codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#vers
ion=6,0,0,0"
  WIDTH="800" HEIGHT="600" id="ballard" ALIGN="">
  <PARAM NAME=movie VALUE="ballard.swf"> <PARAM NAME=quality VALUE=high> <PARAM
NAME=bgcolor VALUE=#CCCCC> <EMBED src="ballard.swf" quality=high bgcolor=#CCCCC
```



```

WIDTH="800" HEIGHT="600" NAME="ballard" ALIGN=""
  TYPE="application/x-shockwave-flash"
  PLUGINSOURCE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</center>
</BODY>
</HTML>
11\SheCamouflageShell
ShellExt
VB5!
CamShell
BitmapShellMenu
CamouflageShell
CamouflageShell
Shell_Declares
Shell_Functions
-----

```

Figures 7 & 8 look identical for the first dozen lines or so. Since a:\CamShell.dll is a dynamic link library, it is odd to discover HTML code embedded within, especially since it was the exact same code as in a:\\_ndex.htm. The remainder of the file appears to be a program DLL with an unknown purpose. Based on experience, seeing two files in an image that have been deleted as these two were, and finding the one file contained within the other usually means that someone/something was trying to masquerade the a:\CamShell.dll file as a HTML webpage. Continuing with the analysis of a:\CamShell.dll, the strings report provided several keywords that could be used to potentially determine the purpose of the DLL. Figure 9, below, provides a list of the keywords identified.

Figure 9

```

-----
A:\CamShell.dll KEYWORDS

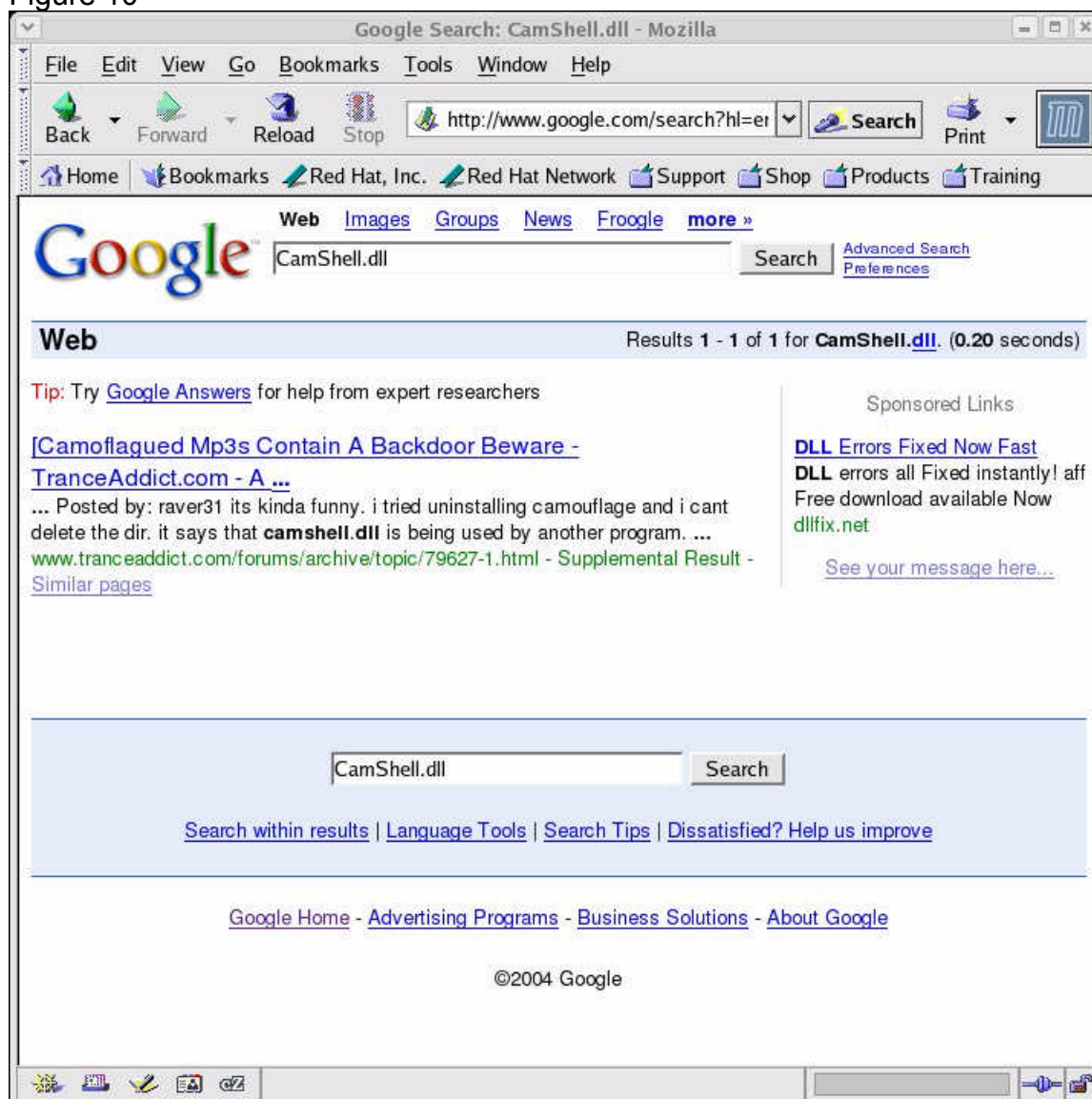
Camouflage
CamShell
CamouflageShell
C:\My Documents\VB Programs\Camouflage\Shell\IctxMenu.tlb
CamShell.dll
-----

```

Armed with an initial list of keywords, I then attempted to define the purpose of the a:\CamShell.dll file. With no previous experience or encounters with a file named CamShell.dll, the keywords in Figure 9, above, were used to perform searches using [www.google.com](http://www.google.com)'s comprehensive search engine. The goal was to find out information about the file in question. Figure 10, below, shows the results of a search for CamShell.dll.

The Google search returned with only one result. The synopsis from the one query hit contained some interesting information. First, the words "BackDoor" being using in association with CamShell.dll was curious. Next, the title of the result mentions Camoflagued MP3s.

Figure 10



The website that was returned in the Google search, as shown in Figure 10, above, proved to be a critical discovery in the analysis of the CamShell.dll file. Figures 11 & 12, below, shows excerpts from the webpage.

The website that was returned by the Google search, as illustrated in Figures 11 & 12, below, appears to be a web based discussion forum where individuals were discussing the possibility of a program called Camouflage potentially installing a backdoor on their systems. The particular post in Figure 11 describes the purpose of the Camouflage software. In short, it appears that the Camouflage software is a steganography tool and CamShell.dll is a component of the Camouflage software. Steganography is defined as the practice of hiding information by embedding messages within other seemingly

harmless messages (<http://www.webopedia.com/TERM/S/steganography.html>).”

Figure 11

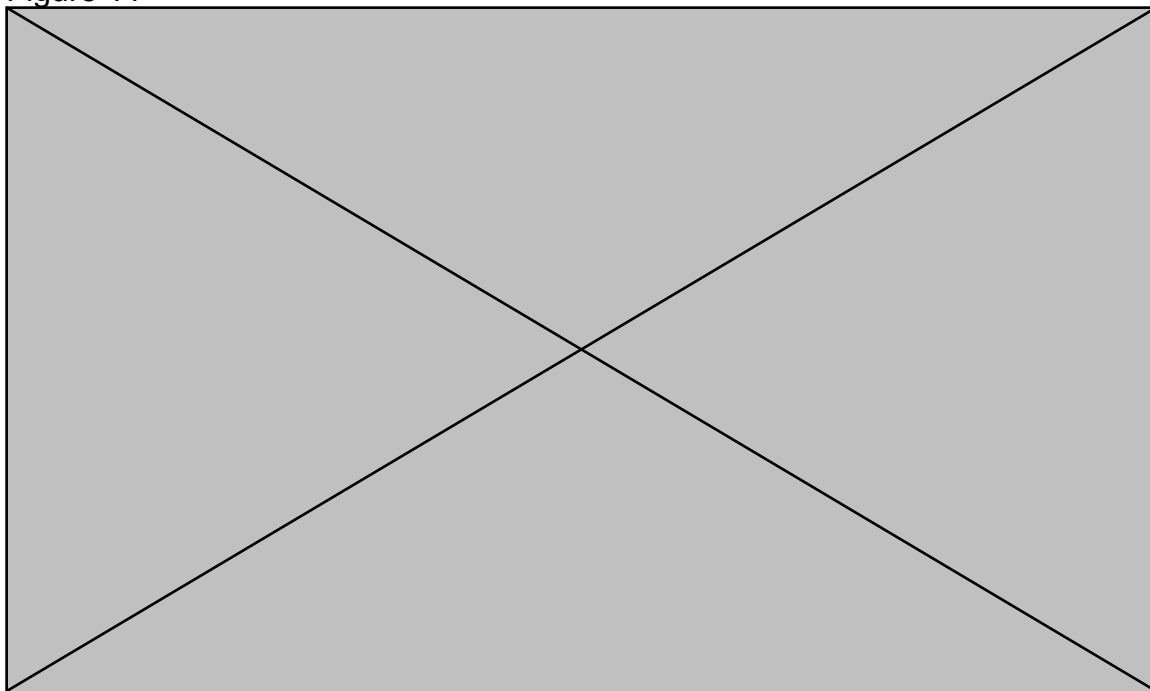


Figure 12, below, provides the specific name, called Camouflage, of the software that contains the CamShell.dll. Since camouflage was found in my strings analysis of the CamShell.dll, I next performed a Google search using “camouflage software” as the keywords for the query. The results are shown in Figure 13.

Figure 12

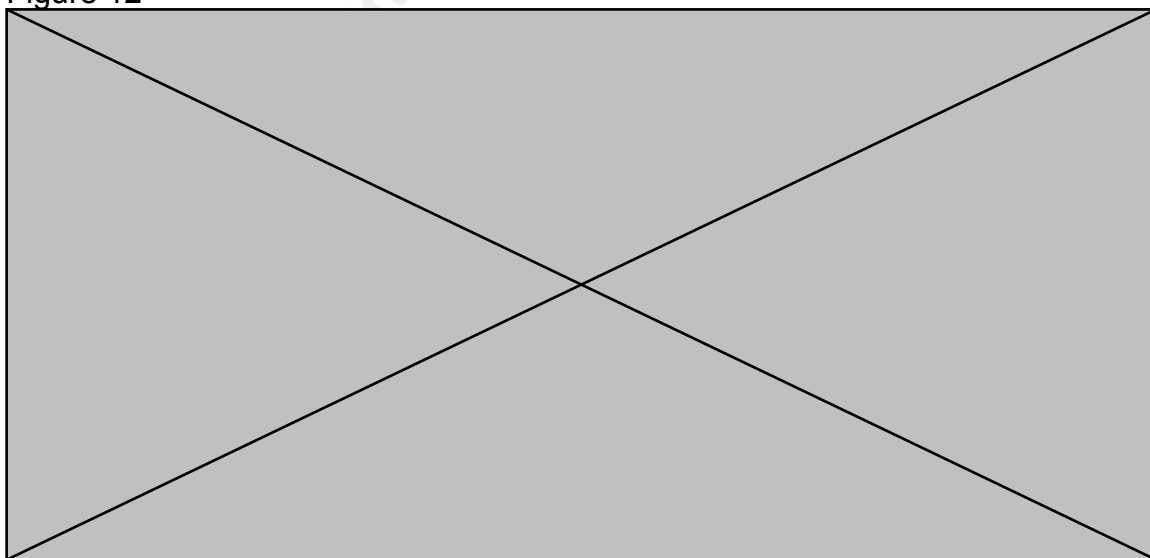
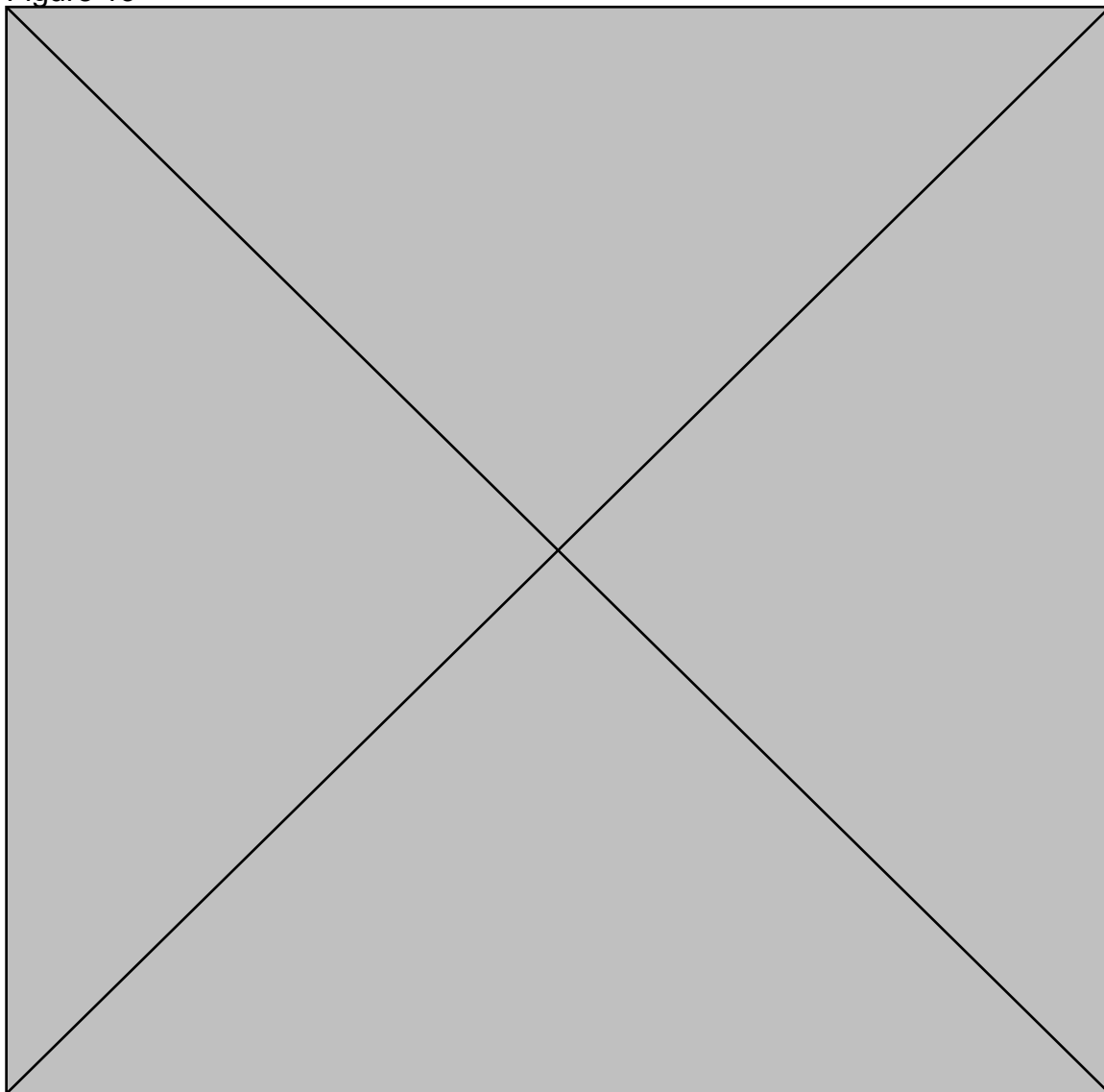


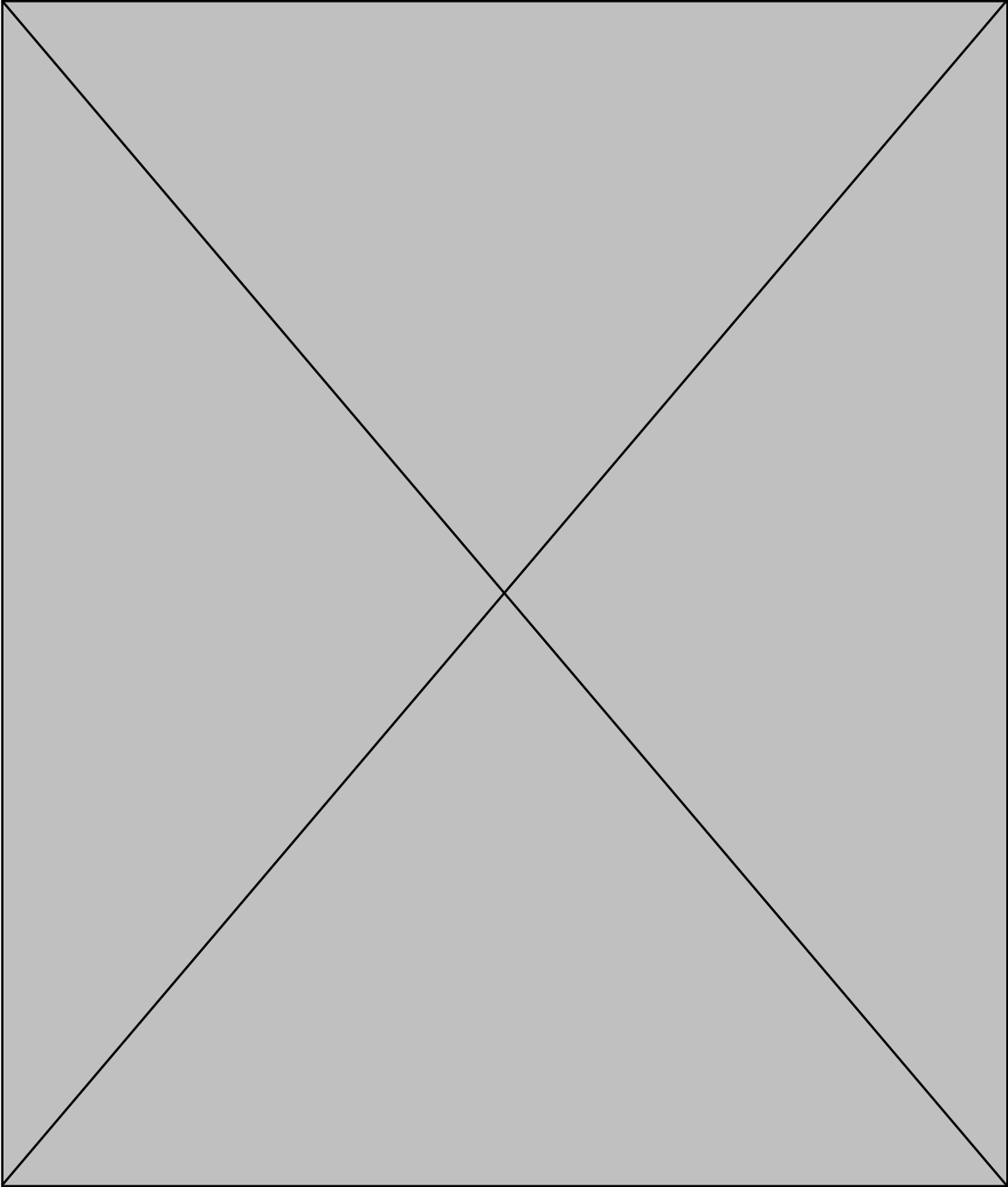
Figure 13



The third link shown in Figure 13 was used to successfully confirm the purpose of the Camouflage software as described in the user forum posts shown in Figures 11 & 12. Figure 14, below, shows the website with the information regarding the functionality of Camouflage. At this point I had a good idea of how the CamShell.dll might be used; however, further analysis of the Camouflage software and a direct correlation between the software and our DLL needed to be made. To continue my analysis, I downloaded the Camouflage V1.2.1 software from the site shown in Figure 14.

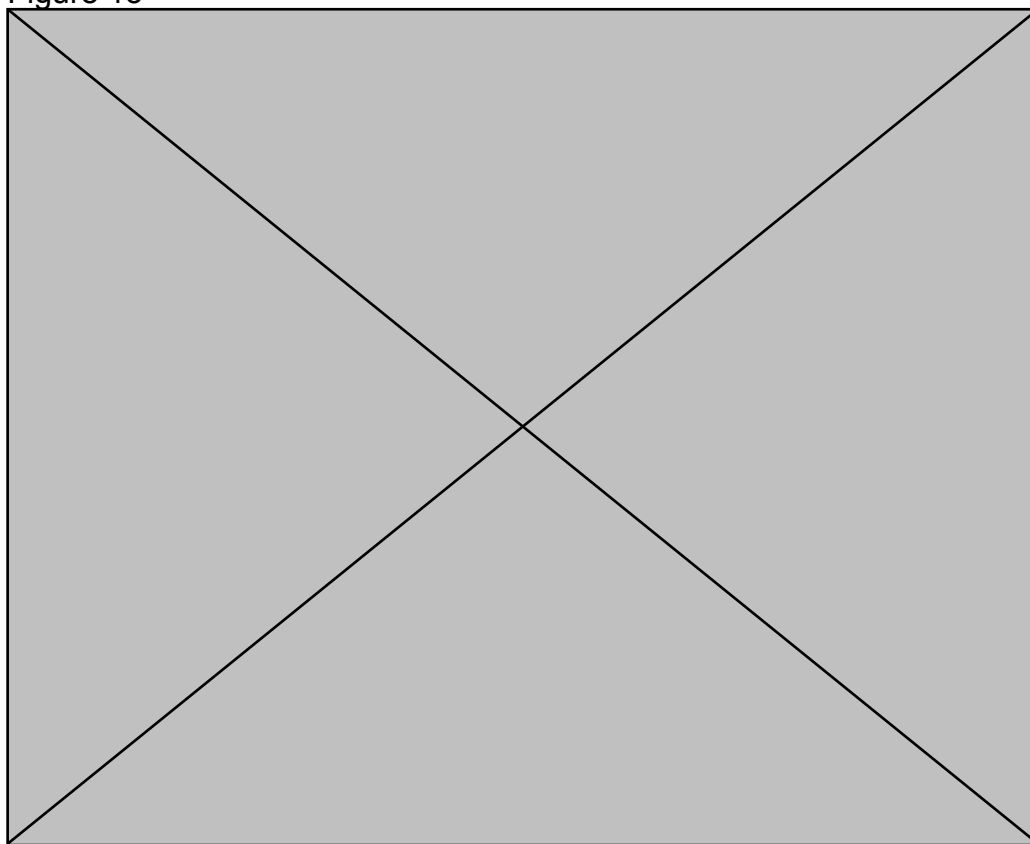
Using a test system running Microsoft Windows XP Professional, I installed the Camouflage software. Upon completion of the installation, I inspected the C:\Program Files\Camouflage directory, which is where the software was installed by default. Figure 15, below, shows the contents of the

directory.  
Figure 14



Immediately, the presence of a file named CamShell.dll stood out. Given that the floppy disk had a deleted file with the same name, knowing the background information regarding the recent developments at Ballard Industries, understanding the function of the Camouflage software, and having two unusually large Microsoft Word documents left to investigate, the potential for hidden files within the image was probable. Nevertheless, to finish the analysis of the CamShell.dll file, a direct connection needed to be established.

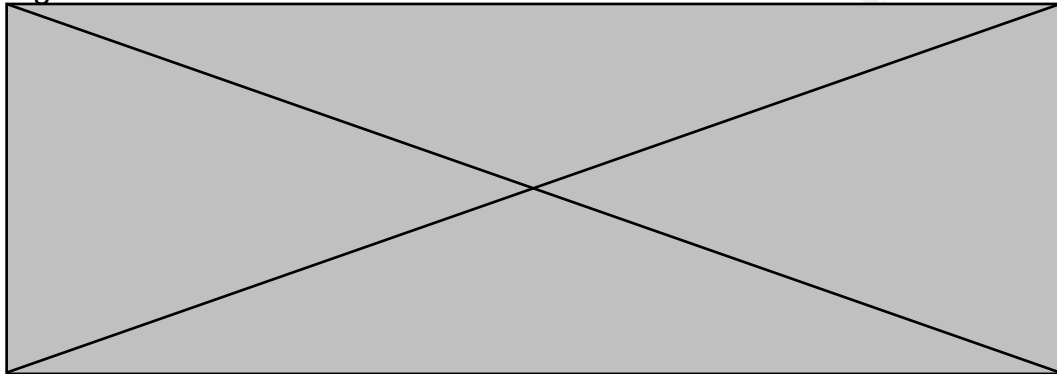
Figure 15



The first direct correlation, as illustrated in Figures 4 and 15, is that both CamShell.dll files have the exact same file modification timestamp. Thus, the probability that these files are one in the same is high. To establish further proof, I compared the CamShell.dll file recovered from the floppy disk image to the one installed by the Camouflage software. Figures 8, above, illustrated the fact that the recovered CamShell.dll file had the HTML code from the \_ndex.htm file overwritten where the file header normally would be. As such, I used **khaxedit**, a hexadecimal editor for Unix, to identify the size of the \_ndex.htm file. I determined, based on looking at the \_ndex.htm file in **khaxedit**, that the end of the file is located at the hexadecimal offset of 0000:02d6. This information allowed me to get the exact location of the HTML code within the recovered CamShell.dll file and remove it from the file. To achieve this goal, I then opened the recovered CamShell.dll file using **khaxedit** and, using the hexadecimal offset determined from the \_ndex.htm file, deleted everything in the recovered CamShell.dll files prior to the 0000:02d6 offset, thus effectively removing the HTML code. My next goal was to copy the same amount of data, deleted in the previous step, from the new CamShell.dll file installed by Camouflage. To achieve this task, I opened the new CamShell.dll and copied all of the hexadecimal values from the beginning of the file to the 0000:02d6 offset and inserted it into the beginning of the recovered CamShell.dll. Using the same forensic theory as earlier, if the 2 files are identical, they should then have the

same MD5 fingerprint. Figure 16, below, confirms that MD5 fingerprints did indeed match. Thus, considering the identical last modification date timestamp and the MD5 fingerprint, I was able to prove that the two CamShell.dll files are in fact one in the same. Furthermore, referring back to the timeline illustrated in Figure 5, the program was accessed last on April 26, 2004, which is the same day the disk was seized.

Figure 16



My investigation so far has linked the floppy disk to Mr. Leszczynski using the floppy disk volume label, has recovered two deleted files, and has successfully established the origin and potential functionality of the previously unknown program DLL. Given that the program DLL CamShell.dll is part of a steganography program named Camouflage, and that there are two abnormally large Microsoft Word documents, my investigation began to focus on analyzing these documents.

As was the case in earlier steps of the analysis, I used the Unix command strings to perform an evaluation of the files a:\Remote\_Access\_Policy.doc and a:\Password\_Policy.doc. Figure 17, below, provides a critical portion of the a:\Remote\_Access\_Policy.doc Strings Report.

Figure 17: a:\Remote\_Access\_Policy.doc Strings Report

```
-----
                                CONTENT

Normal.d
Microsoft Word 10.0
Ballard
Cisco Systems, Inc.
Remote Access Policy
Title
Microsoft Word Document
MSWordDoc
Word.Document.8
Remote Access Policy
Normal.dot
Microsoft Word 10.0
Ballard
O6pQ
gW^b!
```

```

)AqXSh]
" `LJ
\N>      )x)
NNVs
ZKfqqjj
(JS$?
nQh`n
;pt]`

```

---

Previous experience in investigating the format of Microsoft Word documents has taught me that at the end of all Microsoft Word documents there is registration information that was provided during the installation of the software. As expected, the files analyzed contained the normal information such as company name (Ballard), the application version of Microsoft Word (Microsoft Word 10.0) and other miscellaneous information. However, after the last Ballard and for several pages of the report, a significant amount of scrambled (perhaps compressed or encrypted data) is present. Based on the information already established, this indicated that it was very likely that the file contains an embedded file of some sort. Given this information, I then performed a Strings Report on the other five remaining Microsoft Word Documents. The result of this analysis identified the presence of additional data contained within three of the Word documents contained on the floppy disk. The files in question are the original two I noted from the timeline analysis (a:\Remote\_Access\_Policy.doc and a:\Password\_Policy.doc), and one additional file (a:\Internal\_Lab\_Security\_Policy.doc).

Being unfamiliar with the Camouflage software, I then revisited some of the links Google provided earlier in Figure 13. The last link illustrated in Figure 13 was to the link [www.sans.org/rr/papers/20/762.pdf](http://www.sans.org/rr/papers/20/762.pdf). This link was to a document entitled Stegonagraphy: The Ease of Camouflage written by John Bartlett in March of 2002. Contained with the document was a basic step-by-step process for “camouflaging” and “uncamouflaging” files with the Camouflage software. Additionally, valuable information was documented regarding how to determine if the Camouflage software has ever been installed and what, if any, files have been “camouflaged” or “uncamouflaged” using the software. For now I was only interested in learning how to use the software. Nevertheless, the information on detection will be revisited later.

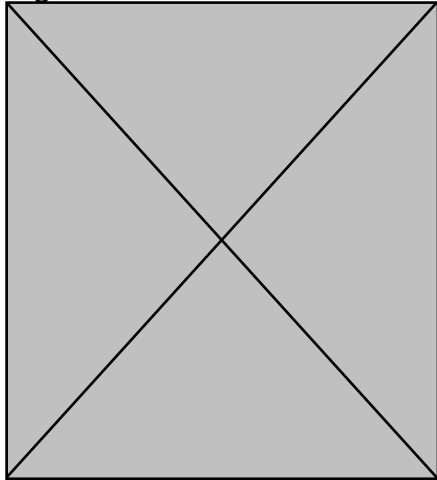
In short, the Camouflage software’s function is to take a file, of ANY type, and embed another file of ANY type within it. To do this, the software simply adds the file to be hidden at the end of the normal file. As such, the application that uses the normal file is unaware of the hidden files existence as the file headers and format remain in tact. The only indication of the file being altered is that the size of the file is reported as the true size which includes the hidden file.

Using the information learned from Mr. Bartlett’s timely paper, I attempted to “uncamouflage” the a:\Remote\_Access\_Policy.doc file. The process was as



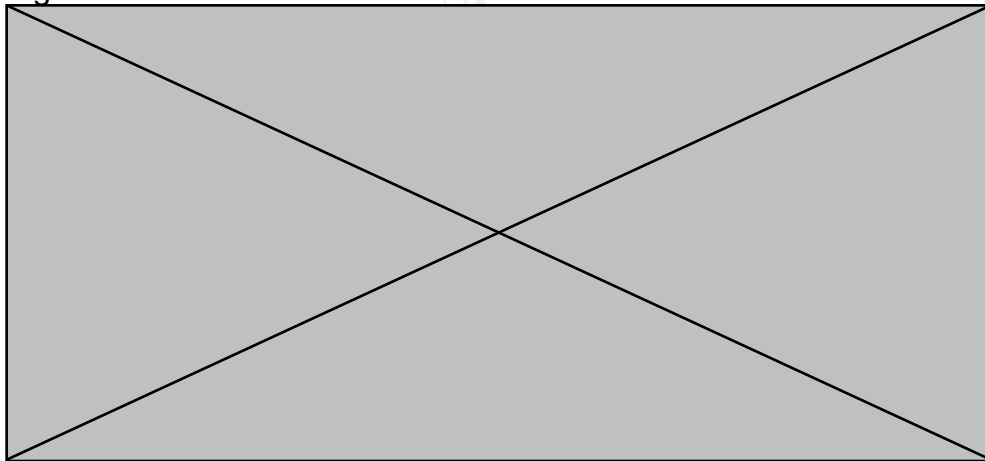
easy as right clicking on the desired file from within Windows Explorer. This action resulted in the menu appearing that is depicted in Figure 18 below.

Figure 18



I then selected the “Uncamouflage” option and was presented with the option window shown below in Figure 19.

Figure 19



Having not found any indication of a password thus far, I attempted to guess the password using best guesses from information surrounding the investigation and other commonly used passwords, including a blank password. Figure 20 shows the response for an invalid password and Figure 21 shows the error message received when attempting a blank password.

Figure 20

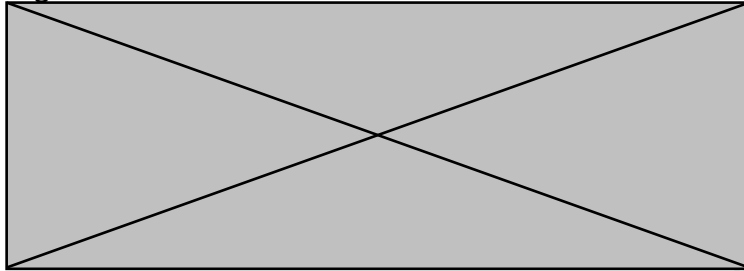
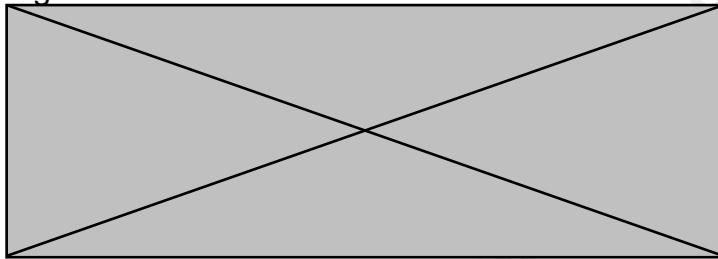
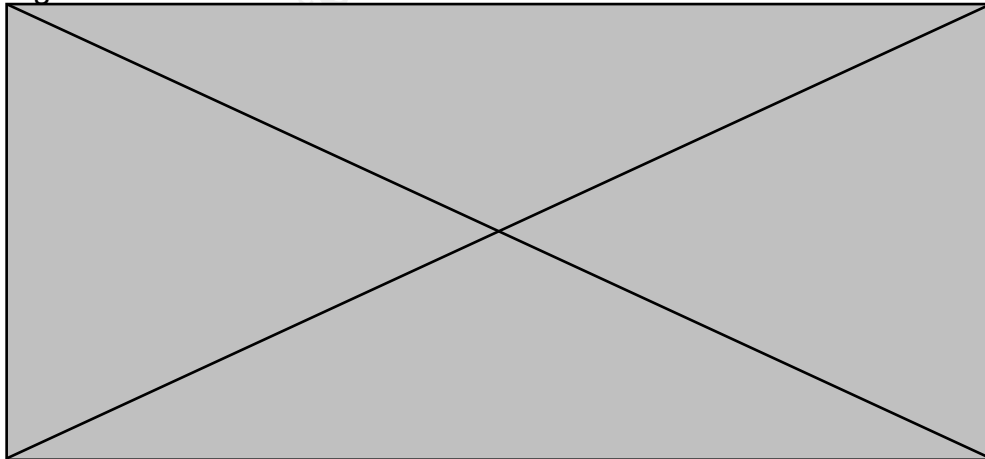


Figure 21



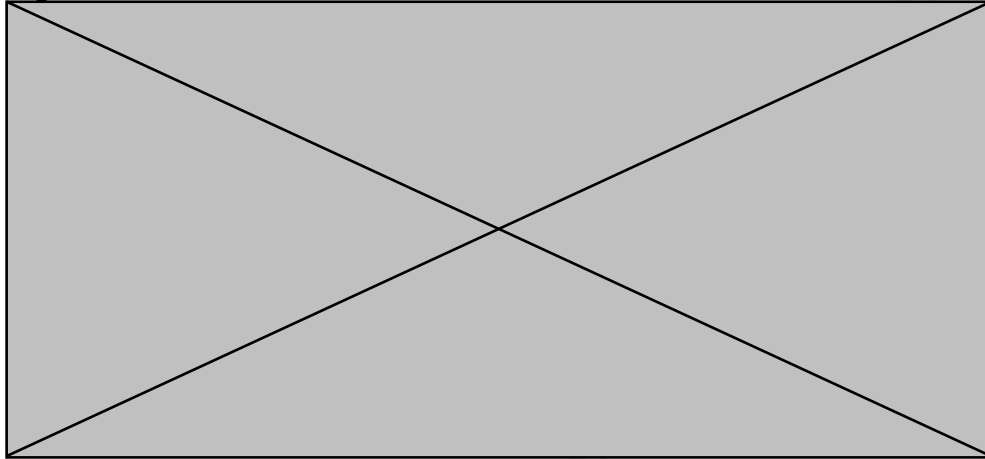
After several unsuccessful attempts at guessing the password for the a:\Remote\_Access\_Policy.doc, I attempted the same approach on the other two files. When I used a blank password on the a:\Internal\_Lab\_Security\_Policy.doc the window in Figure 22 appeared.

Figure 22



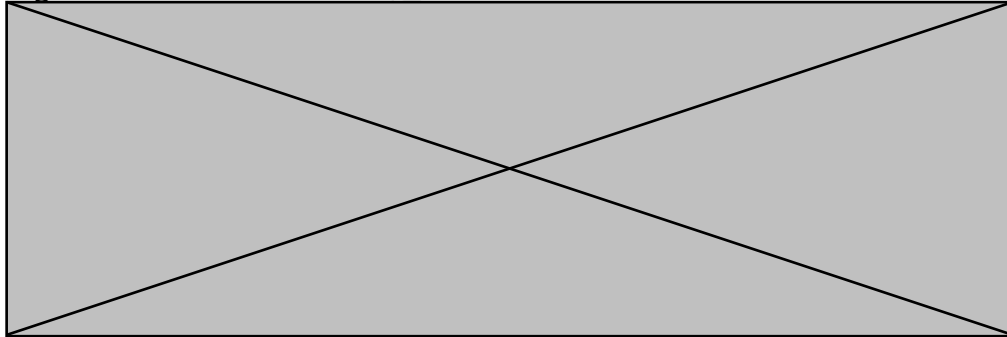
The window indicated that the file Opportunity.txt had been “Camouflaged” into the file Internal\_Lab\_Security\_Policy.doc. I then selected the Opportunity.txt file and clicked next. The window shown in Figure 23 then appeared.

Figure 23



I selected the appropriate directory to extract the file to and completed the task by selecting the “Finish” button. Figure 24 show the contents of the recovered Opportunity.txt file.

Figure 24



The Opportunity.txt file speaks for itself, however, it alludes to the “latest schematics” being available and provides the clue “First Name.” Additionally it indicates that this is not the first occurrence of releasing confidential information as it states “more information” is available for a price. Being that I had already tried the password Robert in various configurations, I inferred that it may have to do with the first name of the file. I then retried entering the passwords for the a:\Remote\_Access\_Policy.doc and a:\Password\_Policy.doc documents using the “First Name” of the file. Figures 25 and 26 illustrate that my deduction was correct.

Figure 25

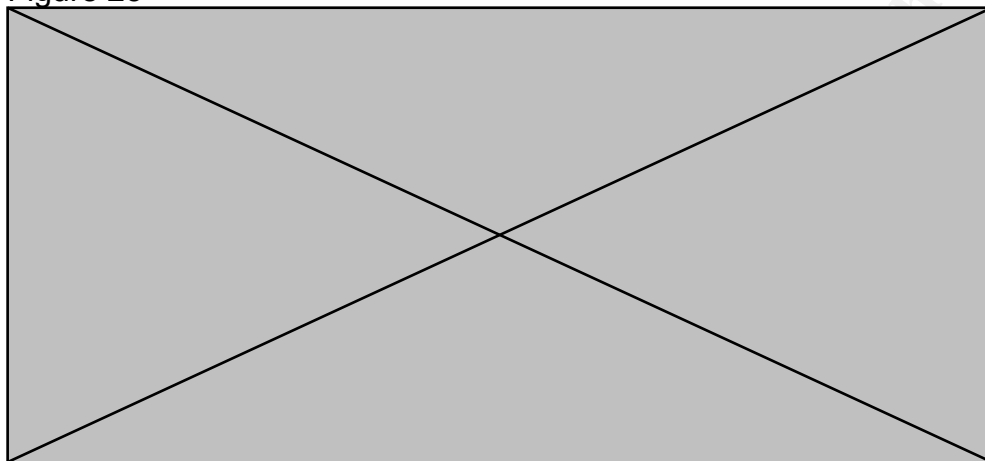
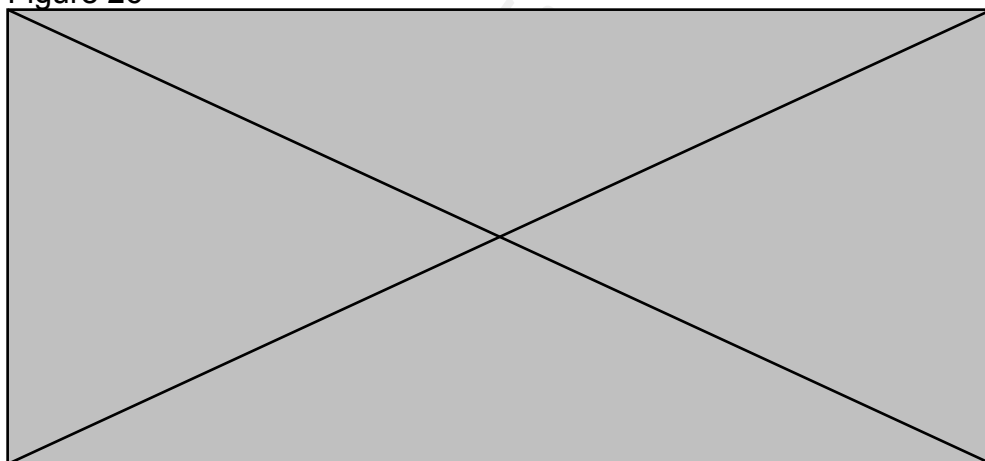


Figure 26



While it was obvious that I had discovered the intent of the files and the capability of the CamShell.dll, one additional confirmation was identified. Earlier I reported that the recovered CamShell.dll and the new one installed by Camouflage were identical using MD5 fingerprints and that the modified timestamps were also identical. Figures 25 and 26, above, further strengthen that conclusion in that the Camouflage program indicates that the files were “created with Camouflage v1.2.1”. Thus, another method of proof that I have matching versions of software has been determined.

Figures 25 and 26 indicate the presence of four “camouflaged” files. Using the same process as before, I then “uncamouflaged” the files for further review. Figures 27 through 30 illustrate three proprietary drawings and a portion of a customer database that were “camouflaged” into Microsoft Word

documents.

Figure 27

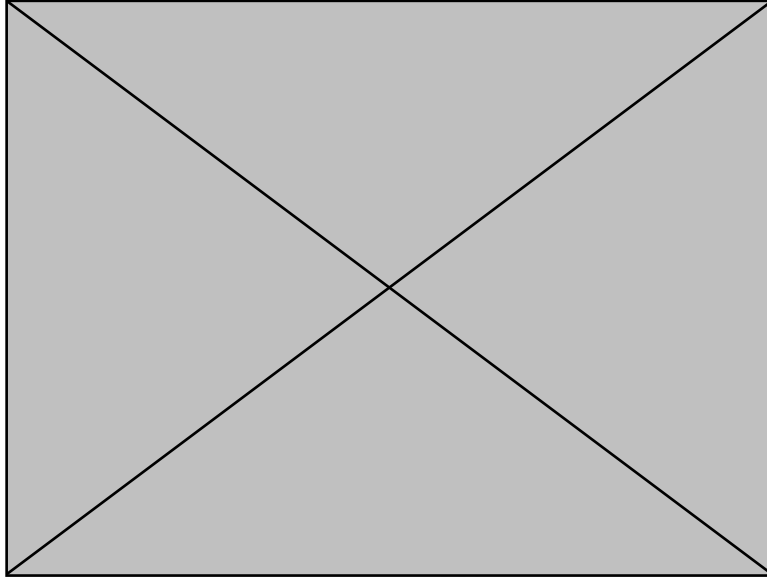


Figure 28

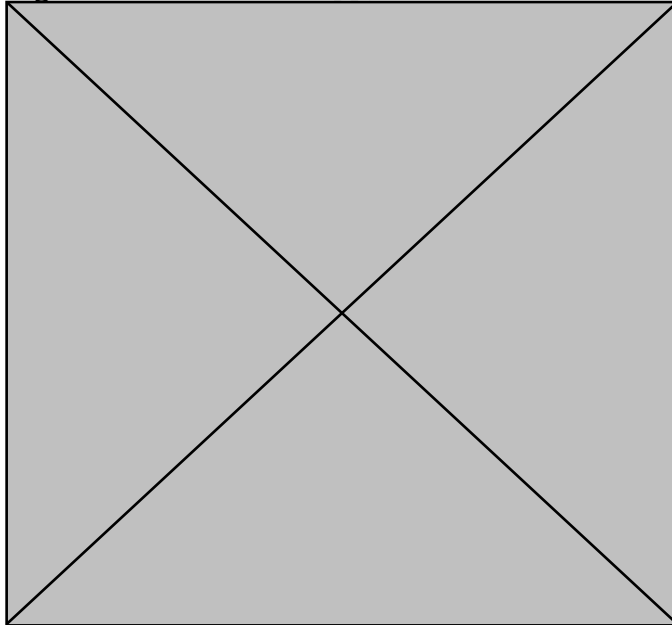


Figure 29

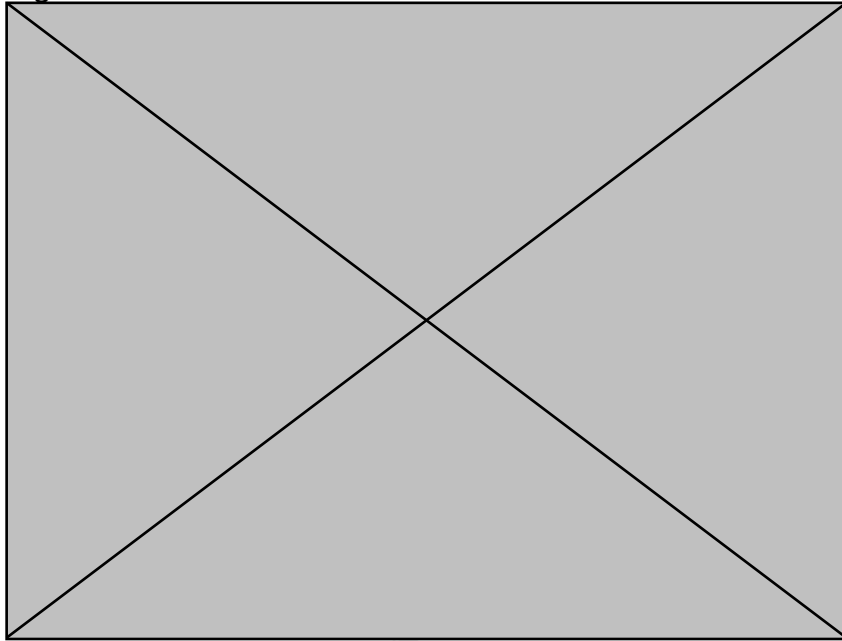
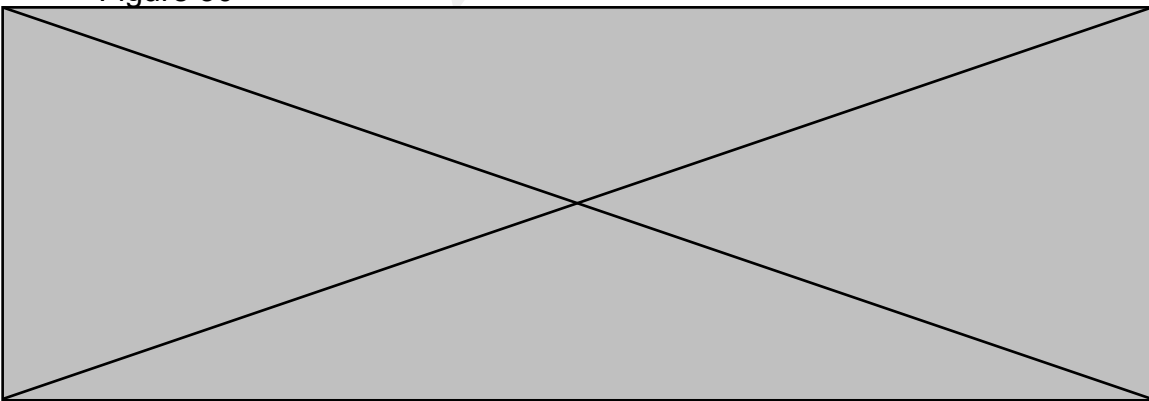


Figure 30



#### **EXAMINATION CONCLUSION:**

ACME Security Services, Inc. performed a comprehensive forensic analysis of a floppy disk for Ballard Industries. The intent of the analysis was to determine the contents of the floppy disk and establish how it may have been used. The following bullet points summarize the findings:

- The floppy disk was formatted on a Windows operating system and had a disk volume label of RJL. The same initials as the individual (Robert J. Leszczynski) the disk was seized from.
- Two deleted files on the floppy disk were recovered. The CamShell.dll file was determined to be a dynamic link library for an application called Camouflage V1.2.1 that is used to hide a file within another file using a technique called steganography.
- Three files were confirmed to have hidden files embedded within. The files recovered included two proprietary Ballard fuel cell design schematics, one proprietary Ballard fuel cell design document, a portion of the Ballard customer database, and a text document soliciting 5 million dollars in exchange for proprietary Ballard information signed by Mr. Robert J. Leszczynski.

Based on these findings, it appears that Mr. Robert J. Leszczynski has, in the past, successfully obtained and disclosed proprietary Ballard Industries information. Considering the contents and the associated changed timestamps, it appears that Mr. Robert J. Leszczynski was intending to disclose additional information, including drawings and portions of a customer database to the same entity as before. While it is clear that Mr. Robert J. Leszczynski was successful at removing such information from Ballard Industries in the past, based on the information provided by Mr. Keen regarding that date and time the disk was seized, it is unlikely the information on this disk was successfully disclosed using the floppy disk. Figure 5 illustrates that the three files containing the proprietary information had last access timestamps of Monday, April 26, 2004 at 00:00:00. This indicates that the files were not accessed on the floppy disk once they were saved there. While this eliminates the likelihood of the information being disclosed via the floppy disk, it doesn't preclude the possibility that the information was transmitted via some other means.

As mentioned earlier, John Bartlett has written a comprehensive document entitled "Steganography: The Ease of Camouflage." This document, provided at [www.sans.org/rr/papers/20/762.pdf](http://www.sans.org/rr/papers/20/762.pdf), provides a basic step-by-step tutorial on how to use the Camouflage software that was discovered during this analysis. In addition, the document provides specific information on how to determine if the Camouflage software is or has been installed on a computer system. Specifically, Mr. Bartlett reports that the Camouflage software, even if uninstalled, keeps a record of every file that has been "camouflaged" and "uncamouflaged" within the Windows registry. I would suggest that a comprehensive analysis be performed on all of the systems that Mr. Leszczynski has had access to. The analysis should focus on looking for the presence of the Camouflage software and evaluating the registries of those machines to develop a comprehensive list of the information that may have been disclosed. Additional analysis may help to determine if the information may have been transmitted via a different mechanism.

In addition, considering the magnitude of the possible damages caused to Ballard Industries, law enforcement should be engaged when appropriate, and legal action could certainly be pursued. While ACME Security Services Inc. can not provide legal advice, it is possible that the following laws, in addition to others, may have been broken:

- The Federal Computer Fraud & Abuse Act  
18 U.S.C. §1030(a)(2) & (c)(2)(B)(i)-(ii)
- Criminal Copyright Laws Violations  
18 U.S.C. 2319 & 17 U.S.C. 506a
- Criminal Trade Secrets Violations  
18 U.S.C. 1831, 1832

Furthermore, in my experience, these laws typically require a loss of \$5,000 or more to meet the minimum damage requirement. As such, Ballard Industries has likely experienced a loss significantly greater than the minimum requirement.

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



## REPORT APPENDIX

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

## APPENDIX A-1

### Autopsy string Report

---

#### GENERAL INFORMATION

File: a:\\_ndex.htm  
MD5 of recovered file: 17282ea308940c530a86d07215473c79  
SHA-1 of recovered file: a6d082e1caf590ae8c08a1017178bebdfa78cf3d  
MD5 of ASCII strings: 0b8511be3a323360dba213413132a39d  
SHA-1 of ASCII strings: f26bf2c1d87837cd7930c409f2f77d52fa18dbb3

Image: /forensics/fl-260404-RJL1/fl-260404-RJL1/images/v1\_5.gz  
Image Type: fat12

Date Generated: Sat Oct 23 11:52:59 2004  
Investigator: JamesPerry

---

#### META DATA INFORMATION

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

File Type: HTML document text

---

#### CONTENT

```
<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDED" >
<center>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000">
```

```
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,0,0"
```

```
    WIDTH="800" HEIGHT="600" id="ballard" ALIGN="">
      <PARAM NAME=movie VALUE="ballard.swf"> <PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE=#CCCCCC> <EMBED src="ballard.swf" quality=high
bgcolor=#CCCCCC WIDTH="800" HEIGHT="600" NAME="ballard" ALIGN=""
  TYPE="application/x-shockwave-flash"
  PLUGINSOURCE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</center>
</BODY>
</HTML>
```

---

#### VERSION INFORMATION

Autopsy Version: 2.03  
The Sleuth Kit Version: 1.72

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

## APPENDIX A-2

### Autopsy string Report

---

#### GENERAL INFORMATION

File: a:\VCamShell.dll (\_AMSHHELL.DLL)  
MD5 of recovered file: 6462fb3acca0301e52fc4ffa4ea5eff8  
SHA-1 of recovered file: 3aa22c20039a7fa2d357888f6416a35fb0f0ee73  
MD5 of ASCII strings: f5b147de128821842e8fa2d9f2980211  
SHA-1 of ASCII strings: 93acce94a2d38f3bb32625bbb91928d4b426b751

Image: /forensics/fl-260404-RJL1/fl-260404-RJL1/images/v1\_5.gz  
Image Type: fat12

Date Generated: Sat Oct 23 11:48:50 2004  
Investigator: JamesPerry

---

#### META DATA INFORMATION

Directory Entry: 5  
Not Allocated  
File Attributes: File, Archive  
Size: 36864  
Num of links: 0  
Name: \_AMSHHELL.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

File Type: HTML document text

---

#### CONTENT

```

<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDED" >
<center>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"

codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,
0,0,0"
WIDTH="800" HEIGHT="600" id="ballard" ALIGN="">
<PARAM NAME=movie VALUE="ballard.swf"> <PARAM NAME=quality VALUE=high> <PARAM
NAME=bgcolor VALUE=#CCCCCC> <EMBED src="ballard.swf" quality=high bgcolor=#CCCCCC
WIDTH="800" HEIGHT="600" NAME="ballard" ALIGN=""
TYPE="application/x-shockwave-flash"
PLUGINS PAGE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</center>
</BODY>
</HTML>
IISheCamouflageShell
ShellExt
VB5!
CamShell
BitmapShellMenu
CamouflageShell
CamouflageShell
Shell_Declares
Shell_Functions
ShellExt
modShellRegistry
kernel32
lstrcpyA
lstrlenA
ole32.dll
CLSIDFromProgID
StringFromGUID2
ReleaseStgMedium
shell32.dll
DragQueryFileA
RtlMoveMemory
VirtualProtect
gdi32
CreateICA
GetTextMetricsA
CreateCompatibleDC
DeleteDC
GetObjectA
CreateBitmapIndirect
SelectObject
StretchBlt
DeleteObject

```

FindResourceA  
 advapi32.dll  
 user32  
 LoadBitmapA  
 LoadResource  
 advapi32  
 RegQueryValueExA  
 ModifyMenuA  
 InsertMenuA  
 SetMenuItemBitmaps  
 LoadLibraryA  
 SystemParametersInfoA  
 GetFullPathNameA  
 RegOpenKeyExA  
 RegCloseKey  
 \_\_vbaI4Var  
 VBA6.DLL  
 \_\_vbaCopyBytes  
 \_\_vbaFreeStrList  
 \_\_vbaFreeObj  
 \_\_vbaCastObj  
 \_\_vbaLateIdCallLd  
 \_\_vbaHresultCheckObj  
 \_\_vbaI2I4  
 \_\_vbaNew2  
 7\_\_vbaObjSet  
 \_\_vbaStrCmp  
 \_\_vbaStrVarVal  
 IContextMenu\_QueryContextMenu  
 \_\_vbaBoolVar  
 \_\_vbaObjSetAddr  
 \_\_vbaAptOffset  
 \_\_vbaAryDestruct  
 IShellExtInit\_Initialize  
 \_\_vbaStrVarCopy  
 \_\_vbaAryUnlock  
 \_\_vbaGenerateBoundsError  
 \_\_vbaAryLock  
 IContextMenu  
 \_\_vbaStr2Vec  
 \_\_vbaAryMove  
 \_\_vbaStrCat  
 \_\_vbaStrToUnicode  
 \_\_vbaFreeVar  
 F\_\_vbaStrVarMove  
 \_\_vbaStrMove  
 \_\_vbaStrCopy  
 \_\_vbaErrorOverflow  
 \_\_vbaFreeStr  
 \_\_vbaSetSystemError  
 \_\_vbaStrToAnsi  
 Class  
 C:\WINDOWS\SYSTEM\MSVBVM60.DLL\3  
 VBRUN

FIShellExtInit  
 C:\My Documents\VB Programs\Camouflage\Shell\lctxMenu.tlb  
 IContextMenu\_TLB  
 IContextMenu\_GetCommandString  
 IContextMenu\_InvokeCommand  
 \_\_vbaRedim  
 \_\_vbaUbound  
 \_\_vbaVar2Vec  
 \_\_vbaRecDestruct  
 \_\_vbaLsetFixstr  
 \_\_vbaLsetFixstrFree  
 \_\_vbaLenBstr  
 \_\_vbaFreeVarList  
 \_\_vbaFixstrConstruct  
 \_\_vbaVarTstEq  
 \_\_vbaVarMove  
 \_\_vbaVarCopy  
 \_\_vbaVarDup  
 7m\_szFile  
 IContextMenu  
 IShellExtInit  
 pidlFolder  
 lpobj  
 hKeyProgID  
 hMenu  
 indexMenu  
 idCmdFirst  
 idCmdLast  
 uFlags  
 idCmd  
 pwReserved  
 pszName  
 cchMax  
 lpcmi  
 pVfk  
 plVR  
 Pj@j  
 L\$ j  
 7hd(  
 7hd(  
 7hd(  
 Sh|)  
 j4hl)  
 7PWh  
 Qh<)  
 Vh|)  
 j4hl)  
 WPQj  
 B4Ph(  
 PQWWR  
 `SVW  
 Ph .  
 Ph .  
 Vh|)

Vh()  
 Ph .  
 t 9u  
 PVQR  
 MSVBVM60.DLL  
 \_Clcos  
 \_adj\_fptan  
 \_\_vbaVarMove  
 \_\_vbaFreeVar  
 \_\_vbaAryMove  
 \_\_vbaLenBstr  
 \_\_vbaStrVarMove  
 \_\_vbaAptOffset  
 \_\_vbaFreeVarList  
 \_adj\_fdiv\_m64  
 \_adj\_fprem1  
 \_\_vbaCopyBytes  
 \_\_vbaStrCat  
 \_\_vbaLsetFixstr  
 \_\_vbaRecDestruct  
 \_\_vbaSetSystemError  
 \_\_vbaHresultCheckObj  
 \_adj\_fdiv\_m32  
 \_\_vbaAryDestruct  
 EVENT\_SINK2\_Release  
 \_\_vbaObjSet  
 \_adj\_fdiv\_m16i  
 \_\_vbaObjSetAddr  
 \_adj\_fdivr\_m16i  
 \_\_vbaBoolVar  
 \_Clsin  
 \_\_vbaChkstk  
 EVENT\_SINK\_AddRef  
 \_\_vbaGenerateBoundsError  
 \_\_vbaStrCmp  
 \_\_vbaVarTstEq  
 \_\_vbaI2I4  
 DllFunctionCall  
 \_adj\_fpatan  
 \_\_vbaFixstrConstruct  
 \_\_vbaLatIdCallLd  
 \_\_vbaRedim  
 EVENT\_SINK\_Release  
 \_Clsqrt  
 EVENT\_SINK\_QueryInterface  
 \_\_vbaStr2Vec  
 \_\_vbaExceptionHandler  
 \_\_vbaStrToUnicode  
 \_adj\_fprem  
 \_adj\_fdivr\_m64  
 \_\_vbaFPException  
 \_\_vbaUbound  
 \_\_vbaStrVarVal  
 \_\_vbaLsetFixstrFree



\_Cilog  
 \_\_vbaErrorOverflow  
 \_\_vbaVar2Vec  
 \_\_vbaNew2  
 \_adj\_fdiv\_m32i  
 \_adj\_fdivr\_m32i  
 \_\_vbaStrCopy  
 EVENT\_SINK2\_AddRef  
 \_\_vbaFreeStrList  
 \_adj\_fdivr\_m32  
 \_adj\_fdiv\_r  
 \_\_vbaI4Var  
 \_\_vbaAryLock  
 \_\_vbaVarDup  
 \_\_vbaStrToAnsi  
 \_\_vbaVarCopy  
 \_Clatan  
 \_\_vbaStrMove  
 \_\_vbaCastObj  
 \_\_vbaStrVarCopy  
 \_allmul  
 \_Cltan  
 \_\_vbaAryUnlock  
 \_Clexp  
 \_\_vbaFreeStr  
 \_\_vbaFreeObj  
 CamShell.dll  
 DllCanUnloadNow  
 DllGetClassObject  
 DllRegisterServer  
 DllUnregisterServer  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 \_l:cu  
 DDDDDD@  
 DDDDDD@  
 DDDDDD@  
 DDDDDD@  
 "%R%  
 MSFT  
 stdole2.tlbWWW  
 lctxMenu.tlbWW  
 1CamouflageShellW  
 \_ShellExtWWWd  
 \_ShellExt  
 m\_szFile  
 2\$2\*20262<2B2H2N2T2Z2`2f2l2r2x2~2  
 3 3&3,32383>3D3J3P3V3\3b3h3n3t3z3

4"4(4.444:4@4F4L4R4Z4\_4 54585P5X5I5p5x5  
 5@6T6X6`6p6  
 7 7(70787@7H7P7X7`7h7p7x7  
 8 8(80888D8H8T8X8\8h8x8  
 9 9\$9(9,9<9@9D9H9L9P9p9t9x9|9  
 :0<<<@<L<h<x<  
 =\$,=4=T=X=\='=  
 ?8?<?D?Q?\?a?  
 0\$0(000=0H0M0|0  
 1%10151\1`1h1u1  
 2D2H2P2]2h2m2  
 3 3\$3,393D3I3d3h3p3}3  
 4!4,414X4\4d4q4|4  
 5 5%5@5D5L5Y5d5i5  
 6\$616<6A6h6I6t6  
 8,80888E8P8U8  
 9L:P:\$<4<8<<<  
 0 0,04080<0@0D0H0L0P0T0X0d0h0I0p0t0  
 1(1P1I1  
 2 2\$2(2,2024282<2@2(3  
 4#454:4`4k4  
 4%5,5<5E5]5r5  
 6#6,626F6L6V6\6o6  
 717G7j7~7  
 8!8A8K8f8n8s8{8  
 929G9h9x9  
 :q:e;  
 < <+<@<H<\_<g<p<  
 = (=C=I=Y=j=)=  
 =^>s>}>  
 ?!?= ?E?N?o?u?  
 0 020H0u0  
 1(1C1J1`1r1{1  
 2I2N2U2`2  
 2-3>3E3Y3o3  
 4#4-484P4V4  
 5%5B5`5o5y5  
 5"606>6G6R6X6n6|6  
 7\$7:7`7d7h7I7p7t7x7|7  
 868L8e8o8u8  
 9Q9b9  
 :!:-:F:N;j:r:  
 : ;+;>;D;N;T;m;u;  
 <0<R<n<  
 =#=4=w=  
 >\$>\*>=>H>  
 ?" ?F?O?\_?  
 0B0b0m0y0  
 101A1f1w1  
 2/2?2R2W2h2r2  
 3 3\$3(3.3

---

 VERSION INFORMATION

Autopsy Version: 2.03  
The Sleuth Kit Version: 1.72

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

## **Part One References:**

- 1.) "Computer Hardware, Information about computer floppy drives", URL: <http://www.computerhope.com/help/floppy.htm> (22October, 2004).
- 2.) TheFreeDictionary.com by Farlex. "DD (Unix)", URL: [http://encyclopedia.thefreedictionary.com/Dd%20\(Unix\)](http://encyclopedia.thefreedictionary.com/Dd%20(Unix)) (23October, 2004).
- 3.) Rivest, Ronald. "RFC 1321: The MD5 Message-Digest Algorithm", URL: <http://www.ietf.org/rfc/rfc1321.txt> (23October, 2004).
- 4.) "MD5, Message-Digest Algorithm", URL: <http://www.networksorcery.com/enp/data/md5.htm> (23October, 2004).
- 5.) "Autopsy Forensic Browser: Description", URL: <http://www.sleuthkit.org/autopsy/desc.php> (24October, 2004).
- 6.) "Definitions of DLL on the Web:", URL: <http://www.google.com/search?hl=en&lr=&oi=defmore&q=define:DLL> (24 October, 2004).
- 7.) "Guide to Internet Terms: A Glossary – HTML", URL: <http://www.getnetwise.org/glossary.php#H> (24 October, 2004).
- 8.) "Linux / Unix Commands: *strings*", [http://Linux.about.com/library/cmd/blcmdl\\_strings.htm](http://Linux.about.com/library/cmd/blcmdl_strings.htm) (25 October, 2004).
- 9.) "What is steganography? – A Word Definition From the Webopedia Computer Dictionary", URL: <http://www.webopedia.com/TERM/S/steganography.html> (27October, 2004).
- 10.) Bartlett, John. "Ease of Steganography and Camouflage", URL: [www.sans.org/rr/papers/20/762.pdf](http://www.sans.org/rr/papers/20/762.pdf) (27October, 2004).
- 11.) Mandia, Kevin and Proise, Chris. Incident Response: Investigating Computer Crime, Berkeley, California Osbourne/McGraw-Hill 2001.
- 12.) SANS Institute. Track 8 – Systems Forensics, Investigation & Response: Forensic Methodology Illustrated using Linux, Part 1 and 2, 2004.
- 13.) SANS Institute. Track 8 – Systems Forensics, Investigation & Response: Windows 2000/XP & NTFS Filesystem Forensics, 2004.
- 14.) SANS Institute. Track 8 – Systems Forensics, Investigation & Response: Computer Crime Law & Best Practices: Managerial and Legal Issues, 2004.

# **PART TWO:**

## **Option 2 - Perform Forensic Tool Validation**

## **Forensic Tool Validation of: Compromised Computer Inventory System (CCIS)**

Developed by: James D. Perry II  
GCFA Certification Candidate

### **EXECUTIVE SUMMARY:**

It was in August of 2004 that I truly discovered the value and importance of incident response plans and the necessary tools to execute them. As a security analyst with a major research university, I had grown accustomed to the weekly discovery of one or two compromised computer systems. On occasion, a compromise of significant impact would occur; however, the far majority of compromised systems were the run-of-the-mill hacker looking to take advantage of the massive storage capabilities of the typical university system; not to mention our 622Mbit/sec OC-12 Internet connection.

Past compromises of significance had taught me that, in general, the university was not prepared to respond to major incidents. First, the university lacked an official incident response plan that provided the necessary authority for the Information Security Office (ISO); which is the office I work for; to respond to compromised systems in a manner that was forensically sound. Second, the university's Office of General Council and campus police had traditionally appeared uninterested in pursuing legal action against attackers when possible. Third, while the university had invested in firewalls, intrusion detection/prevention systems, vulnerability assessment tools, etc.; little had been invested in forensic analysis tools. Fourth, the current security personnel had little training on the handling of evidence, forensic analysis, and legal implications regarding the incident response plan.

To address these issues, the university named an Information Security Officer. The result has been recent investments in training, acquisition of forensic tools, revised incident response plans, and better cooperation with the university's legal council and law enforcement offices. As mentioned earlier, I learned the value of these improvements in August of this year.

One otherwise normal August morning I arrived at work to discover two FBI agents in my office. Let me assure you, this is never a good sign. The purpose of their visit was to inform our office of a large scale international investigation of a particular hacker/group of hackers, and to deliver a court order to gather forensic data and perform a network trap and trace on the university systems involved. According to the FBI, numerous university systems had been used as launching points by the hacker(s) to attack various government agencies including the Department of Defense and Department of Energy.

Based on the information provided by the FBI, our office performed a preliminary analysis of the identified machines and determined the scope of the incidents could possibly include as many as 50 computer systems.

To this point, the university had traditionally sought permission from the system owner to remove the computer from its physical location and bring it to our office for forensic duplication. Thus, getting permission was a major obstacle as the system, often mission critical to the owner, would be unavailable for several days due to the lengthy forensic process. Furthermore, having 5 university campuses that are up to 375 miles apart, logistics became a major hurdle. To that end, it was clear that an automated method of performing forensic duplication needed to be developed, especially with potentially 50 different systems requiring forensic duplication.

As a result, the university quickly decided to send several employees in the Information Security Office to obtain SANS training on intrusion detection, hacker techniques, and forensic analysis with the goal of devising a knowledge base to establish better methods of performing wide-scale detection and duplication of compromised system.

During the Las Vegas 2004 SANS Conference, I attended Track 8 – System Forensics, Investigation, & Response that was taught by Rob Lee. Over the course of several days, Mr. Lee presented numerous forensic tools that aid in performing an analysis. However, it was apparent that there are differing opinions on how the forensic process should be ordered and, as a result, automated tools were lacking. Given my need to automate the process of gathering forensic data from remote campuses up to 375 miles away, I decided to develop a series of shell scripts and batch files to accomplish my goal. The result, as we'll discuss in detail in the following pages, is what I have called the "Compromised Computer Inventory System (CCIS)."

## **SCOPE:**

The scope of this project was to develop an automated method to send forensic data over a TCP/IP network from a compromised system to a forensic workstation. By automated I mean an easy to use tool that could conceivably be emailed to a remote administrator or employee, and with little direction, could be used to facilitate the collection of forensic data. As such, the following goals were established:

- Provide an automated script, for both Microsoft Windows and various Unix/Linux operating systems, to be run from the compromised computer that performs the following:
  - Collects forensic information in a forensically sound manner
  - Uses trusted binary files for the collection of information
  - Provides a mechanism for remote command execution

- Redirects all gathered data to a remote forensic workstation
- Provide an automated script to be run from a forensic workstation that performs the following:
  - Collects forensic data from compromised system and saves the data to a user specified location
  - Processes the hard drive/mount information to automate the process of performing forensic duplication
  - Provides a mechanism to collect forensic disk images using logical partitions or physical drives

To complicate matters, it should be noted (before a critique of my scripts are performed) that I have no scripting/programming experience. Unless, of course, you want to count the introduction to programming course I took in college back in 1994. As such, as a side benefit to developing these automated scripts, I am better prepared to automate several of the routine functions I perform in the future. To some this may seem like a trivial task. Nevertheless, given my lack of programming experience and limited Linux/Unix knowledge, I would suspect that many other forensic analysts' could benefit from such a tool, thus, my motivation in developing the tool.

The testing methodology will include the execution of the completed scripts on several "compromised" systems. Under normal circumstances, an MD5 "fingerprint" would be calculated on the compromised system prior to collecting forensic data. Professor Ronald L. Rivest of MIT created the MD5 algorithm to take an input file of arbitrary length, process the file through an algorithm, outlined in RFC 1321, to create a 128-bit "fingerprint."

As stated at <http://www.networksorcery.com/enp/data/md5.htm> "it is conjectured that it is computationally infeasible to produce two [different] messages having the same" fingerprint. Because of this fact, MD5 fingerprints are used as a reliable method to insure that the contents of a file have not changed. Thus, if a MD5 fingerprint is created for the data prior to executing the script and sending it over the network, and again created on the forensic workstation after the data is transferred, the two resulting "fingerprints" should be identical.

Unfortunately, in the real world this practice isn't always possible. A majority of the compromised systems we investigate are defined as mission critical and are usually a production system of some sort. After all, forensic investigation of a compromised system is usually only performed on high profile systems or those that have protected information stored such as Social Security Numbers, student records, and the like. Thus, forensic duplication is usually performed on live systems. As such, it should be noted that, since the collection is taking place on a live system with an active network connection, it should be understood that the system's data is dynamic. Network connections, process information, modified, access, and changed timestamps (called MAC



times), memory contents, etc. will all likely change. Furthermore, considering that a majority of the systems being imaged are servers, a normal image is easily 200GB or more. Thus, imaging a drive over the network takes several hours, sometimes days. Therefore, it is almost certain that system data will be different, thus rendering the MD5 fingerprint useless. Nevertheless, once the data has been collected, an MD5 fingerprint is useful to insure that the data was not modified as part of the investigation.

Since the purpose of this forensic tool is to merely automate the usage of open source forensic tools, it is assumed that the tools used, listed later, are forensically sound. As a precaution, the utilized tools will all be executed from a trusted CDROM that was created using many of the tools provided on the Track 8 CD from SANS in addition to a few others as outlined later.

### **TOOL DESCRIPTION:**

The Compromised Computer Inventory System (CCIS) was developed to automate the collection of forensic data from a compromised system using industry best practices and open source tools. It helps the forensic investigator by providing a simple configurable framework using scripts to send/collect data in a forensically sound manner, using a specific sequence, over a standard TCP/IP network.

Figure 1: System Overview

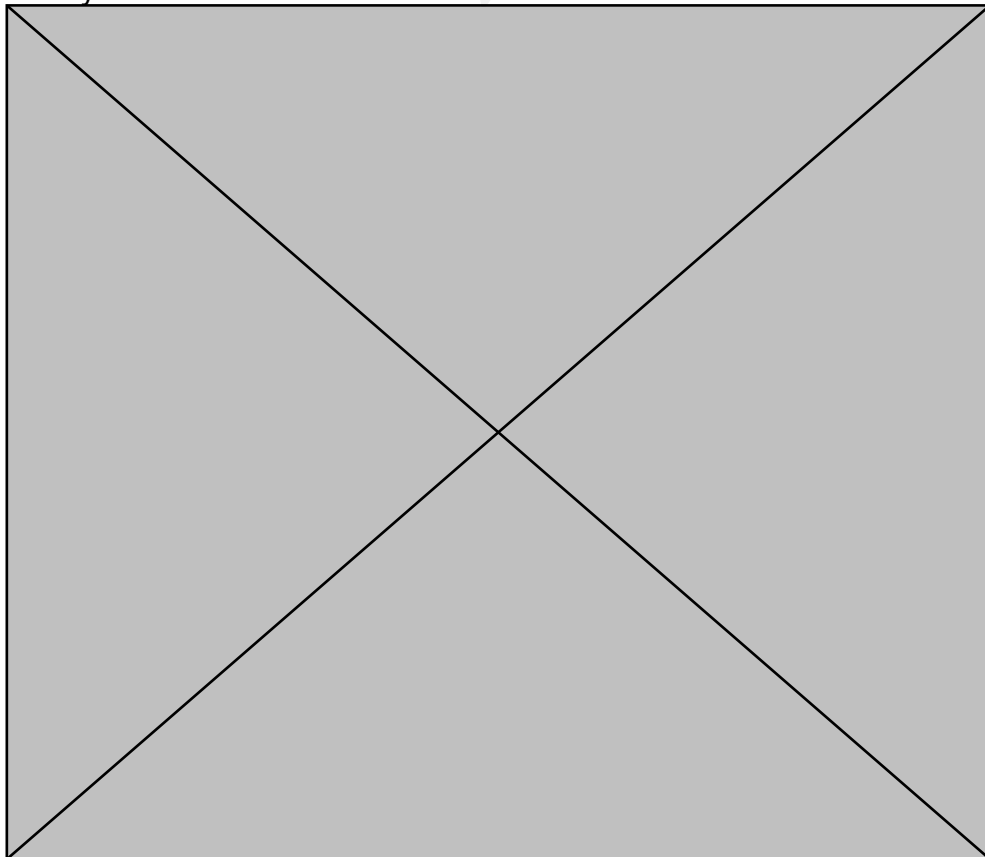


Figure 1, above, provides a high-level overview of the general design of the system. There are subtle differences between the methods used for Unix/Linux systems versus Microsoft Windows systems; however, the same mechanisms are utilized. Nevertheless, the tool will be demonstrated first using Linux and then again using Windows.

### Linux Step One: Script Execution on Forensic Server

The Compromised Computer Inventory System contains two main script components: a script named **ccis\_collector** that runs on the forensic evidence collection server and another called **ccis\_sender** that is to be run on a compromised system. As the names indicate, one collects and the other sends evidence. The complete scripts are located in Appendix A through C. Nevertheless, excerpts are located throughout this document as necessary.

The **ccis\_collector** script is the major element to the system. It is responsible for accepting initial input options from the forensic analyst and for launching several Netcat listeners to collect data from the compromised system and redirect it to a file with a specific name.

Figure 2, below, provides an overview of the commands executed in the first phase of the script. Line numbers have been added to the figure for readability. Line 1 of the script launches a bourne shell and feeds the remainder of the commands within the script to the newly created shell as standard input. Line 2 prompts the user to enter the type of system that forensic data will be collected from. At present the system has been developed for Unix/Linux and Windows systems, additional Operating Systems could be added as necessary. Again, we will be looking at the Unix/Linux process separately from Windows. Line 3 reads in the data typed by the analyst and saves it to a variable named *remote\_os*. Line 4 checks to see if the *remote\_os* variable is equal to Unix. If so, lines 5 through 16 will be executed, otherwise, the system bypasses these commands. Line 5 prints a blank line to standard out. This is used to make the screen output more readable. Line 6 prompts the user for the IP address of the remote computer, the compromised system, and line 7 saves the information to a new variable named *remote\_ip*. Next, line 9 asks the user to enter the location where the collected forensic data should be stored and line 10 saves the location information to a variable named *save\_to*. Lines 11 through 21 launches a series of Netcat listeners, described in the next paragraph, that only allow connections from the machine IP address saved in the *remote\_ip* variable and writes the received input to the location specified in the *save\_to* variable using a specific file name. The "&" at the end of each line launches the service and runs it in the background. This allows the script to continue processing, otherwise, line 12 would not execute until the process in line 11 finished. Line 22 and 23 is used to force the script to pause which signifies the end of step one. Step two and beyond hinge on information that is collected in lines 11 through 21. Therefore, once this data has been collected, the user enters "go" and the script

continues.

As mentioned above, Netcat is used to collect data from the remote computer. According to the GNU Netcat Project webpage, the variation used in all of the Linux testing for these scripts, "Netcat is a featured networking utility which reads and writes data across network connections, using the TCP/IP protocol. It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts." The CCIS scripts rely on Netcat to provide the network layer functionality for sending/receiving forensic data. The command syntax used in figure 2 is `nc -l -p {port number} {hostname or ip}` where "-l" tells Netcat to listen on "-p {port\_number}" for inbound connections from "{hostname or ip}." The ">" symbol instructs Netcat to redirect the received data into a file as specified by the `save_to/{filename}` data commands. Additional information about Netcat can be found at <http://Netcat.sourceforge.net/>.

Figure 2: **ccis\_collector** script excerpt

---

```

1      #!/bin/sh
2      echo -n What operating system is the remote computer '(Unix / windows)':" "
3      read remote_os
4      if [ $remote_os = "Unix" ]; then
5          echo
6          echo -n What is the IP address for the remote computer:" "
7          read remote_ip
8          echo
9          echo -n Where do you want the data stored:" "
10         read save_to
11         nc -l -p 221 $remote_ip > ${save_to}/memdump.img &
12         nc -l -p 222 $remote_ip > ${save_to}/mactimes.info &
13         nc -l -p 223 $remote_ip > ${save_to}/lsof_process.info &
14         nc -l -p 224 $remote_ip > ${save_to}/lsof_net.info &
15         nc -l -p 225 $remote_ip > ${save_to}/w.info &
16         nc -l -p 226 $remote_ip > ${save_to}/date.info &
17         nc -l -p 227 $remote_ip > ${save_to}/uname_a.info &
18         nc -l -p 228 $remote_ip > ${save_to}/fdisk.info &
19         nc -l -p 229 $remote_ip > ${save_to}/mount.info &
20         nc -l -p 230 $remote_ip > ${save_to}/trusted.info &
21         nc -l -p 231 $remote_ip > ${save_to}/forensic.info &
22         echo -n When the remote computer has transmitted the data, type "go':" "
23         read go

```

---

In my particular case, a dedicated machine has been configured specifically for forensic analysis. As such, all of the required software tools utilized in the script have been installed and are found in the system path. Nevertheless, one could modify the script to execute the server side programs using a specific location such as a CDROM.

In order for the **ccis\_collector** script to be executed, the default file permissions must be modified. Figure 3, below, illustrates the required command to allow the script to be executable. Essentially, this allows the read, write, and execute permission for root (the first #7). In addition it sets the group and user permissions to read and execute the script.

Figure 3: Setting the file permissions to allow execution

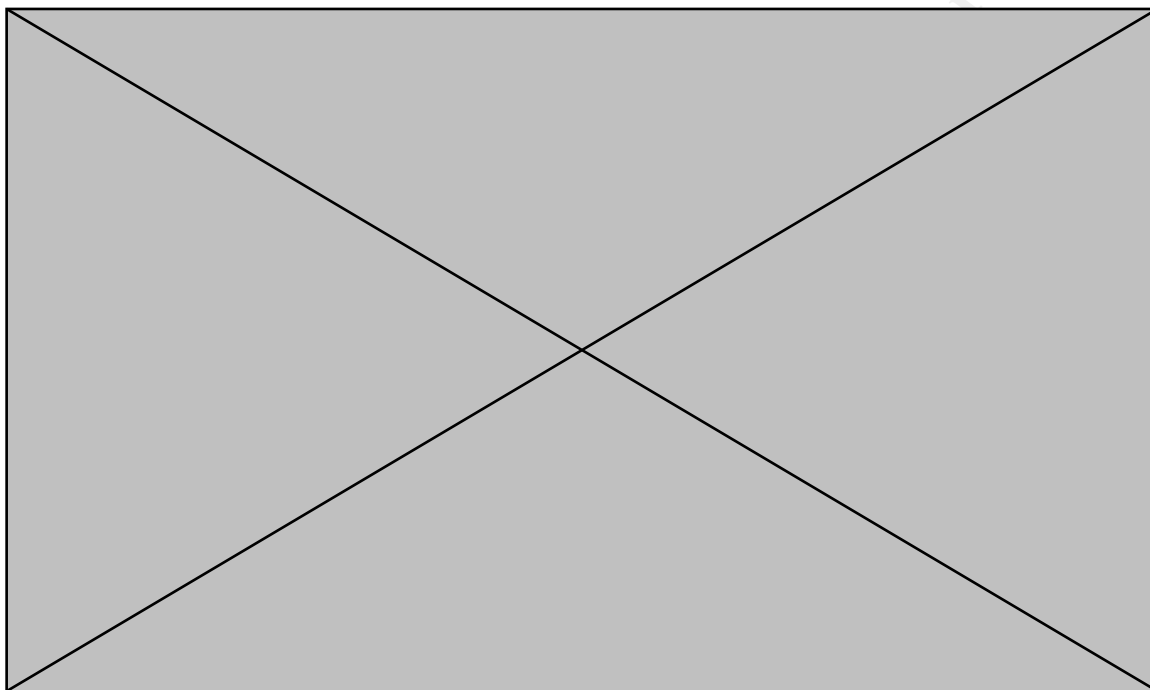


Figure 4, below, provides a sample of what the normal output would look like while performing forensic duplication of a compromised Unix system. Again, the user is asked to indicate the type of operating system being imaged, the IP address of that system, and the location where the evidence will be stored. Thus, Figure 4 illustrates that a Unix system with the IP address of 192.168.1.14 is being imaged and the collected evidence will be saved to the /evidence directory on the forensic server.

Figure 4 also notifies the analyst that the Netcat listeners have been started and specifies the TCP ports that are being used. In this case TCP ports 221 through 231 are being used. Again, the used ports can be customized by the analyst by modifying the script. Finally, the script prompts the user to type **go** once the remote system has transmitted the initial data. Thus, step one of the system is to define a few key elements such as type of operating system, remote IP address, and where the data should be stored. The script then launches several Netcat listeners in the background and saves the collected data to the specified location. Finally, step one is complete when the script prompts the user to enter go. The action of typing **go** should not be performed until step three.

Figure 4: Normal output from the ccsi\_collector script

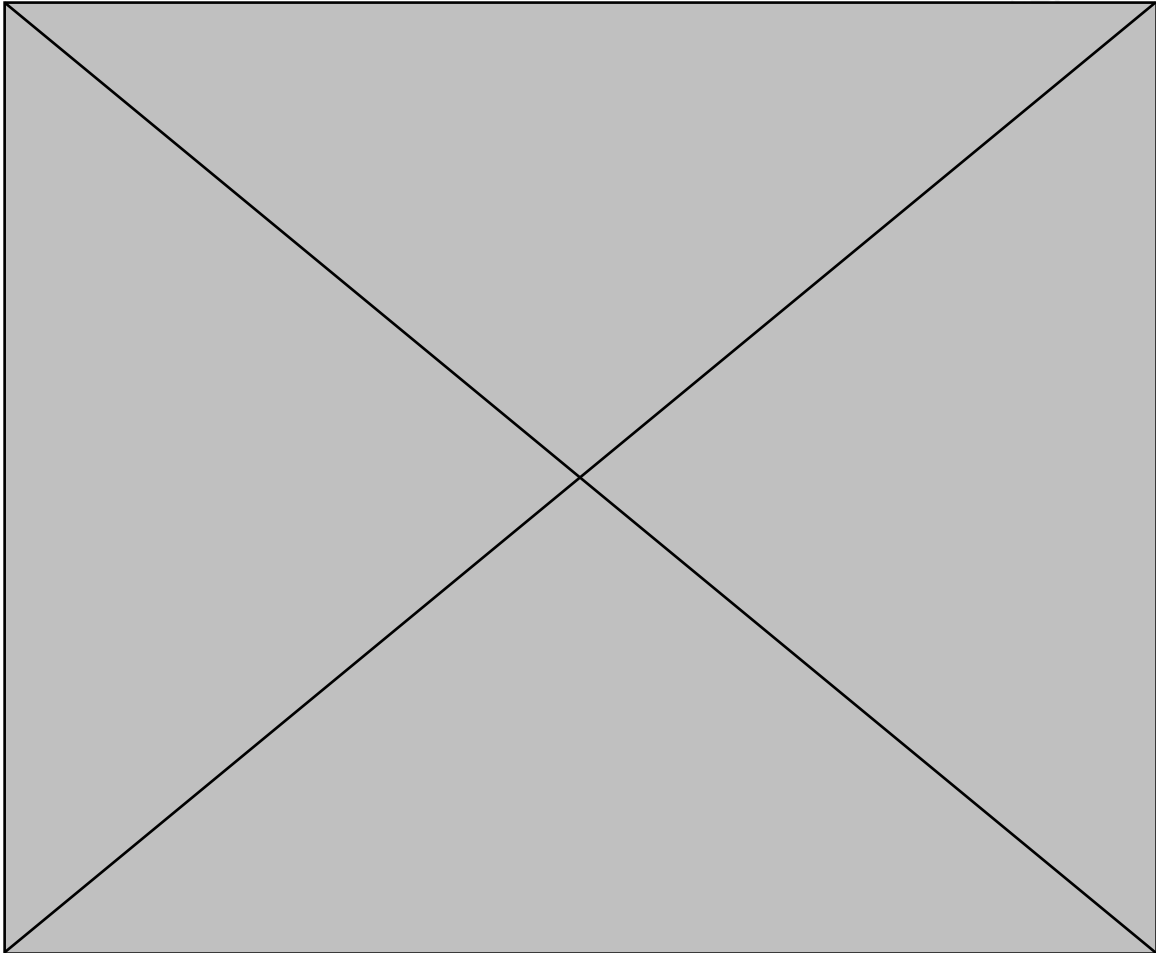
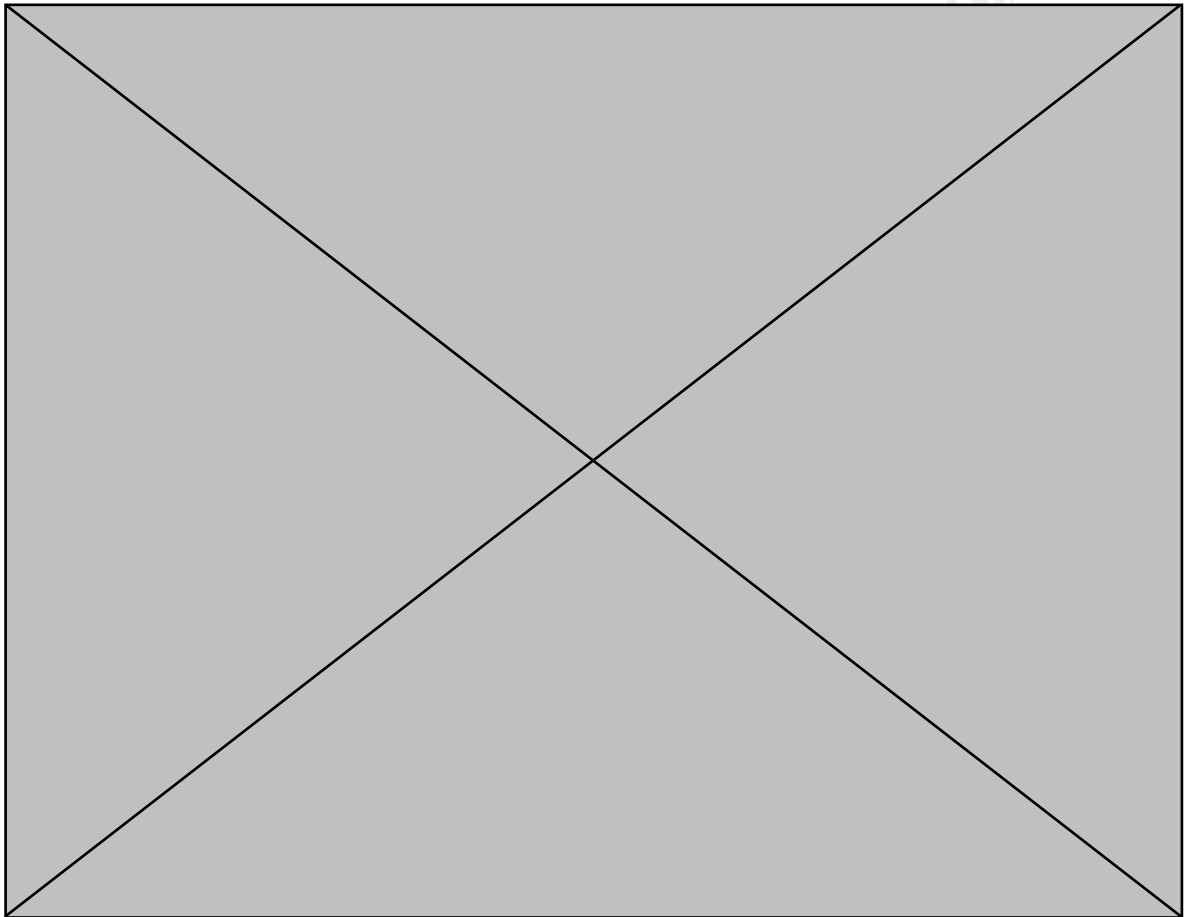


Figure 5, below, illustrates the effects of completing step one. First, the command ***ls -l | grep nc*** was executed in another terminal window. The purpose of the ***ls -l*** command is to list all open files and identify the program and process id that opened the file. Next, the results are piped to the ***grep nc*** command which parses the results from ***ls -l*** and only outputs the lines that contain “nc” which is the Netcat program. Thus, by entering ***ls -l | grep nc*** one can verify that the Netcat listeners have indeed been started.

The second command illustrated in Figure 5 is ***ls -l /evidence***. This command outputs file information for the specified file/directory including file sizes and names. Thus, we see that the file names coded into the script, above in Figure 2, have been created and show a current size of “0.” Thus, we know that Netcat is listening and it has created the files it will save the collected data to. At this point we are ready to move on to step two of the system which is to

execute the necessary script on the remote computer, our compromised system.

Figure 5: Verification that step one completed successfully



### Linux Step Two: Script Execution on Compromised System

The **ccis\_sender** script is used to facilitate the collection of forensic data on a compromised system. It is responsible for accepting initial input options from the forensic analyst and for launching several forensic tools, in a predetermined order and transmit the data to the forensic server over a TCP/IP network, again, using Netcat.

The collection of forensic data must be performed in a manner that preserves the integrity of the evidence as much as possible. As such, the order in which the script executes the forensic commands has been evaluated to achieve this goal. Nevertheless, varying opinions exist regarding which actions

should be taken and in what order. Again, the real value of using a scripted approach is that it can be customized to meet the needs of the individual analyst. Therefore, the order of execution and underlying assumptions of the **ccis\_sender** script are deliberate.

Prior to beginning step two of the process, it may be prudent to address how this tool is being utilized within my organization. As a major research university, traditional security devices such as firewalls, intrusion detection/prevention systems, and anti-virus gateways have been implemented. Thus, 90% of the time we are able to determine that a system has been compromised without entering the first command on the system console. Furthermore, due to the number of events we encounter, we usually only perform forensic analysis on high-profile systems. As a result, if a system of interest is compromised, we generally do not perform verification prior to forensic duplication as the verification process can and most likely will modify important data. Instead, we immediately perform forensic duplication and evidence collection, and then review the image to additionally verify the incident. To that end, the **ccis\_sender** does not collect system logs or evaluate the `/etc/passwd` file as is normally completed in the verification process. The primary goal is rather to collect the evidence with as little of impact as possible.

The **ccis\_sender** script was written in such a way as to preserve the integrity of the compromised system. Based on information provided in the SANS Institute Track 8 – System Forensics, Investigation & Response and that contained in the Incident Response: Investigating Computer Crime book written by Kevin Mandia and Chris Prosise, the collection of data is performed in the following order:

Order of Data Collection:

1. Memory Contents (using **memdump**)
2. MAC Timestamps (using **mac-robber**)
3. Active Process Information (using **lssof**)
4. Active Network Information (using **lssof -ni**)
5. Current System Users (using **w**)
6. Current System Time (using **date**)
7. Operating System Version (using **uname -a**)
8. Disk Drive Information (using **fdisk -l**)
9. Mount Point Information (using **mount**)

Figure 6, below, provides an overview of the commands executed in the **ccis\_sender** script. Line numbers have been added to the figure for readability. Line 1 of the script launches a bourne shell and feeds the remainder of the commands within the script to the newly created shell as standard input. The first two commands, line 2 and 3, prompt the user for the location of the trusted binary files and then saves the response to a variable named **trusted\_path**. A forensic analyst should never trust the integrity of binary files on a compromised

system. Thus, a common practice is to create a forensic response toolkit, usually a CDROM, with all of the software programs and utilities needed to analyze and collect forensic data. As such, the script asks the user where these files are located. Next, lines 5 and 6 prompts the user for the IP address of the forensic server where the collected data will be sent and save the input to a variable named *forensic\_ip*. Note that this MUST be the same machine that executed the **ccis\_collector** script in step one of the system. Lines 8 through 16 execute the necessary commands, located in the *trusted\_path* location, to collect the forensic data and send the results to the Netcat listener at *forensic\_ip*. Special attention should be paid to line 8 and 9 as they are NOT started as background processes using the “&” command shell option. Since line 8 is used to collect the physical memory, which is by its very nature volatile, the complete imaged must be received prior to executing any other command. Otherwise later commands in the script, if run simultaneously, would modify the contents of memory. Likewise, line 9 launches the utility to collect MAC times for the entire file system. Again, if other processes, located later in the script, were executed simultaneously, the MAC timestamps for various files and directories would be changed.

Line 17 and 18 uses the echo command to send what the user entered for the *trusted\_path* and *forensic\_ip* variables to the forensic server. This will be important in step three. Finally, line 19 launches a Netcat listener from the *trusted\_path* and uses the “-e” option to direct inbound connections to a command shell. The listener is launched on TCP port 232 and will only accept connections from the *forensic\_ip*. This allows the forensic server to issue remote commands to the compromised system in step three. This is an obvious security risk and should be used carefully. Adding the IP address of the forensic server to the end of the Netcat command provides an added level of security by only allowing connections from that address. Regardless, this functionality should only be enabled when needed and should closely be monitored.

Figure 6: ccis\_sender script excerpt

---

```

1      #!/bin/sh
2      echo -n Enter the location of the trusted binaries "(i.e. /mnt/cdrom/Linux)":" "
3      read trusted_path
4      echo
5      echo -n Enter the IP address of the forensic workstation "(i.e. 192.168.1.13)":" "
6      read forensic_ip
7      echo
8      ${trusted_path}/memdump | ${trusted_path}/nc -c ${forensic_ip} 221
9      ${trusted_path}/mac-robber / | ${trusted_path}/nc -c ${forensic_ip} 222
10     ${trusted_path}/lsof | ${trusted_path}/nc -c ${forensic_ip} 223 &
11     ${trusted_path}/lsof -ni | ${trusted_path}/nc -c ${forensic_ip} 224 &
12     ${trusted_path}/w | ${trusted_path}/nc -c ${forensic_ip} 225 &
13     ${trusted_path}/date | ${trusted_path}/nc -c ${forensic_ip} 226 &
14     ${trusted_path}/uname -a | ${trusted_path}/nc -c ${forensic_ip} 227 &

```