Global Information Assurance Certification Paper

GIAC Certified Forensic Analyst (GCFA)
Practical Assignment
Version 1.5

Part 1 – Forensic Data Report re: Ballard Industries

Part 2 - Option 2 - The Forensic Validity of Netcat 1.1

Submitted by: Michael Worman, GCIA
Submission Date: October 26th, 2004

Summary

This practical assignment fulfills the initial requirement for GIAC Forensic Analyst (GCFA) Certification. It is a two-part report examining several contemporary issues in the field of computer forensics. In the first part, a theoretical case of alleged corporate espionage is examined for potential forensic evidence. The entire forensic examination process is outlined including methodology, results, and documented conclusions.

The second part is a study of the forensic validity of a software tool. A potentially valuable forensic network utility, **Netcat**, is examined to determine whether its results are both verifiable and repeatable, and how the tool might aid in an investigation and/or criminal prosecution.

Part 1 - Analyze an Unknown Image

Forensic Data Report

Prepared for:
Mr. David Keen
Security Administrator
Ballard Industries

Summary

This report will detail the forensic examination of a floppy disk provided to our lab by David Keen, Security Administrator for Ballard Industries. The floppy disk was confiscated from Robert John Leszczynski, Jr., a Ballard employee, at 4:45 MST on April 26th, 2004. There is reason to believe that proprietary corporate information is being leaked from Ballard, and Mr. Leszczynski's actions are very suspicious and are in violation of Ballard's corporate security policies. The original floppy disk was imaged by the Ballard Security Administrator and a copy of the evidentiary image was provided for examination. The goal is to determine the nature of the data on the floppy disk and whether or not Mr. Leszczynski is involved in the suspected corporate espionage.

Evidence

There is only one piece of known physical evidence in this case:

One (1) 1.4M Floppy Disk

Chain of Custody Information
Tag# fl-260404-RJL1
3.5 inch TDK floppy disk
MD5: d7641eb4da871d980adbe4d371eda2ad *fl-260404-RJL1.img*
fl-260404-RJL1.img.gz

After accepting the floppy disk, the Chain of Custody form was signed and dated. (**Note: For the remainder of the practical this file is referred to by the name provided by SANS, v1_5.img**). This image was burned to a CD-R for further analysis. The floppy disk itself was stored in a secure evidence locker along with its Chain of Custody form.

Examination Environment

All examinations on the evidentiary image were performed on a new installation of Windows 2000 Service Pack 4, installed on a laptop computer dedicated for forensic analysis. A read-only CD containing a basic forensic toolkit (the SANS Track 8 CD Response kit, to be exact) was mounted for access to software necessary for analysis. The examination system itself was not connected to any computer network. The purpose of this was to ensure that any malicious code contained on the image was quarantined from any production networks, as well as to be certain that the forensic analysis system itself is free from compromise. This helps to preserve the confidentiality of our investigation as well as the integrity of all data involved.

Some analysis steps (such as running Autopsy to search for deleted files) were performed using Helix 1.5, a CD-bootable forensics and incident response-oriented Linux distribution. It is available for free download from E-fense at the following website:

```
http://www.e-fense.com/helix/
```

A second, Internet-connected PC was used for online research such as browsing the Cornell Law Library's United States Code Repository and for following investigative leads.

Examination Details

The first step in the analysis process was to verify that the evidentiary image provided by Ballard's security personnel was identical to the evidence originally seized, regardless of filenames. Ballard provided an MD5 hash value for the original floppy image, and by re-processing the image we can compare the two for any inconsistencies. A single-bit difference between the image originally seized and the image provided will generate completely different MD5 values. An examination of the file's MD5 hash produced the following output:

```
C:\ >md5 v1_5.img
d7641eb4da871d980adbe4d371eda2ad *v1_5.img
```

This value was compared to the MD5 hash provided by Mr. Keen:

```
MD5: d7641eb4da871d980adbe4d371eda2ad
```

Since the MD5 hash of the evidence image is identical to the one recorded on the Chain of Custody documentation provided by Mr. Keen, it can be assumed that the integrity of the image is intact. It has not been altered or corrupted since it was originally obtained.

The first step in analysis was to restore the data to its original form (that of a FAT12 filesystem) in order to view what the original data on the disk would have

looked like to a casual observer. A useful utility for this is **rawrite.exe**, which is commonly used for copying images to floppy media. The original image (**v1_5.img**) was copied with **rawrite.exe** to a blank 1.44M floppy disk. It was then mounted in a portable USB floppy drive and its contents were listed in a Windows 2000 command shell. The following file information was displayed:

```
04/23/2004  02:11p              42,496 Information_Sensitivity_Policy.doc
04/22/2004  04:31p              32,256 Internal_Lab_Security_Policy1.doc
04/22/2004  04:31p              33,423 Internal_Lab_Security_Policy.doc
04/23/2004  11:55a             307,935 Password_Policy.doc
04/23/2004  11:54a             215,895 Remote_Access_Policy.doc
04/23/2004  02:10p              22,528 Acceptable_Encryption_Policy.doc
               6 File(s)        654,533 bytes
               0 Dir(s)         798,208 bytes free
```

At first glance, it appears the floppy contains Microsoft Word documents related to internal information security practices at Ballard. This is immediately suspicious, as these types of documents should not be leaving Ballard on any media. It is also very odd that the suspect, Mr. Leszczynski, would be leaving the Research and Development areas of Ballard with documents that are not clearly related to that department. Thus, the documents warrant further inspection.

In order to view their “obvious” content, each of the six documents on the floppy disk was opened and examined in Microsoft Word 2000. A check of the local anti-virus logs showed no viruses or detectable malware. It was noted that each file was an information security policy document between one and five pages in length. The size of the files immediately seemed peculiar. The Word documents themselves contained no images and none of them should be more than a few kilobytes, based on their text-only content. Opening each Word document in the WinHex editor revealed that each file contained a large amount of binary data (possibly encrypted) in addition to the expected content for a Word file. This helped to explain why the documents were several times the expected size.

It seemed clear at this point that the data and the floppy disk were not quite what they seemed and a deeper analysis of the entire disk image, including deleted files, slack space, and unused portions of the floppy was warranted.
The **strings.exe** utility is useful for extracting ASCII and/or Unicode text from a binary image. Returning to the image file itself, the ASCII strings within it were extracted into an output file for analysis, using the following command:

```
C:\strings c:\v1_5.img > v1_5.img.strings
```

After examining the string fragments in this file, it became clear that there was a large amount of additional data on the floppy that was not part of the Word documents, possibly in hidden or deleted files or even within the Word documents themselves. Due to the recent suspected corporate espionage, a search for steganographic (i.e. data-hiding) tools is worthwhile.

After analyzing the initial pages of **strings** output, several lines seemed worth noting:

```
*\AC:\My Documents\VB Programs\Camouflage\Shell\CamouflageShell.vbp
```

This string indicated that some type of Visual Basic program was previously stored on the disk. Since there was no Visual Basic data on the floppy disk, this string was probably part of a deleted file. However the very name of the Visual Basic Project (*.vbp) file is suspicious, so there may indeed be some sort of data-hiding tool present.

```
Camouflage.exe /C
Camouflage.exe /U
```

This seemed to be the command line syntax used by a Visual Basic program called “Camouflage”. This clue warranted further investigation on the Internet to see if it is a known tool.

```
http://www.camouflage.freeserve.co.uk
CompanyName
Twisted Pear Productions
FileDescription
Keeps files containing sensitive information safe from prying eyes.
LegalCopyright
Copyright (c) 2000-2001 by Twisted Pear Productions, All rights
reserved worldwide.
```

These strings seemed to indicate the group responsible for developing the Camouflage utility, along with their website and a brief description of the program. It appears that Camouflage is indeed some type of steganographic (i.e. data hiding) application designed to conceal data from casual observation. Now that the tool had been identified, we can give Ballard security personnel something to look for on the Research and Development systems.

```
CamShell.dll
@RJL        FAT12
```

These lines appeared very important: the name of a specific file (a dynamic link library) associated with the Camouflage software distribution, as well as the Volume Label of the FAT12 partition associating the disk with Mr. Leszczynski, “RJL”. Knowing the name of a critical Camouflage library will allow us to search for signs of the program’s installation/use amongst the IT systems in the Ballard enterprise. More importantly, the Volume Label links the floppy disk directly to the suspect.

The next step was to examine the image in Autopsy 2.0 to search for deleted or hidden data. The forensic laptop was booted with a CD-R copy of Helix 1.5, a Linux distribution specifically designed for forensics and incident response. The original evidentiary image was mounted read-only in Autopsy 2.0. A quick

analysis of the floppy image revealed that there were two recently deleted files on the disk.

The first file, ***index.html***, was a 727-byte HTML file and appeared to be part of a Ballard web page associated with an embedded Shockwave Flash Object movie file named ***ballard.swf***. The purpose and hosted location of this ***index.html*** are unknown but can be provided to Ballard for further details. Although it looks inconspicuous, for all we know this file could be considered proprietary information by Ballard.

The following excerpt from the Autopsy 2.0 ASCII Report illustrates the contents of the file:

```
<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html;  charset=ISO-
8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDEDED">

<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"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer"></EMBED
>
</OBJECT>
</center>
</BODY>
</HTML>
```

The second file was far more interesting. The 36864-byte file was named ***camshell.dll***, indicating that this deleted file was the source of the Camouflage-related strings found within the ASCII strings of the image. Referring back to the strings output obtained earlier, it is becoming more likely that the Camouflage utility had not only been used on this floppy, but had also been deleted. The following details were noted on the Autopsy ASCII Report:

```
                    Autopsy ASCII Report

------------------------------------------------------------
                     GENERAL INFORMATION

File: a:\/CamShell.dll (_AMSHELL.DLL)
```

```
MD5 of recovered file: 6462fb3acca0301e52fc4ffa4ea5eff8

Image: /var/local/evidence/Practical/floppy/images/v1_5.img
```

To compare the deleted file to a stock installation of Camouflage, the research system was used to install and test the software. The files included in the Camouflage 1.2.1 package were also examined.

The test installation of Camouflage 1.2.1 included a 36864-byte ***camshell.dll*** file that appeared to have the same ASCII text strings as the deleted file, line for line. It was noted, however, that the MD5 hashes of the files were not the same. Running md5.exe on the stock version of ***camshell.dll*** revealed the following:

```
C:\Program Files\Camouflage>md5 camshell.dll
4e986ab0909d2946bed868b5f896906f *camshell.dll
```

It was noted, however, that the first 720 bytes or so of the deleted *camshell.dll* file were identical to the data in the deleted *index.html* file. This seemed to indicate that the original *index.html* file was copied into the *camshell.dll* before both files were deleted, perhaps in an attempt to overwrite any evidence of the Camouflage program itself. This would explain the MD5 differences, however due to the fact that the suspect chose to utilize a very small file to cover his tracks, a large portion of the deleted *camshell.dll* file data remained intact in slack space, and only the first 700 bytes or so were overwritten. It was the surviving data (bytes 728 to 36864 of the deleted file) that brought about our conclusions.

Conclusions and Recommendations

The following findings suggest that the suspect, Mr. Leszczynski, is indeed involved in the recently suspected corporate information leaks:

- The floppy disk contains evidence of a steganographic tool (Camouflage 1.2.1 or older).

- The FAT12 Volume Label on the floppy is “RJL”, the suspect’s initials. This serves to associate the floppy with the suspect. In other words, it would be difficult for the suspect to claim the disk was not his.

- There is substantial evidence that the Word documents on the floppy disk have been tampered with, and they are many times larger in size than they should be given their content.

- There is evidence of an attempt to destroy (overwrite) the steganograhic tools (Camouflage 1.2.1) used by the suspect.

- The “Camouflaged” files on the floppy disk are password protected, and cannot be recovered except with a password that Mr. Leszczynski should be able to provide.

System Administrators at Ballard will be able to identify instances of Camouflage by searching for the files *camshell.dll, camouflage.exe*, or any other file that is part of the standard Camouflage software package. Any instance of the text string “Camouflage” in the Windows Registry will also indicate the tool’s installation. System Administrators should also check for recently deleted files that appear similar to the Camouflage files (36864-byte files, for instance). Based on the apparent activities of the suspect, a search for additional suspicious (e.g. exceptionally large) Word documents might also prove valuable. Finally, Word documents matching the filenames of the those on the floppy should be searched for across the enterprise, as the suspect may have left additional evidence on the systems or network drives where they were originally located.

Image Details

The evidentiary file image was that of a floppy disk (FAT12) volume that contained the following files and timestamps:

```
04/23/2004  02:10p          22,528 Acceptable_Encryption_Policy.doc
04/23/2004  02:11p          42,496 Information_Sensitivity_Policy.doc
04/22/2004  04:31p          33,423 Internal_Lab_Security_Policy.doc
04/22/2004  04:31p          32,256 Internal_Lab_Security_Policy1.doc
04/23/2004  11:55a         307,935 Password_Policy.doc
04/23/2004  11:54a         215,895 Remote_Access_Policy.doc
```

Each file was processed using md5.exe, producing the following hashes:

```
f785ba1d99888e68f45dabeddb0b4541 *Acceptable_Encryption_Policy.doc
99c5dec518b142bd945e8d7d2fad2004 *Information_Sensitivity_Policy.doc
b9387272b11aea86b60a487fbdc1b336 *Internal_Lab_Security_Policy.doc
e0c43ef38884662f5f27d93098e1c607 *Internal_Lab_Security_Policy1.doc
ac34c6177ebdcaf4adc41f0e181be1bc *Password_Policy.doc
5b38d1ac1f94285db2d2246d28fd07e8 *Remote_Access_Policy.doc
```

There were also two deleted files within the image, uncovered with the Autopsy Forensic Browser. Selected information from the Autopsy ASCII Report on these files is included below:

Index.html

```
MD5 of recovered file: 17282ea308940c530a86d07215473c79
Size: 727
Written:    Fri Apr 23 10:53:56 2004
Accessed:   Mon Apr 26 00:00:00 2004
Created:    Mon Apr 26 09:47:36 2004
```

Camshell.dll

```
MD5 of recovered file: 6462fb3acca0301e52fc4ffa4ea5eff8
Size: 36864
Written:    Sat Feb  3 19:44:16 2001
Accessed:   Mon Apr 26 00:00:00 2004
Created:    Mon Apr 26 09:46:18 2004
```

A full MAC timeline for the image was extracted using Autopsy and is included with the practical as ***autopsy.timeline.txt***. The MAC timeline illustrates another peculiarity of the data that would have struck an investigator as odd: The files have modified timestamps of April 22th to the 23rd, but their creation times are dated the morning of the 26th. This could be a result of running Camouflage.

Since the Word documents were created or written onto a floppy (FAT12) partition, there is no owner, user, or group information included in the files. The only data signifying an individual is the Volume Label on the floppy partition: @RJL, indicating Robert John Leszczynski as the owner of the disk.

Forensic Details

Based on the ASCII strings found within the deleted ***camshell.dll file***, it appears that Mr. Leszczynski used Camouflage 1.2.1 by Twisted Pear Productions in an attempt to sneak some type of corporate data out of the company. Camouflage is a *steganographic* program designed to hide selected files inside of any other type of file. As observed, common side effects of steganography include files of inordinate size and odd MAC timestamps. The Word documents are clearly many times their normal size.

Camouflage appears to have been last used at 9:46am on April 26th 2004 to create the six "camouflaged" Word files contained on the floppy disk. This was the same day the suspect attempted to move the floppy disk past Research and Development Security.

In an attempt to further investigate the history and operation of the Camouflage program, an attempt was made to access the project homepage that was previously discovered in the strings output of the image:

http://www.camouflage.freeserve.co.uk
Twisted Pear Productions

The website did not appear to be maintained any longer, but a Google search for keywords "Camouflage" and "Twisted Pear" uncovered several existing mirrors of the original Twisted Pear Productions website. I found a link to the most recent version (1.2.1) at the following location:

http://camouflage.unfiction.com/

Downloading and testing this tool only took a few minutes, as the operation is very straightforward. Once Camouflage is installed, a right-click on any file will display “Camouflage” and “Uncamouflage” in the context menu. Selecting “Camouflage” brings up a dialog box for the user. The user simply verifies the file to be hidden by clicking “Next”, selects a “Camouflage Using” file, and finally selects a “Create this File” filename. It also allows the user to choose whether or not to set the file to read-only. The last step is an optional password that must be used to encode or encrypt the data (details on Camouflage’s algorithms could not be found online). The tool’s functionality was examined by Camouflaging and Un-camouflaging several test files, all of which exhibited the same peculiarities detected in the evidence files. The addition of unreadable binary data appended to the “disguising” files was immediately noted.

The Word documents on the evidentiary image all appeared to have been created with some version of Camouflage, most likely the latest version 1.2.1 given that the size of the Camouflage library is identical (36864 bytes).

Several attempts were made to Un-camouflage the Word documents on the disk, but it appeared that a password had been used to protect the data. The following passwords were tested with no success:

A blank password (Camouflage requests a password whether one was used or not)
“Robert”
“John”
“RJL”
“Leszczynski”
“Rift”
“Ballard”
“password”
“secret”

Although the password cannot be easily guessed, we can continue to attempt different passwords. Camouflage contains no lockout or anti-tampering features so Ballard can attempt as many passwords as they wish. A quick search through several security websites located no apparent security weaknesses for Camouflage, so there appears to be no way of bypassing whatever password has been set.

It would also be possible, given sufficient time and resources, to attempt to brute force the encryption itself if the Camouflage encryption algorithm could be obtained. This will require additional research into the program and may even require contacting the developers or analyzing the program in an debugging environment.

Program Identification

The original website for Camouflage is no longer active, the authors do not receive email (per their Contact Us page) and no instances of Camouflage source code could be found via web searches. Binary downloads of Camouflage 1.2.1 are available on existing mirrors of the Twisted Pear website, so this version was installed on a test system running Windows 2000.

Camouflage 1.2.1 installed four files into C:\Program Files\Camouflage:

```
Directory of C:\Program Files\Camouflage

09/17/2004  08:59p      <DIR>          .
09/17/2004  08:59p      <DIR>          ..
03/29/2001  10:13p             217,088 Camouflage.exe
02/03/2001  07:44p              36,864 CamShell.dll
03/28/2001  07:50p              11,649 Readme.txt
09/17/2004  08:59p              19,758 Uninst.isu
               4 File(s)        285,359 bytes
               2 Dir(s)  30,756,566,016 bytes free
```

It was noted that the ***Camshell.dll*** file installed by Camouflage 1.2.1 was the same size as the deleted ***camshell.dll*** found on the floppy disk, 36864 bytes.

As previously noted, the MD5 hashes of the deleted ***camshell.dll*** file on the floppy and a stock ***camshell.dll*** were not the same, and the deleted file was reported by Autopsy 2.0 as being an HTML document:

Stock 1.2.1 Library

```
C:\Program Files\Camouflage>md5 CamShell.dll
4e986ab0909d2946bed868b5f896906f *CamShell.dll
```

Evidence File (from Autopsy ASCII Report)

```
File: a:\/CamShell.dll (_AMSHELL.DLL)
MD5 of recovered file: 6462fb3acca0301e52fc4ffa4ea5eff8
File Type: HTML document text
```

The reason for this is clarified when examining the first 727 bytes of the deleted file ***camshell.dll*** file:

```
<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html;  charset=ISO-
8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDEDED">

<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"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer"></EMBED
>
</OBJECT>
</center>
</BODY>
</HTML>
```

The first 727 bytes are identical to the deleted file ***index.html***. Someone may have intentionally copied ***index.html*** into ***camshell.dll*** in an attempt to overwrite and destroy ***camshell.dll*** completely. Doing so would, in effect:

- Alter the file header of ***camshell.dll*** to give it the appearance of a deleted HTML file instead of a Dynamic Link Library (DLL)
- Change the MD5 hash of the deleted ***camshell.dll*** file to complicate any attempts to link it to parts of the Camouflage program

However since the original ***camshell.dll*** file occupied 36,864 bytes, a large amount of slack space remained intact even after ***index.html*** was copied over it. This was the “silver bullet” evidentiary data that remained intact, linking the otherwise inconspicuous floppy disk to the Twisted Pear Camouflage utility.

Although MD5 hashes cannot be used to verify that the file deleted from the floppy is, in fact, identical to the Camouflage DLL, there was a way to prove that the files are *nearly* identical, or at least identical enough to convince Ballard or a jury. It was possible to perform an MD5 hash on only the last 36,100 bytes of both files! If we can strip out the overwritten portion of the deleted ***camshell.dll*** file and compare it to the stock version, we should be able to produce a very high probability of similarity.

By using the **dd.exe** utility to copy part of the stock Camouflage library, we set the input block size to 1 byte and skip the first 730 bytes (the ‘damaged evidence’):

```
Z:\tools\response_kit\win2k_xp\dd ibs=1 skip=730 if=CamShell.dll
      of=c:\test.bin


Copying C:\Program Files\Camouflage\CamShell.dll to
c:\test.bin...
```

```
Output c:\test.bin (0 bytes)
36134+0 records in
8+1 records out
```

The file ***test.bin*** now contains bytes 731 to 36864 of the deleted ***camshell.dll*** file. Running MD5 on this new, 36,134-byte file obtains the following output:

```
C:\Program Files\Camouflage>md5 c:\test.bin
\4247ae5544e572e4aec0d1027a347140 *c:\\test.bin
```

Autopsy 2.0 was used to extract the deleted ***camshell.dll*** file from the floppy image, which was renamed ***camshell-restored.dll***. Running the same process as above to extract bytes 731 to 36864 reveals the following:

```
C:\tools\response_kit\win2k_xp>dd if=c:\CamShell-restored.dll
ibs=1 skip=730 of=c:\test-restored.bin

Copying c:\CamShell-restored.dll to c:\test-restored.bin...

Output c:\test-restored.bin (0 bytes)
36134+0 records in
8+1 records out

C:\tools\response_kit\win2k_xp>md5 c:\test-restored.bin
\4247ae5544e572e4aec0d1027a347140 *c:\\test-restored.bin
```

As it turned out, the MD5 hashes for these particular sections of each file were identical. Taking into account a 36,864-byte file size, this means that the files were at least 98.0% identical, leaving very little doubt as to the nature of the deleted file. This information (along with the identical file sizes) should be sufficient to prove to Ballard, law enforcement, or the court that the tool used on the floppy was indeed Camouflage 1.2.1.

Legal Implications

It can be proven that Camouflage 1.2.1 was used to create the Word documents at 9:46 AM on April 26th, 2004, and the suspect attempted to remove these documents from the Ballard premises later that day. It was also clear that several actions were attempted to eliminate any evidence left behind by the tool that was used. Further work may be required to determine what data was hidden inside the Word documents on the floppy, but some additional examinations into the Ballard R&D environment should turn up additional evidence supporting the prosecution of the suspect.

Based on the suspect's alleged activity, it is unlikely that any federal *computer crime* laws were broken. The Federal Computer Fraud and Abuse act does not seem to be applicable to the Ballard case since the computer systems involved are:

- Not owned or operated by the Federal Government

- Not owned or operated by a financial institution
- Not involved in interstate or foreign commerce

Likewise, Federal Wiretap and Snooping laws (USC Title 18, Subsections 2511 and 2701, the Federal Wiretap Act) appear inapplicable.

However, if a deeper investigation into the Word documents on the floppy disk as well as Ballard IT systems uncovers additional evidence of proprietary data leaving Ballard in "Camouflaged" files, the suspect may potentially be charged under several Federal Industrial Espionage statutes.

Although possible, it is unlikely that the suspect would be charged under Title 18, Subsection 1831 (the Economic Espionage Act) of the United States Code unless the following assertion was determined:

> "Whoever, intending or knowing that the offense will benefit any foreign government, foreign instrumentality, or foreign agent, knowingly"

Since there is no immediate evidence that the suspect's actions were linked to a foreign government or agent, the penalties defined in this Act are likely inapplicable. Still, additional investigation may show that the suspect was providing proprietary Ballard data to entities other than Rift.

It is highly likely that Mr. Leszczynski could be charged under Title 18, Subsection 1832 (the Trade Secrets Act) of the United States Code. His alleged actions meet the requirement in parts a-1 and a-2:

> (a)
>
> "Whoever, with intent to convert a trade secret, that is related to or included in a product that is produced for or placed in interstate or foreign commerce, to the economic benefit of anyone other than the owner thereof, and intending or knowing that the offense will, injure any owner of that trade secret, knowingly –
> "
>
> (1)
>
> steals, or without authorization appropriates, takes, carries away, or conceals, or by fraud, artifice, or deception obtains such information;
>
> (2)
>
> without authorization copies, duplicates, sketches, draws, photographs, downloads, uploads, alters, destroys, photocopies, replicates, transmits, delivers, sends, mails, communicates, or conveys such information;

If it can be argued that Ballard's fuel cells have any uses in United States interstate commerce (a possibility), the suspect could face the penalties outlined in Subsection 1832, namely a fine with no defined upper limit for individuals and up to 10 years in prison. If it can also be proven that Rift, Inc. is involved in the suspect's actions, that corporation may also be fined up to $5,000,000 in penalties.

If this incident were to occur within the State of New Jersey, the suspect would also be susceptible to a strong state computer crime statute:

> New Jersey Statutes Chapter 20 Title 2C
>
> 2C:20-25. Computer-related theft
>
> A person is guilty of theft if he purposely or knowingly and without authorization:
>
> a. Alters, damages, **takes** or destroys any data, data base, computer program, computer software or computer equipment existing internally or externally to a computer, computer system or computer network;
>
> 2C:20-26. Property or services of $75,000 or more
>
> **a.** Theft under section 4 of this act [FN1] constitutes a **crime of the second degree** if the offense results in the altering, damaging, destruction or obtaining of property or services **with a value of $75,000.00 or more**.
>
> 2C:20-27. Property or services between $500 and $75,000
>
> a. A person is guilty of a **CRIME of the third degree** if he purposely Or knowingly accesses and recklessly alters, damages, destroys or obtains any data, data base, COMPUTER, COMPUTER program, COMPUTER software, COMPUTER equipment, COMPUTER system or COMPUTER network with a value of $75,000.00 or more.
>
> 2C:20-28. Property or services between $200 and $500; degree of crime
>
> a. Theft under section 4 of this act [FN1] constitutes a **crime of the fourth degree** if the offense results in the altering, damaging, destruction or obtaining of property or services **with a value of more than $200.00 but less than $500.00.**

In summary, the suspect is most likely susceptible to both federal and state felonies carrying large fines and significant prison terms.

Additional Information

The following link is a publicly available mirror of the original Twisted Pear website. It is useful for researching the operation and uses of the last available version of Camouflage (1.2.1):

http://camouflage.unfiction.com/

The SANS Organization's Reading Room offers several interesting research papers on Steganography, the art of hiding data within data:

http://www.sans.org/rr/catindex.php?cat_id=54

Cornell University operates a very useful website for researching the full text of the Unites States Code. Title 18 contains federal laws related to computer crime (Subsections 1030, 2511,2701) as well as the Theft of Trade Secrets (Subsection 1832) and Industrial Espionage (Subsection 1831):

http://www4.law.cornell.edu/uscode/

http://www4.law.cornell.edu/uscode/18/

The New Jersey Legislature maintains the following website, which serves as a definitive source for NJ state laws, crimes, and penalties.

http://www.njleg.state.nj.us

Part 2 - Option 2: Perform Forensic Tool Validation

The Forensic Validity of Netcat 1.1

The field of computer forensics and (forensic science in general) has gained increased popularity in recent years due to the evolution of information systems and their ever-broadening reach into the daily lives of individuals. The media allows us to scrutinize case after case involving digital evidence, from the San Francisco Bay current diagrams found on Scott Peterson's personal computer to the email transactions of former Enron CEO Kenneth Lay. The popularity of television shows such as "*CSI*" illustrate the public's interest in the "cat and mouse games" of forensic investigations. This fascination serves to assist the field in many ways, and it will hopefully lead to growth and a continual evolution in skills, tools, and techniques. I would argue, however, that as younger faces join the field it is our duty as veteran investigators to teach them to look past the distracting "flashiness" of forensic science and the "bells and whistles" of tools that we sometimes employ. In fact, newcomers to the field should be taught that the simplest of tools can sometimes be the most useful and more importantly, the most defensible.

As the field evolves, the tools employed by investigators have evolved as well. Where there was once a barren market for digital forensic tools just five years ago, now examiners have access to a diverse selection of commercial and open-source utilities. Software products such as Encase and imaging tools from companies such as Logicube are certainly valuable in specific forensic situations, but care should be taken, particularly with those new to the field. An examiner should always keep in mind that the value of any forensic tool is not in fancy functions and glossy reporting (which can certainly be useful), but in simplicity and validity. A pen, for all its simplicity, is a forensic tool when it is used to record observations regarding a case. The simplicity of the pen also makes it defensible in court, as in the case of an investigator's notes submitted as evidence. Electronic notes can be subject to loss or tampering much more than those that are handwritten in a notebook. It would be difficult if not impossible for any attorney to attempt to argue the forensic validity of penned examination notes.

Likewise, digital forensic examinations should have a similar goal: evidence (when present) should be acquired, transferred, analyzed, and stored in the most simple, transparent, and verifiable ways possible. An expensive or complicated commercial tool might serve these purposes, but how would one defend the validity of a tool if the process involved were simply "point and click"? For some products (e.g. Encase), there is already a wealth of court precedent supporting their use. But with the numerous "forensic" utilities available today, an investigator must put their faith in many valuable tools for which no precedent has been established. It was in this spirit that I chose to examine the forensic

uses of **Netcat** in the hope that novice forensic examiners will continue to re-discover this handy utility for years to come.

Netcat is a free and very popular network utility available from @Stake, Inc. It is commonly referred to as the "TCP/IP Swiss Army Knife" tool due to an enormous number of potential networking uses. The tool provides a simple interface to the process of opening arbitrary TCP or UDP sockets. These sockets are software connectors that allow data transfer between Internet-capable applications. The sockets created by **netcat** can be used for very simple purposes such as copying files between instances of **netcat**, or they can be linked with other tools to provide advanced operations. **Netcat** can be used to copy entire forensic toolkits to a remote system, or it can be used with the **dd** utility to capture remote drive images. Netcat also provides an investigator with the ability to push the results of a remotely executed (perhaps automated?) tool across a network to another listening socket, useful when large amounts of data need to be stored and analyzed on a system other than the target.

Netcat has been examined as a general security tool before, in a SANS paper entitled "Netcat – The TCP/IP Swiss Army Knife" by Tom Armstrong (February 15, 2001). In it, the basic usage of **netcat** is described along with the available options. By any assessment, **netcat** is an invaluable tool in an incident response or computer forensics toolkit. Netcat has many obvious uses to a skilled security professional, but are all of these uses forensically sound? The validation test in this section of the practical will attempt to answer the following questions:

- In what ways is **netcat** useful as a forensic tool?
- In what ways can the data processed by netcat be validated or certified?
- Can netcat be counted on to produce verifiable and repeatable results?
- Does it have the potential to alter data in any way?
- Can the use of such a tool be supported in a court of law?

Testing Scope

Netcat is essentially a network data transfer tool that allows "piping" of binary or text data over a network connection, from one instance of **netcat** to the other. Our test will include two Windows 2000 systems: a "remote" target named SUSPECT1 and a dedicated forensic analysis station named ANALYSIS1.

The standard use of netcat on the SUSPECT1 platform will be that of a **netcat sender**, i.e. data will be redirected from a tool or file to netcat. Netcat can be used on a remote suspect system in a number of ways. If the remote system is already powered down, a trusted assistant in the remote location can mount the suspect system with a CD-based operating system such as Helix, Knoppix, or even some generic Linux distributions. In many cases, netcat is already included on the CD. In other situations (for instance, when a "live" system cannot be shut

down for analysis), netcat can simply be copied on to a CD-R and operated from the remote CD-ROM drive. The remote assistant simply needs to mount the CD and run the appropriate command. In situations where there is no remote assistance available but remote Administrator or root-level access is possible, **netcat** can be copied to the suspect system and run directly via a remote command shell (e.g. via SSH, SCP, or Sysinternals.com's **psexec** utility) or a scheduled job (e.g. via crond, Windows Task Scheduler). Although this scenario requires some interaction with the suspect environment, **Netcat's** small footprint (under 60 Kbytes) and lack of prerequisite dynamic libraries or other software make it a perfect, low-impact solution in most cases.

The use of **netcat** on the ANALYSIS1 system will be that of a **netcat listener**, i.e. **netcat** will bind itself to a local TCP or UDP port and enter a TCP/IP "Listening" state. Typically, data received from a listening instance of **netcat** can then be redirected anywhere, such as to a local file or even to another remote instance of **netcat**!

The **netcat sender** will connect to the **netcat listener**, and data will be transmitted from the remote, sending process (for our test case this will be the **dd** byte-copying utility) to the local, receiving process (a command shell output redirection to a local file). In essence, **netcat** forms a "data bridge" between the remote and local processes.

The scope of testing will include:

- How well netcat preserves the integrity of data transferred between systems
- What effects (if any) the execution of netcat has on either systems' running state, processes, etc. What system libraries does **netcat** access?
- Determining what methods exist for verifying the operation of the tool

Tool Description

Netcat 1.1 for Windows is available for both Windows and Unix platforms from @Stake at the following website:

http://www.atstake.com/research/tools/network_utilities/

According to the website, the tool was ported to the Win32 platform by Chris Wysopal in 1998, from Hobbit's original Unix package developed in 1996.

Netcat has many used to both the system administrator and security professional alike, but its use in forensics is considered particularly valuable. As a binary, it has a very small footprint (under 60 kilobytes). It does not write any unnecessary data to the local disk unless its output is redirected (to a file, for instance). It can

be quickly deployed to a target system and combined with just a few other tools to allow remote transfer of files, output, or even raw disk images.

The **listdlls.exe** utility is a tool available from Mark Russinovich at the following website:

http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml

This is a useful way of determining the system files that are accessed by a running instance of **netcat**. Whether it is executed in sending or listening mode (e.g. nc.exe –l –p <port number>), **netcat** accesses the following libraries under Windows 2000:

```
C:\tools\response_kit\win2k_xp>listdlls.exe -r nc

ListDLLs V2.23 - DLL lister for Win9x/NT
Copyright (C) 1997-2000 Mark Russinovich
http://www.sysinternals.com

------------------------------------------------------------------------------

nc.exe pid: 1592
Command line: nc -l -p 4444  -v

  Base        Size      Version         Path
  0x00400000  0x13000             C:\tools\response_kit\win2k_xp\nc.exe
  0x77f80000  0x7d000   5.00.2195.6899  C:\WINNT\system32\ntdll.dll
  0x7c570000  0xb8000   5.00.2195.6897  C:\WINNT\system32\KERNEL32.dll
  0x75050000  0x8000    5.00.2195.6603  C:\WINNT\system32\WSOCK32.dll
  0x75030000  0x14000   5.00.2195.6601  C:\WINNT\system32\WS2_32.DLL
  0x78000000  0x45000   6.01.9844.0000  C:\WINNT\system32\MSVCRT.DLL
  0x7c2d0000  0x62000   5.00.2195.6876  C:\WINNT\system32\ADVAPI32.DLL
  0x77d30000  0x71000   5.00.2195.6904  C:\WINNT\system32\RPCRT4.DLL
  0x75020000  0x8000    5.00.2134.0001  C:\WINNT\system32\WS2HELP.DLL
  0x782c0000  0xc000    5.00.2195.6603  C:\WINNT\System32\rnr20.dll
  0x77e10000  0x65000   5.00.2195.6897  C:\WINNT\system32\USER32.DLL
  0x77f40000  0x3e000   5.00.2195.6898  C:\WINNT\system32\GDI32.DLL
  0x77980000  0x24000   5.00.2195.6824  C:\WINNT\system32\DNSAPI.DLL
  0x77340000  0x13000   5.00.2195.6602  C:\WINNT\system32\iphlpapi.dll
  0x77520000  0x5000    5.00.2134.0001  C:\WINNT\system32\ICMP.DLL
  0x77320000  0x17000   5.00.2181.0001  C:\WINNT\system32\MPRAPI.DLL
  0x75150000  0xf000    5.00.2195.6897  C:\WINNT\system32\SAMLIB.DLL
  0x75170000  0x4f000   5.00.2195.6949  C:\WINNT\system32\NETAPI32.DLL
  0x7c340000  0xf000    5.00.2195.6695  C:\WINNT\system32\Secur32.dll
  0x77bf0000  0x11000   5.00.2195.6666  C:\WINNT\system32\NTDSAPI.dll
  0x77950000  0x2a000   5.00.2195.6666  C:\WINNT\system32\WLDAP32.DLL
  0x751c0000  0x6000    5.00.2134.0001  C:\WINNT\system32\NETRAP.dll
  0x77a50000  0xef000   5.00.2195.6906  C:\WINNT\system32\OLE32.DLL
  0x779b0000  0x9b000   2.40.4522.0000  C:\WINNT\system32\OLEAUT32.DLL
  0x773b0000  0x2f000   5.00.2195.6601  C:\WINNT\system32\ACTIVEDS.DLL
  0x77380000  0x23000   5.00.2195.6701  C:\WINNT\system32\ADSLDPC.DLL
  0x77830000  0xe000    5.00.2168.0001  C:\WINNT\system32\RTUTILS.DLL
  0x77880000  0x8e000   5.00.2195.6622  C:\WINNT\system32\SETUPAPI.DLL
  0x7c0f0000  0x61000   5.00.2195.6794  C:\WINNT\system32\USERENV.DLL
```

```
  0x774e0000  0x33000   5.00.2195.6625  C:\WINNT\system32\RASAPI32.DLL
  0x774c0000  0x11000   5.00.2195.6604  C:\WINNT\system32\RASMAN.DLL
  0x77530000  0x22000   5.00.2195.6664  C:\WINNT\system32\TAPI32.DLL
  0x71710000  0x84000   5.81.4916.0400  C:\WINNT\system32\COMCTL32.DLL
  0x70a70000  0x64000   6.00.2800.1552  C:\WINNT\system32\SHLWAPI.DLL
  0x77360000  0x19000   5.00.2195.6685  C:\WINNT\system32\DHCPCSVC.DLL
  0x777e0000  0x8000    5.00.2160.0001  C:\WINNT\System32\winrnr.dll
  0x777f0000  0x5000    5.00.2168.0001  C:\WINNT\system32\rasadhlp.dll
  0x74fd0000  0x1e000   5.00.2195.6602  C:\WINNT\system32\msafd.dll
  0x75010000  0x7000    5.00.2195.6601  C:\WINNT\System32\wshtcpip.dll
```

Under Windows XP SP1, **netcat** accesses far fewer system libraries:

```
Z:\tools\response_kit\win2k_xp>more c:\netcat.xp.txt

ListDLLs V2.23 - DLL lister for Win9x/NT
Copyright (C) 1997-2000 Mark Russinovich
http://www.sysinternals.com

------------------------------------------------------------------------------

nc.exe pid: 1388
Command line: nc -l -p 4444

  Base        Size      Version         Path
  0x00400000  0x13000           C:\tools\response_kit\win2k_xp\nc.exe
  0x77f50000  0xa7000  5.01.2600.1217  C:\WINDOWS\System32\ntdll.dll
  0x77e60000 0xe6000   5.01.2600.1106  C:\WINDOWS\system32\kernel32.dll
  0x71ad0000  0x8000   5.01.2600.0000  C:\WINDOWS\System32\WSOCK32.dll
  0x71ab0000  0x14000  5.01.2600.1240  C:\WINDOWS\System32\WS2_32.dll
  0x77c10000  0x53000  7.00.2600.1106  C:\WINDOWS\system32\msvcrt.dll
  0x71aa0000  0x8000   5.01.2600.0000  C:\WINDOWS\System32\WS2HELP.dll
  0x77dd0000 0x8d000   5.01.2600.1106  C:\WINDOWS\system32\ADVAPI32.dll
  0x78000000  0x87000  5.01.2600.1361  C:\WINDOWS\system32\RPCRT4.dll
  0x71a50000  0x3b000  5.01.2600.0000  C:\WINDOWS\System32\mswsock.dll
  0x76f20000  0x25000  5.01.2600.1106  C:\WINDOWS\System32\DNSAPI.dll
  0x76fb0000  0x7000   5.01.2600.0000  C:\WINDOWS\System32\winrnr.dll
  0x76f60000  0x2c000  5.01.2600.1106  C:\WINDOWS\system32\WLDAP32.dll
  0x76fc0000 0x5000    5.01.2600.0000  C:\WINDOWS\System32\rasadhlp.dll
  0x71a90000 0x8000    5.01.2600.0000  C:\WINDOWS\System32\wshtcpip.dll
```

Both the Unix and Windows variants of **netcat** can be statically linked and run directly from a CD, and neither has any special installation requirements such as special libraries or registry keys. It should be noted that **netcat** for Windows uses around 40 standard Windows 2000 dynamic link libraries (only 14 under Windows XP) during regular operation. Even if a CD-based instance of **netcat** is used on a target system, a compromised DLL file in the operating system could impact its operation significantly. If any evidence of operating system tampering is evident, only a statically linked version of **netcat** should be employed.

To ensure maximum integrity, it is possible to create a CD-R with the appropriate DLLs included, or to manually compile the Win32 source to statically link all of the necessary libraries. Doing so will create a much larger **netcat** binary, but it

would be completely standalone and independent of the system libraries in *\WINNT\System32*. Changing PATH variables to accomplish this, or recompiling nc.exe as a statically linked binary, is left as an exercise for the reader.

Test Apparatus and Environment

The testing environment consists of two Intel PC's in a two-node, isolated TCP/IP network. The first system, SUSPECT1 is meant to represent a production system that is part of an incident under investigation. SUSPECT1 has the following configuration:

Hostname	SUSPECT1
Description	An imaginary production server containing potential forensic evidence
Hardware	Intel Pentium 4 Platform, 3.0 GHz
Operating System	Windows XP Service Pack 1
Patch Level	All available security patches applied (except for XP Service Pack 2, still under testing)
IP Address	192.168.0.2

The analysis station used in this test is named ANALYSIS1, and is meant to represent the Forensic Analyst's workstation. ANALYSIS1 has the following configuration:

Hostname	ANALYSIS1
Hardware	Intel Pentium 4 M Platform, 1.6 GHz
Operating System	Windows 2000 Service Pack 4
Patch Level	All available security patches applied
IP Address	192.168.0.1
Tools	SANS Track 8 Toolkit mounted on CD-R Netcat for Windows 1.10 from @Stake

ANALYSIS1 and SUSPECT1 are connected via a UTP crossover cable to simulate a production network, 192.168.0.0/24. The two test systems have no access to any real production networks. In a real-world scenario the two systems would likely be geographically distant.

Testing Scenario and Procedures

In our primary test, we will examine the ability of **netcat** to forensically copy a suspicious file from a remote system (SUSPECT1) to the analysis platform (ANALYSIS1), providing a verifiable audit trail of the operation. We must show that **netcat**'s results are predictable, repeatable, and that some method of verification is possible.

For the actual test, we will assume the scenario presented in Part I of this practical ("Analyze an Unknown Image"). The suspect, Mr. Leszczynski, has been accused of using a steganographic (data hiding) tool that has been identified through a forensic analysis as Camouflage 1.2.1. The distribution of the utility includes a dynamic link library names ***camshell.dll***, and the software package itself must be installed via a self-extracting executable file. Instances of ***camshell.dll*** on any Ballard system indicate that the suspect used Camouflage on that system.

For testing purposes, we assume that a diligent System Administrator has searched for ***camshell.dll*** on all of the Ballard R&D production and test systems. On a critical Windows 2000 server used for developing fuel cell schematics, the System Administrator finds the following deleted file:

```
 Directory of C:\RECYCLER\S-1-5-21-XXX-XXX-XXX-500

02/03/2001  07:44p              36,864 Dc118.dll
               1 File(s)         36,864 bytes
               0 Dir(s)  30,714,478,592 bytes free
```

This file is exactly 36864 bytes and is likely a copy of camshell.dll that has been deleted but not completely purged from Windows. Because of the criticality of this server, it cannot be taken offline for imaging. We have been instructed that any evidence to be moved off of the system must be done in a low-impact manner so as to not negatively affect R&D. By having the System Administrator mount a copy of our CD and using the appropriate commands to pipe the file through **netcat**, we should be able to obtain a pristine forensic copy of ***dc118.dll*** from the Recycle Bin, and we should be able to verify that it is indeed a copy of the stock Camouflage 1.2.1 DLL.

Preparation Phase

In preparation for testing, the SUSPECT1 computer and ANALYSIS1 computer are connected with a crossover cable. (For the purpose of the test it can be assumed that SUSPECT1 has a second network interface available for connecting directly to ANALYSIS1).

Network connectivity between the two systems must first be verified, and we can use the simple **ping.exe** utility to test this:

Using ping to test connectivity from SUSPECT1 to ANALYSIS1:

```
C:\>ping 192.168.0.1

Pinging 192.168.0.1 with 32 bytes of data:
Reply from 192.168.0.1: bytes=32 time<10ms TTL=128
```

```
. . .
```

Using ping to test connectivity from ANALYSIS1 to SUSPECT1:

```
C:\>ping 192.168.0.2

Pinging 192.168.0.2 with 32 bytes of data:
Reply from 192.168.0.2: bytes=32 time<10ms TTL=128
. . .
```

To establish a pre-transfer baseline for comparison, we assume the System Administrator runs a local copy of **MD5.exe** to obtain the following hash value for the deleted file:

```
C:\RECYCLER\S-1-5-21-XXX-XXX-XXX-500>md5 Dc118.dll
4e986ab0909d2946bed868b5f896906f *Dc118.dll
```

The System Administrator mounts the CD-R containing **nc.exe** on SUSPECT1. Only one file is present on the CD:

```
 Directory of Z:\

01/03/1998  02:37p            59,392 nc.exe
               1 File(s)         59,392 bytes
               0 Dir(s)               0 bytes free
```

Test Phase 1 – Initiate Netcat Listener

For our first Test Phase, we will initiate a process for the **netcat** listener on ANALYSIS1. The following command syntax will start a listening process on TCP port 4444, and data received will be redirected to an output file. A 3-second limit is set to ensure that the **netcat** connection is closed once the transfer is complete:

```
C:\nc.exe –l –p 4444 –w 3 > camshell.test.bin
```

Test Phase 2 – Initiate Netcat Sender

Next, the netcat sender is initiated to begin transferring the suspicious file to ANALYSIS1. The command syntax for the **netcat** sender (SUSPECT1) is entered as follows:

```
C:\RECYCLER\S-1-5-21-XXX-XXX-XXX-500\cat Dc118.dll | z:\nc.exe
      192.168.0.1 4444
```

This will copy the data in ***Dc118.dll*** through **netcat** to the IP address of ANALYSIS1 on TCP port 4444.

Test Phase 3 – Results

After a short wait (approximately 1 second) the file transfer is complete, and we now have a complete copy of the suspicious file in the local file ***camshell.test.bin***.

```
02/03/2001  07:44p              36,864 camshell.test.bin
```

Now we must verify that the file transported via the two instances of netcat has not been altered in any way. If a single bit has been altered, the initial MD5 hash and the new hash will not be the same, and **netcat** will be considered weak or unusable for forensic purposes.

Test Phase 4 – Verification and Approval

Executing MD5 on this file reveals the following:

```
C:\>md5 camshell.test.bin
4e986ab0909d2946bed868b5f896906f *camshell.test.bin
```

Now we can compare this MD5 hash with the one obtained previously by the System Administrator. They are the same, so the data transferred is validated and no alteration of the evidence has occurred. In fact, it can already be shown that this MD5 hash is identical to the hash for a stock camshell.dll file included with Camouflage 1.2.1:

```
C:\Program Files\Camouflage>md5 CamShell.dll
4e986ab0909d2946bed868b5f896906f *CamShell.dll
```

The above data transfer test was performed a number of times to determine that the results are indeed predictable and repeatable. Forensically, this illustrates that **netcat** has a valid use as a forensic data transfer tool, and that a **netcat sender** does not change a single bit of data when redirected to a **netcat listener**. Evidentiary data of any form (whether a file, partition, volume, or raw disk image) is preserved when transferred through the pipe. But is there another way of producing a forensically acceptable audit trail for transfers performed with netcat?

As it turns out that, there is. Recent versions of netcat include an outstanding forensic option for logging the transfer in hexadecimal format, as shown below:

```
C:\tools\response_kit\win2k_xp>nc -h
. . .
        -o file         hex dump of traffic
. . .
```

This means that in addition to verifying the unique fingerprint of the file before and after transferring (via MD5), we can also record a hex dump of

the transfer to a file. Using the hex dump option can be a great benefit for two reasons.

First, it provides a basic log of the transfer itself, and even the hex log can be processed via MD5 to create a unique fingerprint of the network transfer itself. For a command-line tool with no-built in logging function, the hex dump option provides an excellent way to record use of the tool, and the process for doing so is built right into the utility.

Second, the hex dump itself can be a vital source of forensic data. Viewing the hex dump of the network transfer is essentially the same as opening the file itself in a hex editor, which is a common task in forensic examinations. If a hex editor is unavailable to the Forensic Analyst, the hex dump file created by netcat can simply be examined instead, revealing important information such as hex offsets, data, and ASCII translations of the data.

The Uses of Netcat for Forensic Imaging

Netcat's uses for imaging are not limited to single files. The tool enjoys a reputation as a tried and true method for moving hard drive images. By combining **netcat** with **dd** (another forensic favorite) we can transfer single partitions, entire hard drives, or even the live contents of physical memory (RAM) between **netcat** instances. The following command (executed on a remote **netcat sender**) will acquire the contents of RAM and push them to an analytical workstation where a **netcat listener** process awaits:

```
C:\dd if=\\.\PhysicalMemory | nc [analysis IP and port]
```

On the analysis station, the listening instance can simply redirect this image stream to a file, which can then be processed through additional tools such as **strings**, Autopsy, or Encase for analysis:

```
C:\nc –l –p [port] > physicalmemory.img
```

Obtaining an MD5 hash of the physical memory is problematic, however. The RAM of any running computer system is constantly changing by design, and it is absolutely certain that a second hash taken will not match any of the previous values. It is still prudent to obtain a value on the imaged RAM, however, as the MD5 hash still provides a timestamp value, and can always be used to verify the integrity of later copies of the physical memory with the one initially imaged.

The Uses of Netcat for Forensic Analysis

To an investigator, **netcat** certainly has more than one potential use in forensics, but its function as a simple network pipe is invaluable. Without having to install software such as FTP or SSH, we can move data between any two computers

capable of running netcat with little or no setup. Moreover, the data can be verified before and after transfer, and an optional hex log of the transfer can be generated to further validate the results.

Netcat was not meant to be an analytical tool, but it can certainly be used to facilitate many parts of the forensic analysis process. Data moved with the **netcat** methods described above is typically redirected (via a command shell redirection such as "> filename") and then the received file is examined. In our test case, a file in a remote Recycling Bin was received via **netcat** on the ANALYSIS1 system for analysis in the following manner:

```
C:\nc.exe –l –p 4444 –w 3 > camshell.test.bin
```

To the examiner, the ***camshell.test.bin*** file should be byte-for-byte identical to the file sent from SUSPECT1, and MD5 hashes were shown above the validate this. Once the file has been hashed, it can be viewed with any other tool for further analysis. It can be opened in Winhex, for example, to view the format of the data. It can be run through the **strings** utility to examine ASCII data within the file that could be important (in this case, the strings in ***camshell.test.bin*** can be compared to the Camouflage 1.2.1 library).

Analyzing the hex dumps provided by the –o option are simple to read (assuming the Forensic Analyst is familiar with hexadecimal):

File Offset	Data (in base 16)	ASCII Decoding

```
 00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 # MZ..............
 00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 # ........@.......
 00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 # ................
 00000030 00 00 00 00 00 00 00 00 00 00 00 00 c8 00 00 00 # ................
. . .
```

Presentation

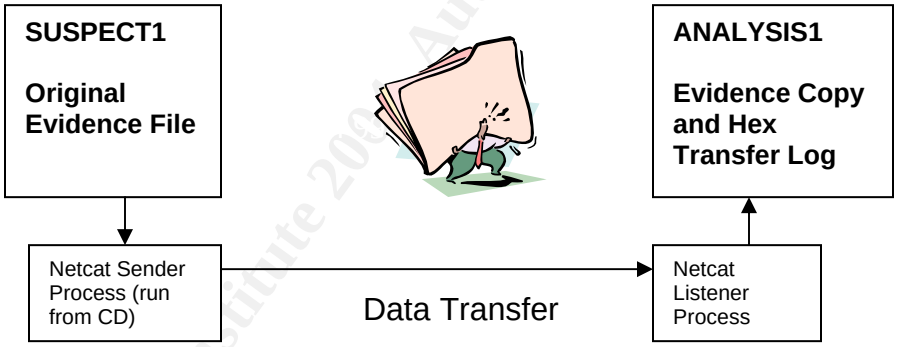
In a court setting where the jury (and potentially the judge) has little technical exposure, explaining the operation of **netcat** in a clear and concise manner is absolutely vital. The advanced technical processes behind raw TCP/IP sockets, terminal I/O redirection, and filesystem internals cannot be explained in a courtroom setting during the brief amount of time that a trial allows. Nor can anyone in the courtroom (with the exception of expert witnesses) be expected to follow any line of reasoning that requires a detailed technical dissertation.

An even more important concern when defending a forensic examination is the complexity of assertions made regarding evidence. Complex, detailed arguments (by either the defense or the prosecution) are the most easily assailed during cross-examination. Simple, straightforward arguments involving clear,

concise evidence are the most defensible and the logic is easier to follow. Our goal is to convince the jury that the evidence is sound and to remove the "mystery" surrounding its origin, so simplicity of form and function in our forensic tools is imperative.

As far as the presentation of **netcat** and its data is concerned, the simplicity of the tool is clearly its greatest asset. **Netcat** is not the fastest or the most secure method for transferring investigative data, but it is also one of the most transparent and flexible. For juries, attorneys, or judges with little or no technical background, the tool's basic command-line usage is very straightforward and specific. An examiner can display, explain, and if necessary justify the various command-line options with very simple analogies. By using simple ideas like "bridge" or "pipe", a non-technical jury can quickly grasp the nuts and bolts of **netcat**. In contrast, a complex commercial tool for remote imaging (a backup utility such as Symantec's Ghost, for instance) may produce useful data in an investigation, but explaining the processes involved in using such as tool is not nearly as cut and dry as a typing a single command and hitting 'Enter'.

The basic networking function that **netcat** provides can be very easily illustrated by the following diagram:



Netcat can thus be presented to the court as a simple data conduit, "pipe", or "bridge" between physically or logically separate computer file systems. Its value as a forensic tool is in the connection that it can create between two instances of itself. A file (or any chunk of data, for that matter) is simply "pushed" into the **netcat sender** process, which connects to a remote **netcat listener** process, which "pushes" the received data to some other location on the remote end. Data is copied verbatim (i.e. byte-for-byte), and nothing in the data stream is altered in any way.

In fact, netcat is so flexible that the sender and recipient systems don't even have to run the same operating system, something else that can help clarify the examiner's actions to the court. Data can be moved from a Unix system to a Windows system or vice versa. The "bridge" analogy works to our advantage again, and we can describe netcat as a "generic data bridge" between

contrasting computer environments, such as those of a suspect's Windows system and a forensic examiner's Linux workstation.

Conclusions

At the conclusion of our testing, **netcat** has displayed flexible use as an analytical tool, and that it is much more than a simple network copying utility. In a forensic investigation, **netcat** can be used to analyze remote potential evidence locally *without modifying any data*. The remote data can be a file or raw sectors from the physical disk drive, and this raw form can be preserved throughout a network transfer. In this fashion, **netcat** can used to create a forensically acceptable remote disk image, removing the need for personnel travel or expensive imaging hardware. Data moved via **netcat** can be analyzed or processed through many additional tools without any fear of manipulating original evidence, which is the fundamental test of forensic validity that we have attempted to prove.

Netcat should be a standard tool in any experienced examiner's forensic or incident response toolkit. Powerful yet simple and completely free, netcat will surely continue to guard its reputation as being one of the most versatile, trustworthy tools in the field of digital forensics. As new investigators enter the field, they should try to learn to look past the forensic "gadget" mentality and marketing hype that goes along with it. No tool is worth the cost of a failed investigation, defense, or prosecution.

As is true in most aspects of computer security, simplicity has great advantages and can lead to the greatest security. In the realm of computer forensics, a simple tool can help to secure your conclusions.

List of References

The United States Code Archive at Cornell University
URL:http://www4.law.cornell.edu/uscode/ (18 Sept. 2004)

Armstrong, Tom. “Netcat – The TCP/IP Swiss Army Knife”. 15 Feb. 2001. URL: http://www.sans.org/rr/papers/5/952.pdf (15 Sept. 2004)

Twisted Pear Productions. “The Camouflage FAQ”. Unknown Date. URL: http://camouflage.unfiction.com/(10 Sept. 2004)

The New Jersey State Legislature Website. URL:
http://www.njleg.state.nj.us (12 Sept. 2004)

The SANS Reading Room Website URL:
http://rr.sans.org (12 Sept. 2004)