
Fight crime.
Unravel incidents... one byte at a time.

# Lessons from a Linux Compromise

GCFA

Practical Assignment

Version 2.0

Option 2

April 13, 2005

John Ritchie
Security 508: System Forensics, Investigation and Response – San Francisco, CA.
November 18-23, 2004

## Abstract

This paper describes the forensic analysis of a Linux web server that had been in use at an agency within our organization when it was attacked and fully compromised. Although the intruder installed many tools to help him or her attack and compromise other systems, he or she was prevented from leveraging the server to spread within the agency network by the effective reaction on the part of agency technical staff, who detected the intrusion and quarantined the compromised server before the attacker was able to do further damage. I analyzed the system with the dual goals of completing my GCFA certification and fostering inter-agency cooperation and education; I was allowed to use the system with the understanding that I will share findings and methodology with my partner agency personnel.

# Table of Contents

## Document Conventions

When you read this practical assignment, you will see that certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

| | |
|---|---|
| `computer output` | The results of a command and other computer output are in this style |
| ***tool name*** | References in narration to operating system commands or forensic tools used are represented in this style |
| *cross-reference* | A cross reference to another section of this paper will be represented in this style. |

In accordance with the Administrivia for this practical assignment and per agreement with the agency whose server I'm analyzing, all references to the real organization name and real data will be masked. In referring to the agency, I will call it “Agency X,” and references to Agency X personnel will be in the form “Agency Tech 1,” “Agency Tech 2,” or “agtech1” and “agtech2” when displaying file ownerships. All Agency X external IP addresses will be masked using the 172.16.60.0/24 private IP range. Additionally, Agency X internal IP addresses will also be masked using the 192.168.0.0/16 private IP range.

All references to my own agency have been changed to “Agency Y.”

## Executive Summary

In January of 2005, a web server suspected of having been compromised many months before was forensically examined. The forensic examination consisted of making exact copies of the web server's disk storage and combing through those copies to discover hidden and illegitimate files and directories on the server to recover deleted files and to construct a time line of activity on the server. Where ever possible, external firewall and externally-stored system logs were examined and interviews were conducted with Agency X personnel to corroborate the evidence.

Evidence gathered on the web server ("web1") shows that it had been attacked from a specific host on the Internet in April 2004 and had been compromised using a flaw in the software used to encrypt the web services it provided. It was revealed from system logs and examination of the timeline of events that the intruder had quickly gained full administrative control of the machine.

Analysis of the file system and forensic examination of findings on the file system shows that the intruder had successfully replaced system software with tools designed to hide their activity on the system, to erase activity on the system, to monitor login activity on the system and capture passwords, to give them uncontrolled network access to the system, to trap unwary system administrators into granting access to other systems under their control, and to attack other systems. In short, the intruder had completely subverted the server and was in a position to gain information about, and launch attacks against, the agency network the server was placed on.

Although the intruder had gained complete control of web1, they weren't completely successful in their overall attack. There is evidence that some of their tools had not been built successfully and that some had not functioned correctly once installed. One of the tools they installed may have caused the system to become unstable; log file analysis shows that it had crashed and rebooted several times after it was compromised.

Interviews with Agency X system administrators reveal that these problems brought the attack to their attention. The system administrators determined that the machine had been compromised and disconnected it from the network, eliminating most of its usefulness as an attack vector. They effectively halted the compromise within hours of the initial attack.

Agency X system administrators didn't have the training to do the careful, in-depth forensic examination that is necessary to discover the attacker's activities without damaging the evidence. Examination of the system clearly shows many of the activities of the post-compromise investigations that Agency X personnel performed, some of which muddies the evidence left by the actual intruder. There is also evidence that the post-compromise investigations caused additional system damage and had the potential to worsen the compromise, possibly even spread it to other Agency X systems.

In accordance with my agreement with Agency X, I will use this practical assignment as a teaching tool for incident response and forensic techniques to more safely and effectively respond to future compromises. Together we will develop some "lessons learned" from this incident that our whole organization can use.

## Synopsis of Case Facts

The machine that I analyzed for this exercise was a test/development web server for another agency (Agency X) within our organization. According to discussions with Agency Tech 1, the agency deployed this web server primarily to test Apache and the RSA web agent. On the morning of April 6th, 2004, the server came under attack and was compromised. As a result of the compromise, the machine crashed or was rebooted. When Agency Tech 1 investigated, there were some system commands that were failing or acting strangely. This prompted further investigation and, at around 12:30 PM on the 6th, the server was determined to have been compromised and its access to the outside network was terminated.

Once Agency Tech 1 had determined that the machine had probably been compromised it was decided that it would need to be rebuilt to be of further use, and, in May of 2004, the server was powered off and stored until a rebuild could take place. Between that time and September 2004 the server was occasionally booted using a Knoppix CD to take part in network tests but wasn't booted from the hard drive until September 1st, 2004 when Agency Tech 1 took a final copy of the drive so the server could be rebuilt. That still hadn't happened in December of 2004, when Agency X allowed me to use the server for the purposes of my GCFA practical assignment.

My being allowed to analyze the machine is unusual within our organization and this exercise is being used as a cooperative precedent to further one of the goals of my office: that of developing an inter-agency Incident Response Team. This cooperative effort is helping our organization learn several things about inter-agency cooperation for response to an incident. I was allowed to analyze the server on the conditions that I maintain secrecy of agency data and that I share my findings with agency personnel for educational purposes. There are some incidentals as well, such as developing guidelines for inter-agency secrecy agreements and the development of a Chain of Custody form *(Appendix B)*.

Using this practical assignment as an educational piece presents some challenges. The assignment clearly states that findings and conclusions should be written such that they could be used in court and scrutinized by opposing council, but that writing style may be counter to an educational writing style. Another challenge is that one may wish to present more details about a procedure when writing educationally than is necessary to fulfill the practical assignment. In cases where there's a potential conflict in writing purpose I've created "educational footnotes" to separate purely educational content from the practical assignment body.

## System Description

The system in question is a dual-processor Intel-based workstation-turned-server on which Agency X had installed Redhat Linux. Agency X built the machine as a test/development/learning environment for Apache and the RSA web agent. The system is running Redhat Linux 7.3 kernel 2.4.18-3smp, Apache 1.3.26 configured with mod_ssl 2.8.7, mod_rsawebagent 5.2.0 and OpenSSL/0.9.6. Other software installed and running included webmin and OpenSSH version 3.1p1. The system was positioned in an Agency X internal network behind an Agency X firewall with a NATted private (RFC 1918) address. The firewall rule set allowed only port 443 (https) connections from outside to the web server but allowed outgoing port 80 (http) and port 25 (smtp) connections from the server to outside. The server was configured to send syslog output to an Agency X syslog server as well as to store it locally.

## System Hardware

**Tag #:** SoOESO2005_01-01
**Description:** Dell Precision 210 workstation/server, **model #** WCM. Custody of this server was given to me by the State of Oregon Agency X for forensic analysis. Refer to Forensics Letter of Agreement *(Appendix A)* and Chain of Custody Form *(Appendix B)*.
**Serial #:** UMG10D
**Agency X inventory #:** X123X-27675
**Detailed Description:** Computer system with dual 450 Mhz. Intel CPUs (each identified with a tag as "5137P RDJD"), internal CDROM reader, **Make:** NEC, **Model #** CDR-1901A, **Serial #** 9189114S115, internal 3.5" high density floppy drive, **Make:** Sony, **Model #** MPF920-F, **Serial #** 12591-8CS-0NLF, internal Quantum Fireball lct 30.0 GB drive (see **Tag #** SoOESO2005_01-02). System motherboard provides built-in sound, USB, keyboard and mouse, two serial and one parallel port and a 10/100 Ethernet port.
The system has the following detachable internal hardware cards:
one PCI Ethernet card, **Make:** unidentified, **Model #:** 100TX,
**Serial #:** 00D0B720E422 and
one graphics adapter, **Make:** Diamond Multimedia, **Model:** Fire GL 1000 AGP graphics adapter card, **Serial #** 0691200176090

System memory consists of:

2 - 32Mb PC-100 SDRAM sticks, **Make:** CMI, **Model #:** 464S2TG8-K,
**Serial #:** unidentified
1 - 128 Mb PC-100 stick, **Make:** unidentified, **Model #:** MH16S72BAMD-8,
**Serial #:** unidentified

for a total system memory of 192 Mb.

I have assigned a distinct tag number to the hard drive of the system because it may be desirable to detach the hard drive from the system for forensic analysis.

**Tag #:** SoOESO2005_01-02
**Description:** Quantum Fireball lct 30.0 GB hard drive with Agency X-attached label
**Make #:** Quantum Fireball lct
**Model:** 30.0 GB AT hard drive, **Part #:** LB30A011 Rev 01-A
**Serial #:** 176491530427 TAZXX
**Agency X label:** WEB1.AgencyX 192.168.200.14 172.16.60.6

## Image Media

The first step to take when performing forensic analysis of a hard drive is to obtain an exact copy of the drive to do analysis on. Doing so allows one to work with the information on the drive without risk that the original copy will be altered.

In order to prove that one is working with an exact copy of the information on the hard drive it is necessary to obtain a digital signature of the original information on the hard drive and be able to prove that copies of that information have the same digital signature. The common practice for doing this is to obtain an MD5 hash from the information on the hard drive, then make an exact copy of the information and obtain an MD5 hash from the copy to prove it's the same as the original.

I used the following techniques for obtaining an exact copy of the hard drive from the server web1.

First, I booted web1 and entered the BIOS menu and changed the BIOS to boot from the CDROM drive first. I then inserted a bootable ***Helix*** CD and rebooted the machine.

Once the system had booted from the ***Helix*** CD I configured web1's network interface and made it a member of the network that I had my forensic workstation on and tested connectivity.

To save the results of the initial MD5 hash of web1's hard drive, I used ***netcat*** to direct output from the ***md5sum*** tool to be transported over the network to my forensic workstation where it could be stored. I used ***netcat*** ("nc" in the output below) to transparently and efficiently move data between web1 and my forensic workstation.

On my forensics workstation (named "aardvark", with IP address 192.168.2.1) I did:

```
ritchiej@aardvark:~/projects/GCFA> nc -l -p 33333 > md5sums.txt
```

*Illustration 1: netcat listener output redirected to md5sums.txt*

This caused ***netcat*** to listen on port 33333 and directed the output to the file md5sums.txt.

On web1 I did:

```
root@ttpy0[~]# md5sum /dev/hda | nc -w 3 192.168.2.1 33333
```

*Illustration 2: piping md5sum via netcat to aardvark*

Doing this makes ***md5sum*** take the digital hash of the primary (only) IDE drive and direct output, via ***netcat***, to the IP address of my forensics workstation (192.168.2.1) on port 33333.

After the process finished I examined the file md5sums.txt on aardvark:

```
ritchiej@aardvark:~/projects/GCFA> cat md5sums.txt
133d5db8c5b52d3567e2d1f9f9918ec5  /dev/hda
```

*Illustration 3: MD5 hash of system harddrive*

This shows that the MD5 hash of the information on the IDE hard drive /dev/hda on web1 was "133d5db8c5b52d3567e2d1f9f9918ec5".

Once I'd obtained the MD5 hash of the original hard drive information I made an exact copy of the drive to my forensics workstation. I again used ***netcat*** to efficiently transfer the information over the network between web1 and aardvark.

On aardvark:

```
ritchiej@aardvark:~/projects/GCFA> nc -l -p 33333 > web1.img
```

*Illustration 4: redirection of netcat output to an image file*

I established a ***netcat*** listener on port 33333 again and directed the output to the file "web1.img".

On web1:

```
root@ttpy0[~]# dcfldd if=/dev/hda hashwindow=0 | nc  -w 3 192.168.2.1
33333
```

*Illustration 5: dcfldd copy of hard drive piped to netcat*

I used ***dcfldd*** to make a copy of the IDE hard drive device ("if=/dev/hda"). Specifying the "hashwindow=0" parameter caused ***dcfldd*** to do an MD5 hash of the total data it processed. This gave me an integrity check against the original saved MD5 hash I took of the hard drive before beginning analysis. I specified the IDE drive device /dev/hda as the input to this command and piped the output, via ***netcat***, to port 33333 of my forensics workstation 192.168.2.1.

Once the process was done, the MD5 hash output from ***dcfldd*** showed the same value as that of the original check:


*Illustration 6: dcfldd of disk image showing MD5 hash*

To confirm that the digital copy of the hard drive had been transported and stored without errors I did an additional MD5 hash of the image stored on my forensics workstation. I did an ***md5sum*** and concatenated the output of that sum into the file where I'd stored the MD5 hash of the original hard drive to make it simple to compare:

```
ritchiej@aardvark:~/projects/GCFA> md5sum web1.img >> md5sums.txt
```

*Illustration 7: concatenation of disk image MD5 hash to md5sums.txt file*

The display of the file "md5sums.txt" demonstrates that the MD5 hash for the original hard drive, /dev/hda, is the same as the digital copy on my forensics workstation, "web1.img:"

```
ritchiej@aardvark:~/projects/GCFA> cat md5sums.txt
133d5db8c5b52d3567e2d1f9f9918ec5  /dev/hda
133d5db8c5b52d3567e2d1f9f9918ec5  web1.img
```

*Illustration 8: MD5 hash of harddrive and image of harddrive*

# Media Analysis

## *Description of Analysis System*

The forensic analysis system I used for this assignment consisted of a Dell Latitude D400 notebook computer with a 1.7 Ghz. CPU, 2 Gigabytes of RAM, an external DVD CDRW and an internal 40 Gigabyte hard drive. Additional hardware was an Iomega 250 Gigabyte USB hard drive, of which only half was available for use, and an Hitachi 40 Gigabyte IDE hard drive attached via an IDE/USB adapter. Given the need to cryptographically protect the massive amounts of data this exercise generated *(Appendix A – Requirement 2)*, the external USB drive and the slow CPU were suboptimal and meant long waiting times to process data.

The analysis system was running SuSE Linux Professional 9.2, selected because of SuSE's broad software offering which includes critical forensics tools such as NTFS and cryptographic filesystem support, khexedit, ethereal and also because it was a new release at the time and had broad driver support for the Dell D400 notebook.

## Description of Tools Used

Software used in the analysis efforts included the following:

- VMware Workstation 4.5.2, VMware, Inc. (http://www.vmware.com) – used to build a copy of web1 for forensic and testing purposes.
- cryptographic filesystem, using loopfs, bundled with SuSE 9.2 – because of the data protection requirements of Agency X (see Appendix A – Forensics Letter of Agreement, requirement 2), it was necessary to ensure that data from the compromised system could not be forensically recovered in case of theft or loss, and it was necessary to allow for easy data erasure at the end of this project. In order to do this, I allocated half of the external USB hard drive to a single file, which I encrypted using the cryptographic file system and mounted using the loopback file system. I stored all copies of web1 disk data on that file system.
- Helix 1.5, e-fense Inc. (http://www.efense.com/helix/index2.html) - I used this bootable forensics CD any time I needed work with the compromised web server or the VM I made from it. It allowed me to maintain data integrity by not booting from the hard drive and it gave me a full, statically-compiled forensics laboratory to analyze the system with.
- Autopsy Browser 2.03 (http://www.sleuthkit.org/autopsy/) - I used Autopsy both for its automation of forensics functions but also because of its organizational capabilities. I found it much easier to manage the forensic data and notes as a case within Autopsy than I would have without it.
- Sleuthkit 1.73 (http://www.sleuthkit.org/sleuthkit/index.php) - I used the Sleuthkit because it has most of the analysis tools I needed. I used the tools bundled in ***Autopsy*** and I used the following components of the Sleuthkit individually as well:
    - mmls – the tool of choice for printing partition information from a disk image.
    - istat – lists detailed information about specific inodes.
    - dstat – lists detailed information about file fragments.
    - ifind – finds the inode associated with a specific allocated file fragment
    - ffind – finds the filename associated with an allocated inode.
- ethereal 0.10.6, bundled with SuSE 9.2 - used to monitor attempted network traffic from the VM instance of the compromised machine.
- dcfldd 1.0, Free Software Foundation – A tool for doing data block copying. Like the Unix command “dd,” makes binary duplicate copies but has added features like built-in MD5 hash functionality (using the “hashwindow” parameter) and a progress indicator. The MD5 hash functionality is useful for verifying forensic integrity as part of the copying process.
- khexedit 0.8.5, bundled with SuSE 9.2 – KDE tool used to edit binary files and to examine files at a binary level.
- foremost 0.69, written by Kris Kendall and Jesse Kornblum. - A tool for categorizing and extracting data from a disk image. Useful for “mass undeletion” of files, but it can be extremely greedy for disk space.
- lazarus, part of The Coroner's Toolkit by Dan Farmer - Another tool for categorizing and extracting data from a disk image. Lazarus will categorize

data on a block-by-block level. Although it is much slower than ***foremost***, it is also much less greedy for disk space.

- I used many tools and commands bundled with the Linux operating system during the course of this investigation. Below are some that had forensic significance:
    - fdisk – displays the partition table of the target hard drive.
    - file – identifies file types and summary information about files.
    - sudo – allows elevation of privileges as necessary, helping protect the forensics workstation from inadvertent damage.
    - md5sum – generates MD5 hashes of data.
    - strings – displays text (ASCII) strings from within a file.
    - ls -latr – directory listing command, sorted by reverse time order. Useful for finding files or directories with odd modification times.

## *Media Analysis Methods*

I used two main methods for direct media analysis of the suspect system: I used ***Autopsy's*** File Browsing Mode (available under the "File Analysis" tab), and I made copies of the file system and browsed the file system directly. Each method had its advantages.

***Autopsy's*** File Browsing Mode presents a view of the file system that includes deleted files and the File Browsing mode is closely tied with ***Autopsy's*** other forensic capabilities such as string searches, deleted file recovery and its ability to quickly view or export files. ***Autopsy*** can also be used to generate MD5 hashes of entire directories, facilitating proof of retention of forensic integrity.

Direct examination of the file system using Unix directory navigation and commands, on the other hand, is much more intuitive for the experienced Unix user and it allows the use of powerful Unix commands to examine the file system. However, because the file systems on the disk images weren't unmounted cleanly, it was necessary to create an analysis copy of the images that could be mounted, examined, then reverted to their original state to maintain their forensic integrity. I accomplished this by using ***VMware*** and creating a virtual machine (VM) for filesystem analysis; the description of this process is included below.

### Preparation For Autopsy

In order to prepare a case within ***Autopsy*** it was necessary to segment the raw disk image *(procured in Image Media)* into actual disk volumes. I used the following procedure to "chop up" the disk and prepare it for ***Autopsy***.

After copying the hard drive image to my forensics workstation I had used ***fdisk*** to display the hard drive volume characteristics. As before, I had piped the output from ***fdisk***, via ***netcat***, to my forensics workstation. The output from ***fdisk*** is:

```
Disk /dev/hda: 30.0 GB, 30020272128 bytes
255 heads, 63 sectors/track, 3649 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1           6       48163+  83  Linux
/dev/hda2               7        3601    28876837+  83  Linux
/dev/hda3            3602        3649      385560   82  Linux swap
```

*Illustration 9: fdisk output from raw disk device /dev/hda*

Although ***fdisk*** gave me useful information such as how big the drive is and how many volumes are on the drive with their size and name, it wasn't the best tool for the job of cutting up a drive because it doesn't succinctly describe the exact start and size of each partition of the drive. The tool for this is ***mmls***. Using ***mmls***, I listed the partition table of the disk image I had created:

```
ritchiej@aardvark:/forensics/GCFA> mmls -t dos web1.img
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:00   0000000063   0000096389   0000096327   Linux (0x83)
03:  00:01   0000096390   0057850064   0057753675   Linux (0x83)
04:  00:02   0057850065   0058621184   0000771120   Linux Swap /
Solaris x86 (0x82)
```

*Illustration 10: mmls listing of disk image partition table*

This tells us that the first partition starts in the 64th 512-byte sector and how many sectors long each partition is. Using this information I was able to extract parts of the drive image into individual partitions. I used ***dcfldd*** to extract the first partition (hda1), then checked my work with ***file:***

```
ritchiej@aardvark:/forensics/GCFA> dcfldd if=web1.img of=web1_hda1.img
bs=512 count=96327 skip=63 hashwindow=0
96256 blocks (47Mb) written.
Total: 57e85461e3ab45fa32ef6afea4a1898e
96327+0 records in
96327+0 records out

ritchiej@aardvark:/forensics/GCFA> file web1_hda1.img
web1_hda1.img: Linux rev 1.0 ext3 filesystem data (needs journal
recovery)
```

*Illustration 11: extraction of hda1 using dcfldd and check with file command*

Note the MD5 hash of “57e85461e3ab45fa32ef6afea4a1898e”.

Using ***file*** told me that I had a valid filesystem of type “ext3,” but it also told me that the filesystem needed journal recovery. This means that it was probably not unmounted cleanly the last time the machine was shut down.

I used the same methods to extract both the swap partition and the hda2 partitions.[*] Their respective MD5 hashes were: "f94362f83caa6c7ba7a4fc73be704426" and "7acde451e7cfd7bd844344861878f282." Doing ***file*** on the hda2 partition indicated that it, too, needed journal recovery.

Before loading the partitions into ***Autopsy*** it is necessary to determine their mount points within the original system. Normally this could be achieved by using the loopback filesystem, mounting each partition image in read-only mode, and examining the contents to determine where on the file system that partition belongs. Mounting the partition read-only would allow analysis of the media without damaging its forensic integrity. However, since each of the file systems was corrupted (they needed journal recovery), it wasn't possible to mount them in read-only mode because journal recovery couldn't happen without write access to the file system. Allowing journal recovery would damage the forensic integrity of the image. I demonstrated this by copying the smaller of the two partitions and doing test ***mount*** commands with it.

I first confirmed that I had an exact copy of the image:

```
ritchiej@aardvark:/forensics/GCFA> md5sum web1_hda1.img
57e85461e3ab45fa32ef6afea4a1898e  web1_hda1.img
ritchiej@aardvark:/forensics/GCFA> cp web1_hda1.img test.img
ritchiej@aardvark:/forensics/GCFA> md5sum test.img
57e85461e3ab45fa32ef6afea4a1898e  test.img
```

*Illustration 12: confirmation of copy integrity*

I then attempted to ***mount*** the image read-only onto the /mnt mount point using the loopback filesystem:

```
ritchiej@aardvark:/forensics/GCFA> sudo mount -o loop,ro,nodev,noexec
test.img /mnt
mount: wrong fs type, bad option, bad superblock on /dev/loop2,
       or too many mounted file systems
       (could this be the IDE device where you in fact use
       ide-scsi so that sr0 or sda or so is needed?)
```

*Illustration 13: unsuccessful read-only mount of partition image*

After that failed, I tried again, leaving the "ro" option off of the ***mount*** command. There were no errors and the partition mounted successfully.

I was then able to examine the /mnt directory and determine that this partition was originally the /boot partition for the system. This meant that the hda2 partition was almost certainly the root (/) partition, as it turned out to be. I

[*] I used the following ***dcfldd*** command line for extraction of the swap partition: "dcfldd if=web1.img of=web1_swap.img bs=512 count=771120 skip=57850065 hashwindow=0"; for the hda2 partition I used: "dcfldd if=web1.img of=web1_hda2.img bs=512 count=57753675 skip=96390 hashwindow=0". The "skip" number for each of these was taken from the "start" value provided by ***mmls*** and the "count" is the "length" value from ***mmls***.

unmounted the test partition image and confirmed that its MD5 hash had changed, and I did a ***file*** command to see that the filesystem had been repaired:

```
ritchiej@aardvark:/forensics/GCFA> sudo umount /forensics/GCFA/test.img
ritchiej@aardvark:/forensics/GCFA> md5sum test.img
0b01f01966d7387649bd32442df57d5a  test.img
ritchiej@aardvark:/forensics/GCFA> file test.img
test.img: Linux rev 1.0 ext3 filesystem data
ritchiej@aardvark:/forensics/GCFA> md5sum web1_hda1.img
57e85461e3ab45fa32ef6afea4a1898e  web1_hda1.img
```

*Illustration 14: confirmation of changed repaired filesystem*

Once I had determined the mount points of the two file system images I was able to import them into ***Autopsy*** and use ***Autopsy's*** "File Browsing Mode" to navigate through the file system and analyze it.

## Creation of a VMware Image

My analysis of the filesystem media was hampered because I couldn't mount the disks read-only and directly navigate the filesystem without forensically altering the data. Although I could have copied the data into another image file and mounted and repaired that filesystem I didn't have enough free disk space to support another full copy. Instead, I created a copy of the filesystem within ***VMware***, which allowed me to utilize ***VMware's*** filesystem compression and maintain a forensically correct copy at the same time. Also, using ***VMware's*** "Revert" functionality, I could make changes to the filesystem if necessary, then revert it back to be a forensically correct image copy.

The following figure illustrates the creation of a VM within ***VMware*** from a disk image while maintaining forensic integrity:


*Illustration 15: Creation of a VM from a disk image*

This is essentially the reverse of the process I used to create a copy of the original disk image. I created a VM within ***VMware***, booted the VM using ***Helix*** and configured the network. I then piped the image across the network using ***netcat*** to ***dcfldd***, which wrote the image onto the VM disk, maintaining its forensic integrity as shown by the MD5 hash I took of the image when finished: "133d5db8c5b52d3567e2d1f9f9918ec5". Because of compression the new image only used approximately 16 gigabytes of hard drive space.

Once the image had been copied to the VM I shut the VM down and took a snapshot of it using ***VMware's*** "snapshot" feature. After doing that I was able to examine the hard drive image using either ***Helix*** or by booting the machine and logging in, then restore the forensic integrity by using the ***VMware*** "Revert" feature to revert the hard drive back to its original state. I tested this technique by rebooting the VM from the installed copy of the server (which involved repairing the filesystem) and creating a test file on that server filesystem. I shut the VM down, "reverted" it, and booted it with ***Helix*** and performed an MD5 hash of the hard drive. The following snapshot shows the result of that ***md5sum***:


*Illustration 16: md5sum after VM filesystem change and revert*

The hard drive for the VM was reverted to its original state as can be seen because its MD5 hash was the same as that of the original hard drive.

## Analysis Using Autopsy

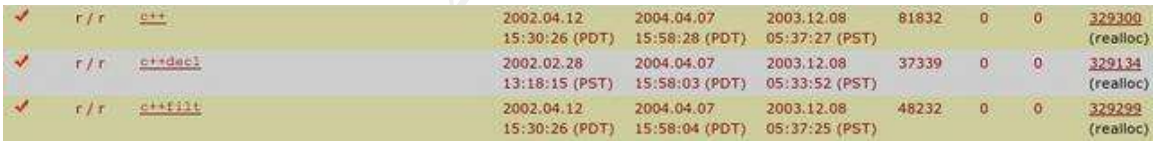
I didn't do much direct inspection of the file system and operating system components as part of the media analysis except to follow up on clues provided by other elements of the forensic investigation such as the timeline or string searches. I did do some cursory looking through system directories early in the investigation, though, scanning for clues.

While browsing the /usr/bin/ directory with ***Autopsy*** I could see several (17) entries with no data other than an inode number and file dates of 12/31/1969. This file date corresponds to the Unix "epoch," or date from which time is measured in Unix and, since file dates in Unix are stored internally as the number of seconds since that epoch, this indicates that the internally-stored value is zero for these files. File size, UID and GID are also zeros. Here's a snapshot of these null entries:

| DEL | Type dir / in | NAME | MODIFIED | ACCESSED | CHANGED | SIZE | UID | GID | META |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | - / - | | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 0 | 0 | 0 | 48 |
| | - / - | | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 0 | 0 | 0 | 48 |
| | - / - | | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 1969.12.31 16:00:00 (PST) | 0 | 0 | 0 | 3160373 |

*Illustration 17: null inode entries in /usr/bin*

Also present in the /usr/bin/ directory were hundreds of deleted files like the following:



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | r / r | c++ | 2002.04.12 15:30:26 (PDT) | 2004.04.07 15:58:28 (PDT) | 2003.12.08 05:37:27 (PST) | 81832 | 0 | 0 | 329300 (realloc) |
| ✓ | r / r | c++decl | 2002.02.28 13:18:15 (PST) | 2004.04.07 15:58:03 (PDT) | 2003.12.08 05:33:52 (PST) | 37339 | 0 | 0 | 329134 (realloc) |
| ✓ | r / r | c++filt | 2002.04.12 15:30:26 (PDT) | 2004.04.07 15:58:04 (PDT) | 2003.12.08 05:37:25 (PST) | 48232 | 0 | 0 | 329299 (realloc) |

*Illustration 18: deleted file entries in /usr/bin*

These deleted files were all kinds of utilities. It appears that there may have been significant file deletion activity occurring in the /usr/bin/ directory at the time the system was last shut down. This is further evidence of file system corruption as initially indicated by the "file" commands done on the volume images, discussed above in *Preparation for Autopsy*.

I used ***Autopsy's*** file browsing feature to examine many files and directories while following up on clues. I either viewed the files directly within ***Autopsy's*** "File Analysis" section by clicking on the file and viewing it, or I viewed the file, then exported it from there by clicking the Export button. In order to ensure that forensic integrity was maintained after the file was exported, I would select "Generate MD5 List of Files" button from the directory listing screen from which I

wanted to export one or more files. This would produce the following screen in a separate browser window:


*Illustration 19: MD5 hash values produced for a directory by Autopsy*

The contents of the browser window could then be saved off to a signature listing file. After extraction and analysis of a file from the forensic image I could compare MD5 signatures of the extracted file with the signatures stored for that directory and ensure that my analyses hadn't changed the subject file. Here's an example with the recovered /bin/ps file, saved to my forensics workstation as web1_hda2.img-bin.ps, and the md5 signatures for the /bin directory, saved as the autopsy-web1_hda2.img-bin.md5 file:

```
ritchiej@aardvark:/forensics/GCFA/web1/output/recovered> md5sum
web1_hda2.img-bin.ps
433b55d0fc16252023f5e8b04918614f  web1_hda2.img-bin.ps
ritchiej@aardvark:/forensics/GCFA/web1/output/recovered> grep ps
autopsy-web1_hda2.img-bin.md5
433b55d0fc16252023f5e8b04918614f        ps
```

*Illustration 20: checking forensic integrity of recovered ps file*

I browsed and viewed the following files and directories using ***Autopsy***:

- /usr/local/games directory and /usr/local/games/.sniffer file. This file shows root password changes, scp transactions to other machines (including the username and password on the remote system), telneting to a network switch device (including the username and password), and changing agadmin1's password.
- /etc directory and files in it: /etc/hosts to determine the system IP address; /etc/syslog.conf to confirm that syslog information was being sent to an AgencyX remote log server 192.168.200.3;
  /etc/redhat-release to confirm that the system was running Redhat 7.3.
- /etc/sysconfig/console directory and the files found there: default.netstat, default.ps, default.syslog, default.ls and default.socklist.

- /etc/rc.d/init.d system startup directory and many of the files there, including the /etc/rc.d/init.d/functions file that had been altered to cause startup of some of the rootkit tools upon system reboot.
- /root, the root user's home directory and recovered the /root/.bash_history and /root/chkroot.out files. These files helped determine what actions Agency X personnel had taken in response to the system compromise.

Using ***Autopsy*** I browsed and recovered each of the files installed during the rootkit build, shown in the timeline starting at Tue Apr 06 2004 08:31:55. I examined each of these files using strings and in some cases I referred to other sources for information; below is a list of each of them and what they do:

**/bin/ps:** ***strings*** examination of this binary reveals the hard-coded value of "/etc/sysconfig/console/default.ps" within the file; this appears to use that file as a configuration file to hide certain commands from being displayed by the "ps" command.

**/etc/sysconfig/console/default.ps:** Configuration file for /bin/ps. This contains the following lines:

```
3 slice
3 vadim
3 tcpd
3 kernel
3 stealth
3 ettercap
```

*Illustration 21: contents of /etc/sysconfig/console/default.ps*

**/bin/netstat:** ***strings*** examination of this binary reveals the hard-coded value of "/etc/sysconfig/console/default.netstat" within it; this binary appears to use that file as a configuration file to hide network connections from being displayed when the netstat command is used.

**/etc/sysconfig/console/default.netstat:** Configuration file for /bin/netstat. This contained the following lines:

```
1 194.102
3 101
3 112233
4 6660
4 6666
4 6667
4 6668
4 6669
4 7000
4 101
```

*Illustration 22: contents of /etc/sysconfig/console/default.netstat*

**/bin/ls:** Presumably trojaned copy of the "ls" command. Examination of this file using ***strings*** does not reveal the presence of the "/etc/sysconfig/console/default.ls" string.

**/etc/sysconfig/console/default.ls:** Presumably a configuration file for /bin/ls. Since /bin/ls makes no reference to this file it's unclear how it would be read. This contained the following lines:

```
psybnc
banner
tcp.log
psybnc.conf
wp
shad
tcpd
mech.set
mech.pid
mech.session
ftpusers-
identd
kernel
ettercap
Collected
```

*Illustration 23: contents of /etc/sysconfig/console/default.ls*

**/bin/socklist:** This showed up in the timeline with a modification and creation time during the rootkit build. Comparing the /bin/socklist Perl script recovered from the compromised machine and a clean one from my forensics workstation shows no differences. This file appears not to be trojaned.

**/etc/sysconfig/console/default.socklist:** Presumably a configuration file for /bin/socklist. Since the /bin/socklist script makes no reference to this file it's unclear how this configuration file would be read. Since /bin/socklist reads its values from the /proc filesystem it's possible that some other utility uses this configuration file to alter what's displayed by the /proc filesystem. This contained the following lines:

```
101
112233
nfsd
kernel
```

*Illustration 24: contents of /etc/sysconfig/console/default.socklist*

**/sbin/syslogd:** Presumably a trojaned copy of syslogd, this binary shows a creation time entry in the timeline at the time of the rootkit build. Doing ***strings*** examination of the file does not reveal the presence of the "/etc/sysconfig/console/default.syslog" string within the file.

**/etc/sysconfig/console/default.syslog:** Presumably a configuration file for /sbin/syslogd to control which strings would not be logged by it. Since that binary contained no reference to this file it's unclear how this configuration file would be read. This contained the following lines:

```
www.geocities.com/ixiondark
194.102
kernel
sshd
syslog
klogd
net-pf-10
modprobe
operator
games
promiscuous
PF_INET
60G
nix
dem
```

*Illustration 25: contents of /etc/sysconfig/console/default.syslog*

**/usr/bin/slice:** This binary was created at the time of the rootkit install. ***strings*** examination reveals that it is probably a Denial-of-Service attack tool. The "usage" section shows:

```
Usage: %s <target> <clones> [-fc] [-d seconds] [-s packetsize] [-a
srcaddr] [-l lowport] [-h highport] [-incports] [-sleep ms] [-syn[ack]]
    target      - the target we are trying to attack.
    clones      - number of attacks to send at once (use -f for more
than 6).
    -f          - force usage of more than 6 clones.
    -c          - class C flooding.
    -d seconds  - time to flood in seconds (default 200, use 0 for no
timeout).
    -s size     - packet size (default %d, use 0 for random packets).
    -a srcaddr  - the spoofed source address (random if not specified).
    -l lowport  - start port (1 if not specified).
    -h highport - end port (65335 if not specified).
    -incports   - choose ports incremental (random if not specified).
    -sleep ms   - delay between packets in miliseconds (0=no delay by
default).
    -syn        - use SYN instead ACK.
    -synack     - use SYN|ACK.
```

*Illustration 26: strings from /usr/bin/slice*

**/usr/bin/clean:** A bash shell script that goes through log files in /var/log and removes values from them. The value to be "cleaned" is supplied on the command line.

**/usr/bin/chsh:** The system timeline shows this binary with a modification and creation time updated during the rootkit build. A ***strings*** examination of the binary doesn't reveal anything obviously suspect that isn't consistent with functionality that chsh would normally have. The binary is setuid root on the system.

**/usr/bin/vadim:** According to ***strings*** examination, this binary appears to be an attack tool capable of spoofing the source IP address. In her GCIH Practical Assignment, Heather Larrieu calls this a "DoS" (Larrieu, p. 9)

**/usr/bin/stealth:** This binary may be an attack tool. Not much information about what the tool does is available using ***strings***:

```
Stealth > %s: port %d
Stealth > Non-existant host: %s
twitch@Stealth: This tool is extremely dangerous. Use at your own risk!
Usage: st-kill <host> <port>
```

*Illustration 27: "usage" strings from /usr/bin/stealth*

Since the binary was not stripped after linking, one can also glean information about its build environment from the strings output. Amongst lots of C declarations we see that this binary appears to have been built on a Mandrake Linux system, using GCC 2.95.2, and we can see the names of some of the source code files that were used to build the binary:

```
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.2/include/stddef.h
initfini.c
init.c
crtstuff.c
stealth.c
```

*Illustration 28: compile information strings from /usr/bin/stealth*

Larrieu refers to an “st” tool as a “stealth DDoS”, or distributed denial-of-service tool (Larrieu, p. 9).

**/usr/bin/wp:** Analysis of ***strings*** output from this file indicates that it was a wiping utility for cleaning user entries out the utmp, wtmp and lastlog log files:

```
USAGE: wipe [ u|w|l|a ] ...options...
UTMP editing:
    Erase all usernames      :   wipe u [username]
    Erase one username on tty:   wipe u [username] [tty]
WTMP editing:
   Erase last entry for user :   wipe w [username]
   Erase last entry on tty   :   wipe w [username] [tty]
LASTLOG editing:
   Blank lastlog for user    :   wipe l [username]
   Alter lastlog entry       :   wipe l [username] [tty] [time] [host]
        Where [time] is in the format [YYMMddhhmm]
```

*Illustration 29strings from /usr/bin/wp*

**/sbin/init:** ***strings*** output from this file indicates that it is almost certainly a version of the SucKIT kernel rootkit:

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:./bin
:/usr/local/games:/usr/local/games/bin
PS1=\[\033[1;30m\][\[\033[0;32m\]\u\[\033[1;32m\]@\[\033[0;32m\]\h \
[\033[1;37m\]\W\[\033[1;30m\]]\[\033[0m\]#
Can't open a tty, all in use ?
Can't fork subshell, there is no way...
BD_Init: Starting backdoor daemon...
FUCK: Can't allocate raw socket (%d)
HOME=/usr/local/games
HISTFILE=/dev/null
SHELL=/bin/bash
TERM=linux
pqrstuvwxyzabcde
0123456789abcdef
/dev/ptmx
/dev/pty
/dev/tty
/dev/null
/bin/sh
Can't execve shell!
FUCK: Can't fork child (%d)
Done, pid=%d
/usr/local/games/.rc
use:
%s <uivfp> [args]
u       - uninstall
i       - make pid invisible
v       - make pid visible
f [0/1] - toggle file hiding
p [0/1] - toggle pid hiding
FUCK: Failed to uninstall (%d)
Suckit uninstalled sucesfully!
FUCK: Failed to hide pid %d (%d)
FUCK: Failed to unhide pid %d (%d)
Failed to change %s hiding (%d)!
Detected version: %s
Pid %d is hidden now!
Pid %d is visible now!
file
%s hiding is now %s!
__kmalloc
/dev/kmem
RK_Init: idt=0x%08x,
sct[]=0x%08x,
FUCK: Can't find kmalloc()!
kmalloc()=0x%08x, gfp=0x%x
FUCK: Out of kernel memory!
Done, %d bytes, base=0x%08x
FUCK: Can't open %s for read/write (%d)
FUCK: IDT table read failed (offset 0x%08x)
FUCK: Can't find sys_call_table[]
FUCK: Can't read syscall %d addr
Z_Init: Allocating kernel-code memory...
core
/sbin/initsk12
FUCK: Got signal %d while manipulating kernel!
0123456789abcdefghijklmnopqrstuvwxyz
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
<NULL>
```

```
/dev/null
1.3b
sk12
/usr/local/games/.sniffer
/proc/
/proc/net/
socket:[
/sbin/init
/sbin/initsk12
login
telnet
rlogin
rexec
passwd
adduser
mysql
ssword:
```

*Illustration 30: strings output from /sbin/init*

Comparison of these strings with SucKIT source code (version 1.1c) found on Phrack (sd and devik, section 9) show many similarities; differences being attributable to version differences – note the "1.3b" above, a possible version number. According to Rik Farrow (Farrow, p. 13) and sd and devik (sd and devik, section 2), SucKIT manipulates the /dev/kmem device, giving it direct access to kernel memory. It uses this access to hide information about intruder activities and establish a backdoor to the system. According to ***strings*** output, /sbin/init referred to the /usr/local/games/.sniffer file where username/password information was stored from ssh and telnet sessions, and the strings "login," "telnet," "ssword," "passwd," etc., also support the conclusion that it was monitoring kernel memory for authentication-associated activity by applications and logged that information into the .sniffer file.

Placement of the SucKIT rootkit as a trojaned /sbin/init file would give it total control of the system upon system startup and upon runlevel changes, ensuring that control could be maintained between system boots.

**/sbin/initsk12:** According to ***strings*** analysis, this appears to be the original system /sbin/init binary. Installation of the SucKIT rootkit may have moved the original file to this location so that the trojaned /sbin/init could execute it to perform "init" functions. ***lsof*** output from the compromised system *(Illustration 35)* shows that initsk12 is running as PID 1 which would normally be the "init" process on a Linux machine.

**/usr/sbin/kernel:** According to ***strings*** analysis of the binary and ***lsof***, ***netstat*** and ***ps*** output from the running system, this appears to be a copy of an SSH daemon listening on port 101. It appears to use the file /etc/kernel_config as an sshd_config file. Examining the contents of that file confirm that it's configured to use port 101.

## Log Analysis

I had two sources of log information to work with when investigating the system: off-system logs stored on a syslog server, and log files that Agency Tech 1 had archived to examine as part of his determination of whether the server had been compromised. The former were provided to me by Agency Tech 1 but the latter I found on the server in the file `/var/log/backup/web1-logs-20040406.tar`, which contained copies of many system log files. I extracted the log file manually from my VM of the system by using ***netcat*** and piping it to my forensics workstation. The following screen shots show that I maintained forensic integrity of this file.

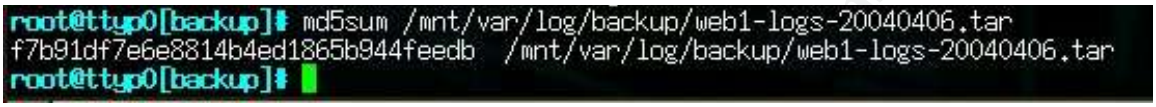



*Illustration 31: md5sum of web1-logs-20040406.tar file on filesystem*

The above is the MD5 hash of the file as it existed on the filesystem (mounted as /mnt from the ***Helix***-booted VM), and this is the MD5 hash of the file after recovery to my forensics workstation:

```
ritchiej@aardvark:/forensics/GCFA/web1/output/recovered> md5sum
web1_hda2.nc-var.log.backup.web1-logs-20040406.tar
f7b91df7e6e8814b4ed1865b944feedb  web1_hda2.nc-var.log.backup.web1-
logs-20040406.tar
```

*Illustration 32: md5sum of web1-logs-20040406.tar file after recovery*

Most of the log entries I used *(Timeline Evidence of a Compromise)* are taken from the off-system logs, but one file that was included in the tar-file was a copy of the "wtmp" file; the file that records logins and reboots. This file didn't have much information in it, but I was able to see reboot times recorded there using ***last*** which was useful for helping determine the timeline of activities:

```
ritchiej@aardvark:/forensics/GCFA/web1/output/recovered/var_log_backup>
last -f wtmp
agtech1  pts/0        192.168.38.25    Tue Apr  6 10:38    gone - no
logout
root     tty1                          Tue Apr  6 09:50 - 10:25
(00:34)
reboot   system boot  2.4.18-3smp      Tue Apr  6 09:35
(340+07:43)
reboot   system boot  2.4.18-3smp      Tue Apr  6 09:33
(340+07:44)
reboot   system boot  2.4.18-3         Tue Apr  6 09:30
(00:00)
reboot   system boot  2.4.18-3smp      Tue Apr  6 09:26
(00:04)
```

*Illustration 33: output of last command on recovered wtmp file*

Examination of the log files yielded several other things useful to the case as well. Within the maillog log file I found evidence of email sent from the machine as part of the compromise that included a recipient address *(Timeline Evidence of a Compromise)*.

Looking through recovered Apache logs and the off-system logs showed me that the attack came from 211.55.78.25, which, according to whois data provided by the Korean Network Information Center (KRNIC), is a Korean IP address:


*Illustration 34: information about 211.55.78.25 provided by KRNIC*

## Analysis On the Running System

In addition to examining the VM disk image and manually exporting files to my forensics workstation I also booted the VM directly to its installed operating system. Doing this allowed me to forensically examine the running compromised system and it allowed me to get information about the running programs on the system. Since I had recovered the root password from the /usr/local/games/.sniffer file where it had been captured by the intruder, logging in to the system as root was no problem. I could then use the ***Helix*** CD to examine the system using uncorrupted binaries.

Information I gathered from the running system included doing a “uname -a” to confirm the kernel version (2.4.18-3smp).

Running ***lsof*** from the ***Helix*** CD produced useful information. Note the initsk12 program running instead of /sbin/init, the /usr/sbin/kernel program running using libcrypt and listening on “hostname” (“hostname” is the /etc/services translation

for port 101), and the /bin/mktemp program with /dev/hdx1 open and a protocol socket that ***lsof*** can't identify:

```
COMMAND    PID USER   FD   TYPE     DEVICE    SIZE      NODE NAME
initsk12     1 root  cwd    DIR        3,2    4096         2 /
initsk12     1 root  txt    REG        3,2   26920    245597 /
sbin/initsk12
kernel     664 root  txt    REG        3,2  630693    376367 /
usr/sbin/kernel
kernel     664 root  mem    REG        3,2   89547    130819 /lib/ld-
2.2.5.so
kernel     664 root  mem    REG        3,2   89424    130836 /
lib/libnsl-2.2.5.so
kernel     664 root  mem    REG        3,2   23575    130830 /
lib/libcrypt-2.2.5.so
kernel     664 root  mem    REG        3,2   11174    130870 /
lib/libutil-2.2.5.so
kernel     664 root  mem    REG        3,2 1401027   3433924 /
lib/i686/libc-2.2.5.so
kernel     664 root    0u   CHR        1,3             66861 /dev/null
kernel     664 root    3u  IPv4       1046               TCP *:hostname
(LISTEN)
mktemp    1254 root  cwd    DIR        3,2    4096   3482977 /bin
mktemp    1254 root  rtd    DIR        3,2    4096         2 /
mktemp    1254 root  txt    REG        3,2    8332   3482981 /
bin/mktemp
mktemp    1254 root    5u   REG        3,2       0     70776 /dev/hdx1
mktemp    1254 root    6u  sock        0,0              3423 can't
identify protocol
```

*Illustration 35: lsof output from the running system*

The output of “ps auxwwe” tells us that the /usr/sbin/kernel process may be using /etc/kernel_config as a configuration file:

```
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root       664  0.0  0.5  1844  632 ?        S    09:05   0:00 /
usr/sbin/kernel -f /etc/kernel_config PWD=/usr/sbin
CONSOLE=/dev/console PREVLEVEL=N CONFIRM= runlevel=3 OLDPWD=/
LANG=en_US.iso885915 SHLVL=2 previous=N HOME=/ TERM=linux
PATH=/usr/sbin RUNLEVEL=3 INIT_VERSION=sysvinit-2.84 _=/usr/sbin/kernel
```

*Illustration 36: ps output from the running system*

The output of “netstat -na” confirms that something is listening on port 101:

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
State
tcp        0      0 0.0.0.0:101             0.0.0.0:*        LISTEN
```

*Illustration 37: netstat output from the running system*

In order to determine what version of OpenSSL that the apache module mod_ssl had been compiled with, I altered the ServerSignature directive to “On” within the Apache configuration file httpd.conf to display version numbers within the Apache logs, and restarted the webserver. I was then able to examine the web server log file error.log to get version information:


*Illustration 38: extract from Apache startup log*

## Analysis of RST.b

I also used the VM to study the RST.b trojan/virus on the system. I wanted to test the reported infection behavior of the trojan. In order to do this, I booted the VM from the ***Helix*** CD and mounted the VM disk from the /mnt mount point on the ***Helix*** system. I configured the network for the ***Helix***-booted system so that the default route sent network traffic via my forensics workstation, and I started ***ethereal*** on the forensics workstation to monitor all network traffic coming from the VM. I then set up a test root infection. The following screen shots illustrate this experiment:

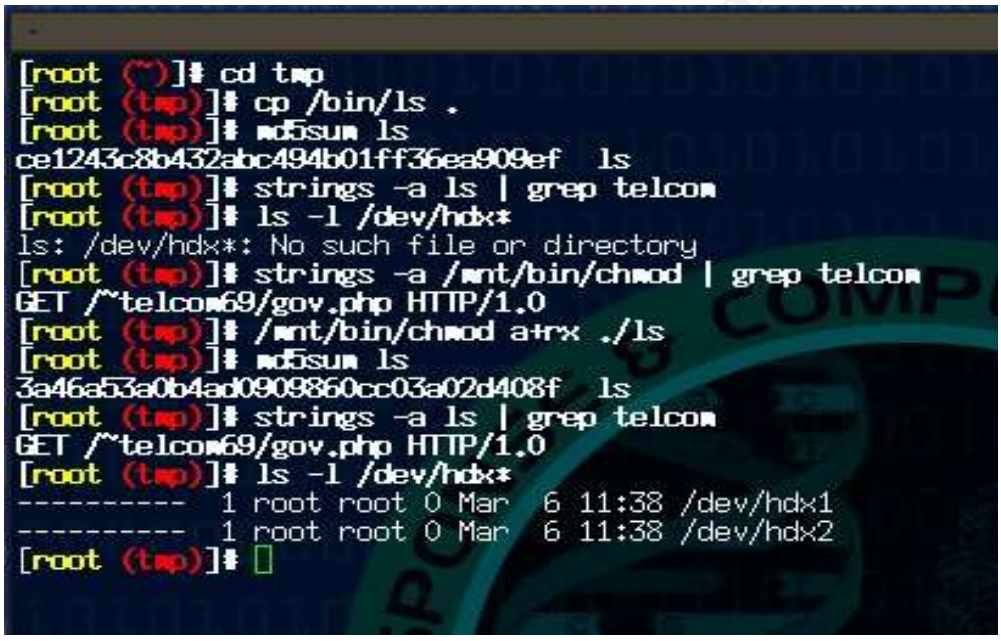



*Illustration 39: virus/trojan infection experiment*

I first copied a clean, uninfected ELF-format binary from the ***Helix*** distribution to /tmp. The ***md5sum*** and the ***strings*** commands illustrate its beginning state. I did an ***ls*** of /dev/hdx* to show that the files were not present on the running system. I did a ***strings*** command on an RST.b-infected binary on the compromised system's disk, /mnt/bin/chmod; this illustrates the presence of the "telcom69" string within that binary. I then executed, as root, the infected binary from within the directory that I'd copied the clean binary to (/tmp).

The instant I executed the /mnt/bin/chmod command, the following warning popped up from ***VMware***:


*Illustration 40: **VMware** promiscuous mode warning box*

After clicking "OK," ***ethereal*** registered network traffic from the experimental VM. Continuing on with the experiment, doing an ***md5sum*** of the test binary /tmp/ls shows that the binary is changed. Doing a ***strings*** on that binary shows that it now has the "telcom69" string within it. Doing a directory listing of /dev/hdx* showed the presence of the semaphore files /dev/hdx1 and /dev/hdx2 on the test system.

Examination of the ***ethereal*** output shows us (after an initial ARP request and reply) that the experimental machine attempted to establish an http connection with 207.66.155.21:

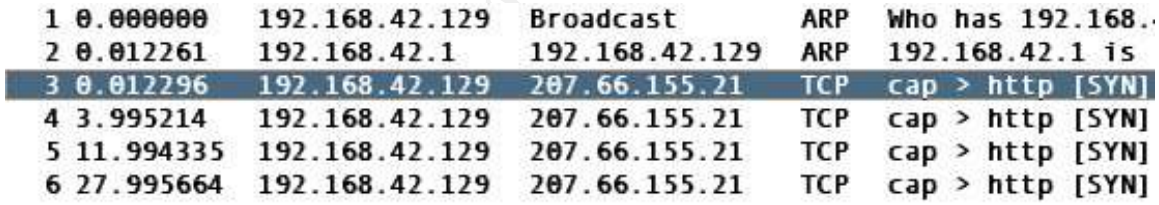



*Illustration 41: ethereal capture of http connection attempt to 207.66.155.21*

Running ***lsof*** on the experimental system showed:

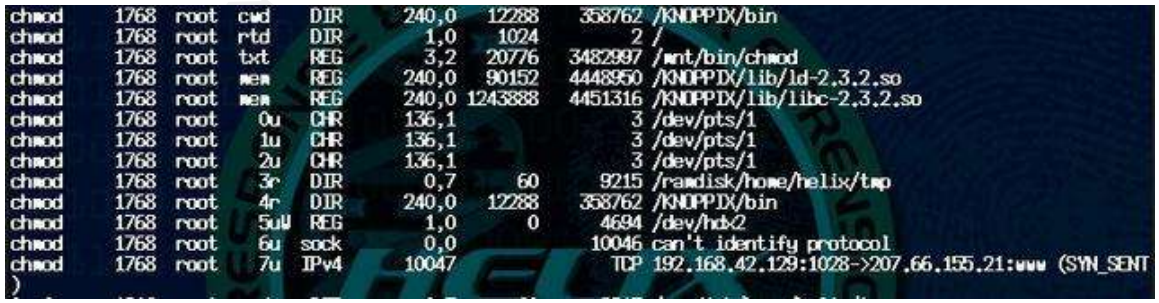



*Illustration 42: lsof output from newly-infected VM*

The /mnt/bin/chmod command had the file /dev/hdx2 open, had a socket open of an unidentifiable (by ***lsof***) protocol, and had an attempted connection to 207.66.155.21 in state SYN_SENT.

This experiment proved that the trojan present in the binaries of the compromised system would, upon initial infection of a new ELF-executable by the root user, place the semaphore files in the /dev directory, attempt to set the system network interface into promiscuous mode, and attempt to make an http connection to a remote IP address.

Exhaustive analysis of the trojan's behavior could have been performed (i.e. running ***strace*** on it, setting up a destination webserver with the correct IP to allow and capture the full TCP connection, or even disassembling the executable) but I felt that the characteristics of the trojan matched those of the RST.b trojan already analyzed and described by Ryan Russell (Russell, “RST.b”) and “lockdown” (lockdown, “RST-b commented asm dump”) closely enough to assume that they're the same. My tests did reveal the target IP address of the http connection attempt so I did some more research on that instead.

According to whois data acquired from http://www.ratite.com, the IP address 207.66.155.16 is allocated to Bondo and Remer:

**207.66.155.16**

```
Wolfe Internet Access, L.L.C. WOLFE-BLK-1 (NET-207-66-128-0-1)
                                  207.66.128.0 - 207.66.255.255
Bondo and Remer BONDO-REMER (NET-207-66-155-16-1)
                                  207.66.155.16 - 207.66.155.31
```

*Illustration 43: whois listing for 207.66.155.16*

Geobytes (geobytes, inc., “IP Address Locator tool”) locates the IP address in Seattle Washington, and provides the following map to its location:

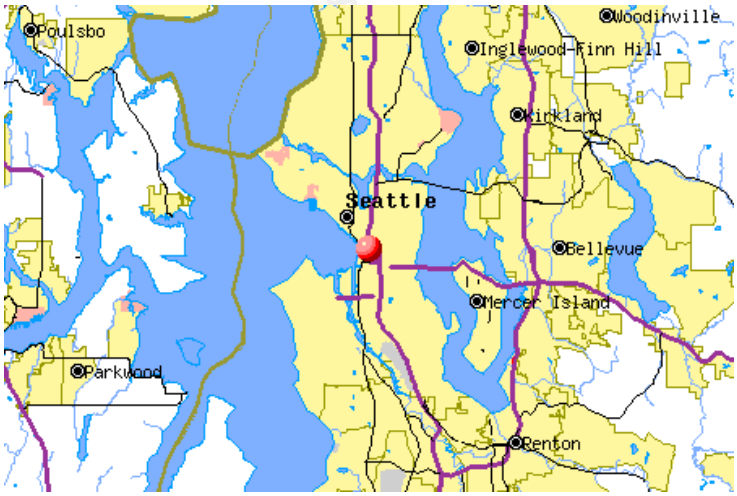



*Illustration 44: map showing location of 207.66.155.16, provided by Geobytes, Inc.*

Searching for “Bondo and Remer” on Google yielded quite a few results which confirmed that it is an advertising agency in Seattle and provided their address and telephone number. I did not pursue the information further because there is little point in contacting them for what was probably a hacked machine back when this version of RST.b was released.

## Timeline Analysis

One of the first and most important forensic tasks to perform when analyzing a potentially compromised system is to build a timeline of activity from the filesystem being analyzed. This will help pinpoint changes and activity on the filesystem.[*] Other data sources that contribute to the timeline include data from system and off-system logs and interviews conducted with people relevant to the case.

Based on interviews with Agency Tech 1, I knew that the subject machine was suspected of having been compromised on or around April 6th, 2004 and that it had not used much since that incident. I knew that it had been shut down on September 2nd, 2004 for the final time. Given those dates, it was relatively easy to browse through the timeline until I found evidence of a system compromise.

### *Timeline Evidence of a Compromise*

Analysis of the timeline shows what appears to be the beginning of an attack and root compromise of the system; starting at Tue Apr 06 2004 08:29:57 and ending at Tue Apr 06 2004 08:30:09 there are hundreds of file creations by user/group “apache/apache,” for example:

```
Tue Apr 06 2004 08:29:57      864 m.c -/-rw-r--r-- apache   apache
311562   /etc/httpd/rsawebagent/data13071.shm
```

*Illustration 45: apache file creation example*

These entries correspond with entries stored on the Agency X log server that start and end at approximately the same times:

[*] To someone familiar with the operating system and its normal operations, timeline analysis can be a relatively quick way to determine if something is abnormal on a suspect machine. This is especially true if you have some idea of what time a potential incident occurred.

```
syslog-bootlog-20040406:Apr  6 08:30:03 192.168.200.14 httpd[12090]:
[error] mod_ssl: SSL handshake failed (server web1.agencyX:443, client
211.55.78.25) (OpenSSL library error follows)
syslog-bootlog-20040406:Apr  6 08:30:03 192.168.200.14 httpd[12091]:
[error] mod_ssl: SSL handshake failed (server web1.agencyX:443, client
211.55.78.25) (OpenSSL library error follows)
syslog-bootlog-20040406:Apr  6 08:30:03 192.168.200.14 httpd[12771]:
[error] mod_ssl: SSL handshake failed (server web1.agencyX:443, client
211.55.78.25) (OpenSSL library error follows)
syslog-bootlog-20040406:Apr  6 08:30:04 192.168.200.14 httpd[12772]:
[error] OpenSSL: error:1406908F:SSL
routines:GET_CLIENT_FINISHED:connection id is different
syslog-bootlog-20040406:Apr  6 08:30:06 192.168.200.14 httpd[12092]:
[error] OpenSSL: error:0306C072:bignum routines:bn_expand2:bignum too
long
syslog-bootlog-20040406:Apr  6 08:30:07 192.168.200.14 httpd[12092]:
[error] OpenSSL: error:140BB004:SSL
routines:SSL_RSA_PRIVATE_DECRYPT:nested asn1 error
```

*Illustration 46: excerpt of OpenSSL error messages*

These OpenSSL errors can be indicative of an OpenSSL buffer overflow attack (Larrieu, p. 37), (Carrier), (Behounek). A detailed description of one OpenSSL buffer overflow attack that produces these log messages is available in Heather Larrieu's GCIH practical assignment “A J0k3r Takes Over” (Larrieu, p. 11-14+).

At the tail end of the SSL attack there are the following entries in the timeline:

```
Tue Apr 06 2004 08:31:32        0 .a. -rw-r--r-- 1001     users
2551850  <web1_hda2.img-dead-2551850>
                                0 .a. -rwxr-xr-x root     users
1488976  <web1_hda2.img-dead-1488976>
                                0 .a. -rw-r--r-- 1001     users
2551852  <web1_hda2.img-dead-2551852>
                                0 .a. -rw-r--r-- 1001     users
2551851  <web1_hda2.img-dead-2551851>
                                0 .a. -rwxr-xr-x root     users
1488974  <web1_hda2.img-dead-1488974>
                                0 .a. -rw-r--r-- 1001     users
3124170  <web1_hda2.img-dead-3124170>
                                0 .a. -rw-r--r-- 1001     users
2551847  <web1_hda2.img-dead-2551847>
                                0 .a. -rw-r--r-- 1001     users
3124171  <web1_hda2.img-dead-3124171>
                                0 .a. -rw-r--r-- 1001     users
2551853  <web1_hda2.img-dead-2551853>
                                0 .a. -rw-r--r-- 1001     users
2551848  <web1_hda2.img-dead-2551848>
etc...
```

*Illustration 47: "user 1001" timeline evidence of a tar file*

These “user 1001” files may be the remains of a tar-file package that the intruder downloaded to leverage “apache” access to gain “root” access. The fact that they are showing a userid of 1001 instead of translating to an actual username indicates that they are files that were created on another system that had a userid of 1001 on it but there's no userid of 1001 on this system. This is a typical

appearance of a ***tar*** archive built on one system and unpacked on another system.

The next entries in the timeline are very suspicious; neither of these root-owned files should be showing a create-time entry here:

```
Tue Apr 06 2004 08:31:52   260616 ..c -/-rwxr-xr-x root     root
376340   /usr/sbin/sshd
                            1192 m.c -/-rwxr-xr-x root     root
3515688  /etc/rc.d/init.d/syslog
```

*Illustration 48: suspicious file creations in timeline*

I was unable to determine why these files had updated modification and/or creation times; examination of the files didn't reveal anything out of the ordinary.

The next entries in the timeline clearly show that the intruder has root access. He or she began to compile programs and install them into system directories with root privileges:

```
Tue Apr 06 2004 08:31:55     2391 .a. -/-rw-r--r-- root     root
2126365  /usr/include/linux/file.h
                            6346 .a. -/-rw-r--r-- root     root
2126600  /usr/include/linux/proc_fs.h
                             995 .a. -/-rw-r--r-- root     root
2126715  /usr/include/linux/uio.h
                             628 .a. -/-rw-r--r-- root     root
311020   /usr/include/asm/ipcbuf.h
                              83 .a. -/-rw-r--r-- root     root
2126325  /usr/include/linux/dcache.h
                           88620 m.. -/-r-xr-xr-x root     root
3483022  /bin/ps
                            1506 .a. -/-rw-r--r-- root     root
2126599  /usr/include/linux/prefetch.h
                               0 m.c -rw-r--r-- 1001     users
3124171  <web1_hda2.img-dead-3124171>
                            3984 m.c -/-rwxr-xr-x root     root
330137   /usr/bin/vadim
```

*Illustration 49: first evidence in timeline of intruder as root: compiling the rootkit*

There are many lines of updated access times of linux C-language header files in /usr/include, mixed with updated modification times of root-owned system binaries such as /bin/ps and /usr/bin/vadim. This appears to be compilation and installation of a rootkit. Also present at this time are many “dead” inodes, which are indications of deleted files where no trace of the original filename exists. Depending upon the file permissions and ownership, one can make assumptions about what they are. If they're root-owned with execute permissions of “r-xr-xr-x” they may be compiled executables being staged for placement. If they're root-owned with permissions of “rw-r--r--” they may be source code files or temporary configuration files. If they're owned by user 1001 with only the access times updated they may be source code for the executable that's being compiled at that stage.

System files modified or new files created during this rootkit build include:

```
/bin/ps
/usr/bin/vadim (new)
/etc/sysconfig/console/default.ps (new)
/etc/sysconfig/console/default.syslog (new)
/etc/sysconfig/console/default.socklist (new)
/etc/sysconfig/console (a new directory)
/usr/bin/stealth (new)
/bin/netstat
/etc/sysconfig/console/default.netstat (new)
/usr/bin/slice (new)
/etc/sysconfig/console/default.ls (new)
/usr/bin/clean (new)
/usr/bin/chsh
/bin/socklist
/usr/bin/wp (new)
```

*Illustration 50: files modified or added in rootkit build at 08:31:55*

Refer to the *Media Analysis Performed* for an analysis of each of these files.

One can tell the new files from replacement files because new files have both the modification and creation times set at the same time in the timeline, whereas replacements for existing files only show a modification time at this point in the timeline; a few seconds later we see the creation times for system binary replacement files. An example of this is `/bin/ps` with a modification time of Tue Apr 06 2004 08:31:55 but a creation time of Tue Apr 06 2004 08:32:06. I'm unsure why that is so, but it seems to be a reliable indicator of new versus changed files during this rootkit installation.

Again placing externally-saved syslog entries into this timeline gives us another clear indicator that the intruder has achieved root access at this point:

```
syslog-maillog-20040406:Apr  6 08:32:12 192.168.200.14 sendmail[14483]:
i36FW7F14483: from=root, size=1775, class=0, nrcpts=1,
msgid=<200404061532.i36FW7F14483@web1.agencyX>, relay=root@localhost
syslog-maillog-20040406:Apr  6 08:32:58 192.168.200.14 sendmail[14487]:
i36FW7F14483: to=sflavius2002@yahoo.com, ctladdr=root (0/0),
delay=00:00:47, xdelay=00:00:46, mailer=esmtp, pri=31775,
relay=mx2.mail.yahoo.com. [64.156.215.5], dsn=2.0.0,
stat=Sent (ok dirdel)
```

*Illustration 51: syslog entries from email to sflavius2002@yahoo.com*

These are two log entries from sendmail indicating that root (ctladdr=root) sent an email of size 1775 bytes to sflavius2002@yahoo.com and that it was successfully delivered to mx2.mail.yahoo.com. This is almost certainly a notification email that the root compromise was successful with some system information. It also proves that the Agency X firewall allowed outgoing SMTP connections from web1.

The rootkit festivities continue:

```
Tue Apr 06 2004 08:34:23    26920 m.c -/-rwxr-xr-x root     root
245597   /sbin/initsk12
                                0 .a. -rwxr-xr-x root     root
245381   <web1_hda2.img-dead-245381>
                            31380 mac -/-rwxr-xr-x root     root
245598   /sbin/init
                             8192 m.c d/drwxr-xr-x root     root
245282   /sbin
                            26920 m.c -/-rwxr-xr-x root     root
245597   /etc/sysconfig/network.lock (deleted-realloc)
                            31380 mac -/-rwxr-xr-x root     root
245598   /etc/sysconfig/static-routes.lock (deleted-realloc)
                            10302 .a. -/-rw--w--w- root     root
884259   /usr/local/games/.sniffer
```

*Illustration 52: more timeline evidence of rootkit installation*

Starting at Tue Apr 06 2004 08:35:04, it appears that the attacker performed another SSL-based attack on the system. From the system timeline we see the same pattern of many apache-owned file creations:

```
Tue Apr 06 2004 08:35:05        0 .a. -/-rw-r--r-- apache   apache
311730   /etc/httpd/rsawebagent/data14517.shm (deleted)
                                0 .a. -rw-r--r-- apache   apache
311729   <web1_hda2.img-dead-311729>
                                0 .a. -rw-r--r-- apache   apache
311730   <web1_hda2.img-dead-311730>
                                0 .a. -/-rw-r--r-- apache   apache
311729   /etc/httpd/rsawebagent/data14516.shm (deleted)
etc...
```

*Illustration 53: evidence of second web attack*

These also have the accompanying off-site log entries:

```
syslog-bootlog-20040406:Apr  6 08:35:11 192.168.200.14 httpd[13079]:
[error] mod_ssl: SSL handshake timed out (client 211.55.78.25, server
web1.agencyX:443)
syslog-bootlog-20040406:Apr  6 08:35:11 192.168.200.14 httpd[13083]:
[error] mod_ssl: SSL handshake timed out (client 211.55.78.25, server
web1.agencyX:443)
etc...
```

*Illustration 54: second wave of OpenSSL attacks*

They are also followed by a lot of “dead” inodes, both root-owned and “user 1001” file:

```
Tue Apr 06 2004 08:43:13        0 m.c -rwxr-xr-x root     root
1488973  <web1_hda2.img-dead-1488973>
                                0 m.c -rwxr-xr-x root     users
1488978  <web1_hda2.img-dead-1488978>
                                0 m.c -rw-r--r-- 1001     users
2551850  <web1_hda2.img-dead-2551850>
                                0 m.c -rwxr-xr-x 1001     users
1488971  <web1_hda2.img-dead-1488971>
                                0 m.c -rw------- 1001     users
1129223  <web1_hda2.img-dead-1129223>
```

*Illustration 55: "user 1001" file modification/creation times*

This last set of “user 1001” and root file entries show the modification and creation time updates for the inodes corresponding to the entries at Tue Apr 06 2004 08:31:32 and for the rootkit build of Tue Apr 06 2004 08:31:55. Since these inodes do not point to actual files (they're “dead”) then these modification/creation times indicate that these files that were last accessed at 08:31:32 are being deleted now, at 08:43:13. The intruder may be cleaning up some of the rootkit build files and directories.

These cleanup lines are followed by more signs of root activity:

```
Tue Apr 06 2004 08:44:13        0 mac -rwxr-xr-x root     root
884260   <web1_hda2.img-dead-884260>
                                0 mac -/-rwxr-xr-x root     root
884260   /usr/local/games/sk (deleted)
                                0 mac -/-rwxr-xr-x root     root
884260   /usr/local/games/httpd (deleted)
                             4096 m.c d/drwxr-xr-x root     root
883010   /usr/local/games
Tue Apr 06 2004 08:53:13        0 m.c -/---------- root     root
70776    /dev/hdx1
                                0 m.c -/---------- root     root
70784    /dev/hdx2
```

*Illustration 56: more root activity, RST.b semaphore activity*

After the last spate of root activity, the next set of timeline entries of interest are:

```
Tue Apr 06 2004 09:30:40   375704 .a. -/-rw-r--r-- root     root
1847837  /lib/modules/2.4.18-3/kernel/drivers/atm/idt77252.o
                           198876 .a. -/-rw-r--r-- root     root
1226422  /lib/modules/2.4.18-3/kernel/drivers/addon/qla2200/qla2300.o
                            70673 .a. -/-rw-r--r-- root     root
735874   /lib/modules/2.4.18-3/kernel/drivers/addon/e1000/e1000.o
```

*Illustration 57: system boot into 2.4.18-3 kernel at 09:30:40*

These entries last for many pages and show the access times left on kernel modules when the machine was booted using the 2.4.18-3 kernel. This can be confirmed against the ***last*** output referred to in *Illustration 33*, above *(Media Analysis Performed)*. Note that the boot immediately following this reboot does not leave this type of trace in the timeline:

```
Tue Apr 06 2004 09:33:43      477 m.c -/-rw-r--r-- root     root     21
/boot/kernel.h
                               22 m.c l/lrwxrwxrwx root     root     29
/boot/System.map -> System.map-2.4.18-3smp
```

*Illustration 58: system boot into 2.4.18-3smp kernel at 09:33:43*

This is because the system was booted into the 2.4.18-3smp kernel at a later date, thus overwriting all the access times on those kernel module files. This later date was the last time the system was booted, at Wed Sep 01 2004 13:08:25, as can be seen from the timeline:

```
Wed Sep 01 2004 13:08:25    87988 .a. -/-rw-r--r-- root     root
2125782  /lib/modules/2.4.18-
3smp/kernel/drivers/addon/bcm5700/bcm5700.o
                            27870 .a. -/-rw-r--r-- root     root
1161013  /lib/modules/2.4.18-3smp/kernel/abi/sco/abi-sco.o
                            36179 .a. -/-rw-r--r-- root     root
2158478  /lib/modules/2.4.18-3smp/kernel/drivers/addon/cipe/cipcb.o
```

*Illustration 59: last system boot using 2.4.18-3smp kernel*

Likewise, the system boots seen in the ***last*** output *(Illustration 33)* at Tue Apr 06 2004 09:26, 09:33 and 09:35 are completely masked by later boot activity except for sparse timeline entries that might point to them:

```
Tue Apr 06 2004 09:26:53        0 m.c -rw-r--r-- root     root
1489016  <web1_hda2.img-dead-1489016>

Tue Apr 06 2004 09:33:43      477 m.c -/-rw-r--r-- root     root     21
/boot/kernel.h
                               22 m.c l/lrwxrwxrwx root     root     29
/boot/System.map -> System.map-2.4.18-3smp
                                0 m.c -rw-r--r-- root     root
1489014  <web1_hda2.img-dead-1489014>

Tue Apr 06 2004 09:35:08        0 m.c -rw-r--r-- root     root
1489013  <web1_hda2.img-dead-1489013>
                                0 mac -rw-r--r-- root     root
196864   <web1_hda2.img-dead-196864>
```

*Illustration 60: possible traces of 09:33 and 09:35 system boots*

## *Timeline Evidence, Post-compromise*

After the system boot at 09:30 on April 6, 2004, there is no further evidence of intruder activity on the system but there are lots of traces of Agency X investigations of the intrusion.

From 09:54 to 10:20 or so there were several login attempts and successes via SSH by Agency X personnel. The only sign of this activity in the system timeline

is the creation of a backup SSH “known_hosts” file, indicating modification of root's host key file:

```
Tue Apr 06 2004 10:22:19      665 m.c -/-rw-r--r-- root     root
1161432  /root/.ssh/known_hosts~
                             4096 m.c d/drwx------ root     root
1161421  /root/.ssh
```

*Illustration 61: timeline evidence of Agency X personnel logins*

There are lots of log entries from the syslog server covering this activity:

```
syslog-auth-20040406:Apr  6 09:54:52 192.168.200.14 sshd[14889]: Could
not reverse map address 192.168.38.25.
syslog-auth-20040406:Apr  6 09:54:54 192.168.200.14 sshd(pam_unix)
[14889]: authentication failure; logname= uid=0 euid=0 tty=NODEVssh
ruser= rhost=192.168.38.25  user=agtech1
syslog-auth-20040406:Apr  6 09:54:57 192.168.200.14 sshd[14889]: Failed
password for agtech1 from 192.168.38.25 port 3699 ssh2
syslog-auth-20040406:Apr  6 09:57:39 192.168.200.14 sshd(pam_unix)
[14889]: 1 more authentication failure; logname= uid=0 euid=0
tty=NODEVssh ruser= rhost=192.168.38.25  user=agtech1
syslog-auth-20040406:Apr  6 09:58:09 192.168.200.14 sshd[14900]: Failed
password for root from 192.168.38.25 port 3700 ssh2
syslog-auth-20040406:Apr  6 09:58:11 192.168.200.14 sshd[14900]: ROOT
LOGIN REFUSED FROM 192.168.38.25
syslog-auth-20040406:Apr  6 10:00:08 192.168.200.14 sshd(pam_unix)
[14905]: authentication failure; logname= uid=0 euid=0 tty=NODEVssh
ruser= rhost=192.168.200.3  user=root
syslog-auth-20040406:Apr  6 10:00:09 192.168.200.14 sshd[14915]:
Accepted password for agtech2 from 192.168.38.132 port 40326 ssh2
```

*Illustration 62: syslog entries of Agency X login attempts*

At 10:42:26, an Agency X administrator backed up a lot of log files into /var/log/backup, then tarred them into a tar-file and deleted them. There are hundreds of lines in the timeline like these:

```
Tue Apr 06 2004 10:42:26        0 .a. -/-rw-r--r-- root     root
3369468  /var/log/backup/dmesg.gz (deleted)
                                0 .a. -/-rw-r--r-- root     root
3369504  /var/log/backup/rpmpkgs.gz (deleted)
                                0 .a. -rw-r--r-- root     root
3369466  <web1_hda2.img-dead-3369466>
                           645120 m.c -/-rw-r--r-- root     root
3369560  /var/log/backup/web1-logs-20040406.tar
Tue Apr 06 2004 10:42:39        0 m.c -rw-r--r-- root     root
3369476  <web1_hda2.img-dead-3369476>
                                0 m.c -/-rw-r--r-- root     root
3369528  /var/log/backup/secure.7.gz (deleted)
                                0 m.c -/-rw-r--r-- root     root
3369524  /var/log/backup/secure.3.gz (deleted)
```

*Illustration 63: creation of syslog backup tar file*

At Apr 06 2004 13:03:38, root downloaded a Tripwire 2.3.7 package into Agency Tech 1's home directory and at 15:35:32 he unpacked it:

```
Tue Apr 06 2004 13:03:38  3224386 m.c -/-rwxrw-r-- agtech1  agtech1
2993354  /home/agtech1/tripwire-2.3-47.bin.tar.gz
Tue Apr 06 2004 13:04:16     4096 ..c d/drwxr-xr-x root     root
2878364  /home/agtech1/tripwire-2.3/bin
Tue Apr 06 2004 15:35:32       21 ..c -/-rwxr-xr-x root     root
2878375  /home/agtech1/tripwire-2.3/ChangeLog
                            3957 ..c -/-rwxr-xr-x root     root
3385481  /home/agtech1/tripwire-2.3/man/man8/twintro.8
                            5221 ..c -/-rwxr-xr-x root     root
2878377  /home/agtech1/tripwire-2.3/README
```

*Illustration 64: unpacking Tripwire*

The next day, April 7th, Agency Tech 1 downloaded chkrootkit-0.40, unpacked it, then compiled it as root. Here are extracts:

```
Wed Apr 07 2004 15:05:14   153600 m.c -/-rwxr-xr-x agtech1  agtech1
2993355  /home/agtech1/chkrootkit.tar
Wed Apr 07 2004 15:05:35     1323 ..c -/-r--r--r-- 1000     1000
2846166  /home/agtech1/chkrootkit-0.40/README.chklastlog
                            7191 ..c -/-r--r--r-- 1000     1000
2846168  /home/agtech1/chkrootkit-0.40/check_wtmpx.c
                             552 ..c -/-r--r--r-- 1000     1000
2846174  /home/agtech1/chkrootkit-0.40/chkrootkit.lsm

Wed Apr 07 2004 15:06:34     8886 .a. -/-rw-r--r-- root     root
2927524  /usr/include/bits/siginfo.h
                            1952 .a. -/-rw-r--r-- root     root
311059   /usr/include/asm/sigcontext.h
                            2833 .a. -/-rw-r--r-- root     root
2927522  /usr/include/bits/sigaction.h
                            3483 .a. -/-rw-r--r-- root     root
2927525  /usr/include/bits/signum.h

Wed Apr 07 2004 15:07:15     8756 m.c -/-rwxr-xr-x root     root
2846182  /home/agtech1/chkrootkit-0.40/ifpromisc
Wed Apr 07 2004 15:08:12    10764 m.c -/-rwxr-xr-x root     root
2846179  /home/agtech1/chkrootkit-0.40/chkproc
```

*Illustration 65: chkrootkit unpack and build by root*

You can tell that Agency Tech 1 downloaded the file because the chkrootkit.tar file is owned by agtech1. The unpacked files are owned by userid/group "1000/1000" which is not unusual for an unpacked tar-file. The compile is indicated by line after line of access times on system C-header files and link libraries. It can be seen that the compile was done as root because the resulting binaries (ifpromisc, chkproc, etc.) are owned by root even though they are placed in agtech1-owned directories.

At this time it appears that the ELF binaries in the `/home/agtech1/chkrootkit-0.40` directory became infected with the RST.b virus/trojan, as discussed in *Media Analysis Performed*. An indication of this is the access times on the RST.b semaphore files `/dev/hdx1` and

/dev/hdx2 that correspond to the chkrootkit build in the timeline:

```
Wed Apr 07 2004 15:08:14        0 .a. -/---------- root     root
70784    /etc/rc.d/rc5.d/S08ipchains.lock (deleted-realloc)
                                0 .a. -/---------- root     root
70784    /dev/hdx2
                                0 .a. -/---------- root     root
70776    /dev/hdx1
                                0 .a. -/---------- root     root
70776    /etc/rc.d/rc5.d/S98wine.lock (deleted-realloc)
                            20553 .a. -/-rwxr-xr-x root     root
65419    /dev/MAKEDEV
```

*Illustration 66: timeline activity on RST.b semaphore files*

At Wed Apr 07 2004 15:08:22 the file /root/chkroot.out was produced.

From Wed Apr 07 2004 15:43:48 to Wed Apr 07 2004 16:00:39 there are thousands of lines of systematically updated access times on much of the file system including /usr/sbin, /sbin, /usr/bin, /usr/lib, /lib and /usr/libexec/webmin. One possible explanation for this can be found by looking at a section of root's ".bash_history file", condensed below:

```
cd /etc
[ section removed ]
cd ..
grep ifpromisc * -r
[ section removed ]
strings wp > wp.strings
```

*Illustration 67: excerpt from root's .bash_history file showing recursive file reads from / directory*

This tells us that the root user was in the / (root) directory and did a recursive search on all files looking for files containing the string "ifpromisc." This type of recursive command opens each file and examines it, which would cause this type of systematic update of the access times on the file system.

We can correlate the .bash_history entries with the timeline because the command "strings wp > wp.strings" can be seen in the system timeline by locating the creation date of the file /bin/wp.strings:

```
Wed Apr 07 2004 18:05:39     1112 m.c -/-rw-r--r-- root     root
330140   /usr/bin/wp.strings
                                    61440 m.c d/drwxr-xr-x root
root     327041   /usr/bin
                                     6100 .a. -/-rwxr-xr-x root
root     328822   /usr/bin/wp
```

*Illustration 68: timeline location of creation of the wp.strings file found in root's .bash_history file*

This demonstrates that these recursive searches are located before Apr 07 2004 18:05:39 in the timeline.

## *Other Timeline Details*

Other facts that may be gleaned from the system timeline include the system build date, the date the system was last used and dates for software installs. Timeline evidence can also be used to deduce network firewall configuration.

The following timeline entries show the system build date:

```
Mon Dec 08 2003 04:57:47    16384 m.c d/drwx------ root     root     11
/lost+found
                                0 mac ---------- root     root     1
<web1_hda2.img-alive-1>
Mon Dec 08 2003 04:57:59        0 mac ---------- root     root     1
<web1_hda1.img-alive-1>
                            12288 m.c d/drwx------ root     root     11
/boot/lost+found
Mon Dec 08 2003 04:58:00     4096 mac d/drwxr-xr-x root     root
130817   /proc
                             4096 mac d/drwxr-xr-x root     root
32705    /boot
                             4096 mac d/drwxr-xr-x root     root
98113    /dev/pts
```

*Illustration 69: timeline entries showing system creation date*

These show that inode 1 of /dev/hda2 and the /lost+found directory were created at Mon Dec 08 2003 04:57:47, followed by inode 1 of /dev/hda1 and the /boot/lost+found directory at 04:57:59. This indicates that the initial system build started on December 8th, 2003 at 04:57:47 because this is the create date on root disk volume.*

The following excerpt from the timeline shows the installation of webmin:

```
Tue Dec 23 2003 13:03:26  7762218 m.c -/-rw------- root     root
213135   /tmp/Rsndk46c (deleted-realloc)
                          7762218 m.c -/-rw------- root     root
213135   /root/webmin-1.121-1.noarch.rpm
Tue Dec 23 2003 13:03:30        0 .a. -rwx------ root     root
2878420  <web1_hda2.img-dead-2878420>
Tue Dec 23 2003 13:03:54     4096 m.c d/drwxr-xr-x root     root
2485519  /etc/pam.d
                             1183 ..c -/-rwxr-xr-x root     root
3515872  /etc/rc.d/init.d/webmin
                               18 m.c l/lrwxrwxrwx root     root
3532848  /etc/rc.d/rc0.d/K10webmin -> /etc/init.d/webmin
                              101 ..c -/-rw-r--r-- root     root
2485902  /etc/pam.d/webmin
```

* These dates were buried about 40% of the way through the system timeline, after entries for packaged system software loaded on the system. I found them by searching for the first creation date stamp in the timeline, which was Dec 08 2003 04:47:47. A quick way to find this was by using ***less*** and entering the search expression "/^.{36}c" from the beginning of the file. This will jump to the first line that has a "c" in the 37th position on the line.

```
                                18 m.c l/lrwxrwxrwx root     root
3564971  /etc/rc.d/rc2.d/K74nscd -> /etc/init.d/webmin (deleted-
realloc)
                                18 m.c l/lrwxrwxrwx root     root
3548945  /etc/rc.d/rc1.d/K74nscd -> /etc/init.d/webmin (deleted-
realloc)
                                18 m.c l/lrwxrwxrwx root     root
3564971  /etc/rc.d/rc2.d/S99webmin -> /etc/init.d/webmin
                                18 m.c l/lrwxrwxrwx root     root
3581215  /etc/rc.d/rc3.d/S99webmin -> /etc/init.d/webmin
                                18 m.c l/lrwxrwxrwx root     root
3548945  /etc/rc.d/rc1.d/K10webmin -> /etc/init.d/webmin
                               101 ..c -/-rw-r--r-- root     root
2485902  /etc/pam.d/webmin;3fe8adab (deleted-realloc)
                                18 m.c l/lrwxrwxrwx root     root
70858    /etc/rc.d/rc5.d/S99webmin -> /etc/init.d/webmin
Tue Dec 23 2003 13:03:55     1326 ..c -/-rw-r--r-- root     root
916528   /usr/libexec/webmin/at/lang/hu
```

*Illustration 70: timeline evidence of webmin installation*

We're tipped off by the appearance of the webmin rpm file at Tue Dec 23 2003 13:03:26 but the actual install starts at 13:03:54 with the creation of the pam entries and the startup scripts in /etc/rc.d/init.d with soft links from the various rc directories. Beginning at 13:03:55 there's a long series of /usr/libexec/webmin file creations.

The system was last used on September 2nd, 2004. At Thu Sep 02 2004 11:52:21 we see the “shutdown” command being run:

```
Thu Sep 02 2004 11:52:21    14952 .a. -/-rwxr-xr-x root     root
245387   /sbin/shutdown
                                 0 mac -/-rw-r--r-- root     root
2455617  /var/run/shutdown.pid (deleted)
```

*Illustration 71: timeline evidence of final system shutdown*

The last time stamp in the timeline file is at Thu Sep 02 2004 11:52:29.

We can find lots of timeline evidence of the root user browsing the Internet in the timeline, similar to these entries:

```
Fri Jan 02 2004 11:42:46      291 m.c -/-rw------- root     root
2322927  /root/.netscape/cache/16/cache3FF5C9B600F7295.gif
                               637 m.c -/-rw------- root     root
2322915  /root/.netscape/cache/16/cache3FF5C9B60027295.js
                              1661 m.c -/-rw------- root     root
2322917  /root/.netscape/cache/16/cache3FF5C9B60047295.gif
```

*Illustration 72: timeline evidence of root web browsing activities*

Although we could reconstruct the root user's browsing activities from their web browser cache, more pertinent to our current investigation is that this web browsing activity demonstrates that the server has open access to the Internet through the Agency X firewall on port 80.

## Deleted File Recovery

Since the system under investigation used ext3 file systems, recovery of deleted files was difficult: “When EXT3FS was introduced, the behavior changed and now all links are wiped. This makes file recovery MUCH harder” (SANS Institute, p. 114). Although the link between the file name and the inode number still existed in many cases, the links between the inode and the data block on disk had all been reset on deleted files. This meant that, in some cases, I could see file names with inode numbers that I wanted to recover but the inodes all pointed to disk fragment “0” so there was no association to where the data had previously been stored on the drive.

Because the link between file name/inode and the data blocks had been reset and I couldn't correlate recovered data blocks with file names, I was also unable to determine and prove exactly when recovered files were deleted. Not knowing the file name of a recovered data block meant that I couldn't locate it on the system timeline, and not knowing which inode had pointed to a recovered data block meant that I couldn't get deletion information from the inode. The only method of determining these associations was by deduction.

I used a couple of methods to find and recover deleted files: I did string searches on unallocated disk areas, then used ***Autopsy*** to examine and recover relevant data blocks, and I used ***foremost*** to do mass file recovery from unallocated space, then attempted to identify the resulting recovered files. I'll describe each of these two methods below.

The primary method I used to recover files was using ***Autopsy,*** since I was using that tool to view most of the string search results (as described in *String Searches)*. Refer to this screen capture for the following discussion:

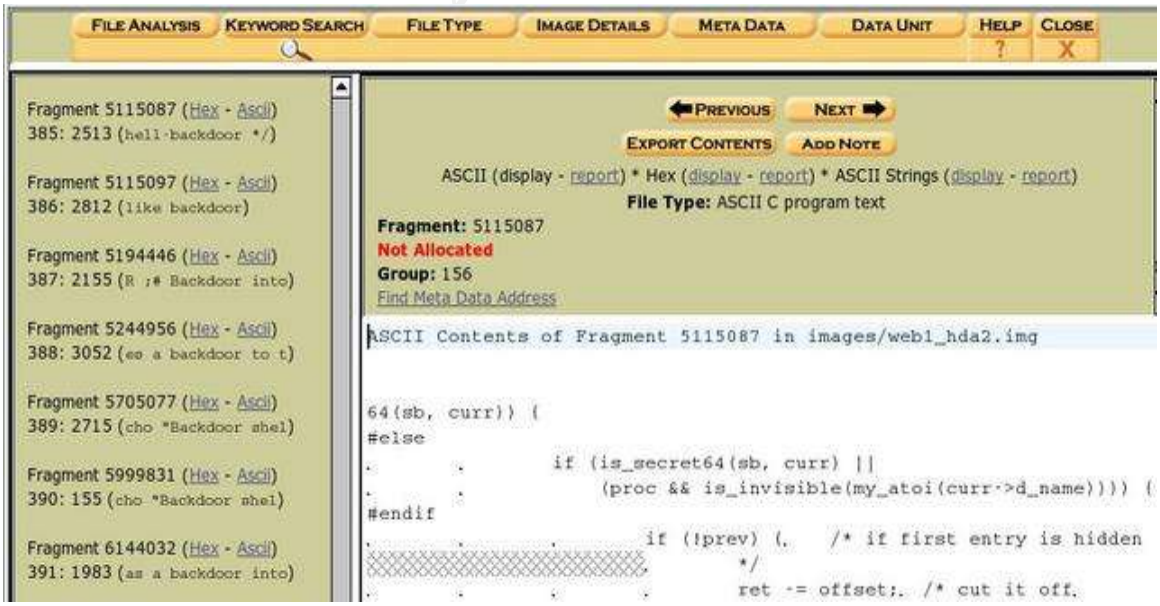



*Illustration 73: Autopsy Keyword Search screen*

Examining a string search result I selected the “Ascii” view of Fragment 5115087 and viewed that data block. Since it looked like something I wanted to recover I selected “Export Contents” and saved it off to the file `web1_hda2.img-Fragment5115087.raw`. Because this fragment appeared to occur in the middle of the file, I used the “Previous” and “Next” buttons to view preceding and following sequential data blocks, which contained more of the file. I exported each of the preceding and following blocks until I had the beginning and the end of the file, fragments 5115085 and 5115090, respectively. The beginning of the file was recognizable to someone familiar with the C programming language and the end of the file was recognizable because the text contents of the file stopped and the rest of the block (the slack space) was filled with ASCII 0 (“null”) characters.

Once I had exported all data blocks for the file I combined them into one file, (“file1.out”) using ***cat***[*]:

```
ritchiej@aardvark:/forensics/GCFA/web1/output/recovered> cat
web1_hda2.img-Fragment511508[5-9].raw web1_hda2.img-Fragment5115090.raw
> file1.out
```

*Illustration 74: concatenation of file fragments into one file using cat*

In order to remove the ASCII “null” slack space from the end of the file I opened the file with ***khexedit***, went to the end of the file and selected all the “null” characters following the last text to the end of the file and deleted them:

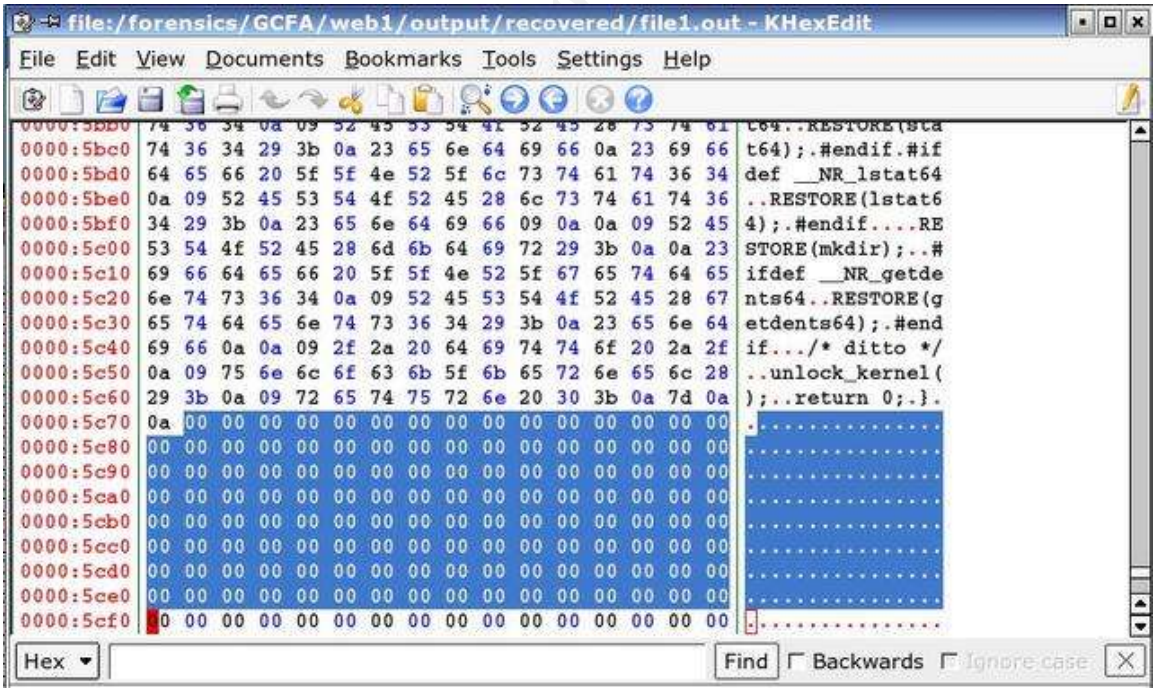



*Illustration 75: khexedit deletion of trailing null slack space*

I then saved the edited file without the trailing null characters.

[*] Because I had named the files with sequential names by block number they were reassembled in the correct order, otherwise my ***cat*** statement would have to reflect the correct order.

Using these methods, I was able to recover what appears to have been source code files for the “adore” kernel module rootkit. I found a set of consecutive unallocated file blocks (fragments 5115084 through 5115099) that, when recovered, constituted a Makefile that describes the adore package contents, 6 C source code files, two C header files identifiable as adore.h and libinvisible.h by their #ifndef and #define precompiler directives, and a shell script used to determine kernel version and modify the Makefile. Without finding and downloading an exact copy of the adore package I couldn't identify all of the files with certainty; although I can identify a C source code file I can't tell which one of several that it could be. Also, according to the Makefile, there should be seven C source code files and I was only able to find and recover six. Following is the Makefile:

```
all:    adore ava cleaner

adore: adore.c
        rm -f adore.o
        $(CC) -c -I/usr/src/linux/include $(CFLAGS) adore.c -o adore.o

ava: ava.c libinvisible.c
        $(CC) $(CFLAGS) ava.c libinvisible.c -o ava

dummy: dummy.c
        $(CC) -c -I/usr/src/linux/include $(CFLAGS) dummy.c

rename: rename.c
        $(CC) -c -I/usr/src/linux/include $(CFLAGS) rename.c

module.o: module.c
        $(CC) -c -fPIC -I/usr/src/linux/include $(CFLAGS) module.c

cleaner: cleaner.c
        $(CC) -I/usr/src/linux/include -c $(CFLAGS) cleaner.c
clean:
        rm -f core ava *.o
```

*Illustration 76: adore Makefile*

Because there is no connection remaining between the file blocks and their names or inodes I can only make deductions about when the files may have been deleted. Given that there were at least 12 files as part of the adore package and that they were almost certainly brought onto the system as part of a ***tar*** archive that had been built elsewhere and they were probably deleted all at once, one possibility is that these files were part of the “user 1001” package that was brought in as part of the system compromise (see discussion under *Timeline Evidence of a Compromise*). If that is the case, then these files were deleted at Tue Apr 06 2004 08:43:13 because that's when their inode entries' modification and creation dates are found in the timeline:

```
Tue Apr 06 2004 08:43:13        0 m.c -rwxr-xr-x root     root
1488973  <web1_hda2.img-dead-1488973>
                                0 m.c -rwxr-xr-x root     users
1488978  <web1_hda2.img-dead-1488978>
                                0 m.c -rw-r--r-- 1001     users
2551850  <web1_hda2.img-dead-2551850>
                                0 m.c -rwxr-xr-x 1001     users
1488971  <web1_hda2.img-dead-1488971>
                                0 m.c -rw------- 1001     users
1129223  <web1_hda2.img-dead-1129223>
                                0 m.c -rwxr-xr-x root     root
1129224  <web1_hda2.img-dead-1129224>
                                0 m.c -rw-r--r-- 1001     users
2551844  <web1_hda2.img-dead-2551844>
                                0 m.c -rw-r--r-- 1001     users
2551853  <web1_hda2.img-dead-2551853>
                                0 m.c -rw-r--r-- 1001     users
2551848  <web1_hda2.img-dead-2551848>
etc....
```

*Illustration 77: timeline evidence of "user 1001" file deletion*

Because the ext3 filesystem preserves the actual deletion time of the file we can gather additional information directly from the inode using ***istat***:

```
ritchiej@aardvark:/forensics/GCFA> istat -f linux-ext3 web1_hda2.img
1488973
inode: 1488973
Not Allocated
Group: 91
uid / gid: 0 / 0
mode: -rwxr-xr-x
size: 0
num of links: 0

Inode Times:
Accessed:       Tue Apr  6 08:31:55 2004
File Modified:  Tue Apr  6 08:43:13 2004
Inode Modified: Tue Apr  6 08:43:13 2004
Deleted:        Tue Apr  6 08:43:13 2004
```

*Illustration 78: istat output showing file deletion time*

Other tools I used or considered for recovery of deleted files included ***foremost*** and ***lazarus***. Although I recovered thousands of deleted files and file fragments using ***foremost***, I didn't have enough disk space to do a full recovery of unallocated space on the system because ***foremost*** is very greedy with disk space. As already discussed, it's not enough to just recover the files from unallocated space, one must also manually examine and identify them, a process that could take weeks. Since I was already locating files by using string searches against unallocated disk space I didn't feel the results of manual examination would justify the time. I decided not to use ***lazarus*** for the same reasons.

## String Searches

String searches were critical for the recovery of evidence in this investigation. I located several important deleted files through the use of keyword searching and then subsequently recovered them. String searches were also critical in locating previously undiscovered allocated files that had been installed by the intruder on the system.

I didn't perform many interactive serial string search exercises with ***Autopsy*** on my overloaded equipment before I decided to hack together some command-line tools to do searches in batch mode overnight. I wrote three tools (groupgrep.sh, mkautfromgrep.pl and filefind.pl) to facilitate batch string searches with output suitable for manual and ***Autopsy*** examination of the results. See *Appendix C* for descriptions and listings.

Early in the investigation I encountered the file /usr/local/games/.sniffer. Using ".sniffer" as a string search led me to Fragment 495501, which was allocated to inode 245598, pointed to by the file /sbin/init.

In addition to the detailed analysis of /sbin/init in *Media Analysis Performed*, the strings in /sbin/init *(Illustration 30)* gave me several pieces of information such as other pathnames to look for (/sbin/initsk12 and /usr/local/games/.rc), and additional keywords to add to my dirty word list. Keywords I added were "FUCK," "backdoor," "initsk12" and "Suckit."

Doing a string search for "backdoor" yielded 395 instances in the root disk image. Elimination of the many valid files containing the term led to the discovery and recovery of deleted source code for the "adore" rootkit. That process is described above in the *Deleted File Recovery* section.

I searched for "knoppix" in swap space to determine what impact, if any, Agency X had had on the machine by occasionally booting it from a Knoppix CD after they semi-retired it. I found 219 instances of "knoppix" on the swap partition. Although ***Helix*** is a modified version of Knoppix, it "has been modified very carefully to NOT touch the host computer in any way and it is forensically sound. Helix wil [sic] not auto mount swap space, it will also not auto mount any found devices" (e-fense, "Helix home page"). Therefore, these instances are signs of other, non-Helix, Knoppix activities.

I searched for "i36FW7F14483" (the sendmail ID) and "sflavius," *(refer to Illustration 51)* trying to locate and recover sendmail queue files for the email sent to sflavius2002@yahoo.com ostensibly notifying of root access gained. I was only able to find and recover a copy of the original syslog file containing the log entry. I also searched for "sflavius2002" on Google without result.

The following search strings were all found in some piece of evidence and added to the dirty word list in hopes of uncovering more evidence[*]:

- psybnc – first encountered in /etc/sysconfig/console/default.ls but found nowhere else.
- vadim – It turns out that Vadim is a common name. I found lots of files but no new malicious ones.
- FUCK – I didn't find anything useful but this word is more common in an open-source system than I expected.
- suckit – first encountered in /sbin/init file but found nowhere else.
- 60G – first encountered in /etc/sysconfig/console/default.syslog. Too generic of a search term and I got far too many findings.
- ettercap – first encountered in /etc/sysconfig/console/default.ls. Using this search term I discovered the altered /etc/rc.d/init.d/functions system file.
- ftpusers- - first encountered in /etc/sysconfig/console/default.ls. I found an unidentified, unallocated file fragment that matches the end of the /etc/rc.d/init.d/functions shell script.
- ixiondark – first encountered in /etc/sysconfig/console/default.syslog but found nowhere else, including on Google.

### RST.b string searches

Media examination revealed that Agency Tech 1 made private copies of the "ls," "netstat" and "ps" commands. When I did strings commands against each of these files I found identical suspect strings:

```
DOM`
/bin/sh
xxxxyyyyzzzz
Y[XXXXXX
GET /~telcom69/gov.php HTTP/1.0
ppp0
eth0
h/bin
snortdos
tory
/dev/hdx
```

*Illustration 79: strings output from ls, netstat and ps*

Many of these strings are distinctive enough to search via Google. Findings on Google led me to believe that these binaries were infected with the "Root Shell Trojan" RST.b. Details of that search and its results are provided in the *Media Analysis* section of this paper, but the portion of the tale relating to media string searches is that once I had determined that there was a possible virus infection

[*] There were many instances during the investigation where findings in one area of inquiry fueled findings in another. One example of this is the location of ".sniffer" in the timeline, which led to a string search that located /sbin/init, which yielded the dirty word "backdoor" that led to finding the adore source code.

on the system it became useful to find each and every infected file. I searched for and identified the string “telcom69” in the following system files:

```
/etc/httpd/rsawebagent/acestatus
/etc/httpd/rsawebagent/acetest
/etc/httpd/rsawebagent/generateDomainSecret
/home/agtech1/chkrootkit-0.40/chkpro
/home/agtech1/chkrootkit-0.40/chklastlog
/home/agtech1/chkrootkit-0.40/chkwtmp
/home/agtech1/chkrootkit-0.40/ifpromisc
/home/agtech1/chkrootkit-0.40/chkproc
/home/agtech1/chkrootkit-0.40/chkdirs
/home/agtech1/chkrootkit-0.40/check_wtmpx
/home/agtech1/chkrootkit-0.40/strings
/home/agtech1/ls
/home/agtech1/netstat
/home/agtech1/ps
/bin/ping
/bin/mail
/bin/mktemp
/bin/mt
/bin/hostname
/bin/ls
/bin/setserial
/bin/ed
/var/ftp/bin/cpio
/bin/cp
/bin/dd
/bin/ln
/bin/mkdir
/bin/mknod/bin/chgrp
/bin/chmod
/bin/chown
/bin/df
/var/ftp/bin/ls
/bin/mv
/bin/gawk-3.1.0
```

*Illustration 80: files containing "telcom69" string*

Some of these files are hard-links of each other (for example /var/ftp/bin/ls and /bin/ls) but knowing the extent of the trojan-infected files gives us information about the spread of the trojan and about problems that were being experienced on the system.

I also searched for “telcom69” in the system swap partition to determine if there were fragments of memory swapped out from active trojaned processes. I found six instances of “telcom69” in swap and was able to positively identify two of them. The first one appears to be /bin/mail:

```
i686./bin/mail.-s.LogWatch for
web1.agencyX.root.PWD=/.LOGWATCH_DEBUG=0.LOGWATCH_TEMP_DIR=/tmp.secure_
ip_lookup=0.named_ip_lookup=0.MAILTO=root.LOGWATCH_DATE_RANGE=yesterday
.PRINTING=.LOGNAME=root.SHLVL=3.ftpd_ignore_unmatched=0._=/bin/mail.LOG
WATCH_DETAIL_LEVEL=0.SHELL=/bin/bash.ignore_services=sshd.HOME=/.PATH=/
sbin:/bin:/usr/sbin:/usr/bin./bin/mail.....options ...]
       mail [-iInNv] -f [name]
       mail [-iInNv] [-u user]
.....................You must specify direct recipients with -s, -c, or
-b.
.........Cannot give -f and people to send to.
```

*Illustration 81: /bin/mail process in swap*

The second one appears to be /bin/awk:

```
i686.awk.-v.progname=/etc/cron.daily/0anacron.progname {
                           print progname ":\n"
                           progname="";
                         }
                         { print; }.
PWD=/.MAILTO=root.LOGNAME=root.SHLVL=2.SHELL=/bin/bash.HOME=/.PATH=/sbi
n:/bin:/usr/sbin:/usr/bin._=/bin/awk./bin/awk.....
```

*Illustration 82: /bin/awk process in swap*

No clear conclusions could be drawn about what the other four instances of "telcom69" found within swap space were.

## Conclusions

On Tuesday, April 6th, 2004, starting at 08:29:57, the web server “web1” was attacked from the Korean IP address 211.55.78.25. The attacker gained access to the system using an exploit against the server's vulnerable version (0.9.6) of OpenSSL (CERT Coordination Center). The attack quickly gained access to web1's file system and had gained root access by 08:31:55. Although there is no evidence of exactly how the intruder gained root access, web1 was running a Linux kernel version (2.4.18-3smp) that was vulnerable to the “ptrace” bug (Red Hat Network, “RHSA-2003-098”), for which exploit tools existed to gain root access from an unprivileged account (Larrieu, p. 21-24), (Carrier), (Behounek).

Once the intruder had gained root access to web1, he or she compiled and installed many rootkit tools. Tools found on the system included trojan replacements for system tools (ls, netstat, ps, init, etc.) to hide intruder activity, tools to allow backdoor SSH access to the system (/usr/sbin/kernel), and tools to monitor and alter kernel memory, allowing capturing login and password-changing information (SucKIT). The intruder also installed the RST.b virus/trojan to the system but it's not clear to what purpose; one possibility is that it was in the hope that the virus would be spread to other systems. To ensure that control could be maintained between reboots, the intruder altered system startup scripts to cause the rootkit tools to be started if the system was booted and the SucKIT kernel rootkit was trojaned as the /sbin/init file. The intruder also installed tools to attack other systems and was well placed to leverage their access on this machine to compromise other systems on Agency X' network.

Not all that the intruder did appears to have been successful. I found most of the source code for the adore kernel module rootkit but there was no trace of the compiled binaries for it. Introduction of the RST.b virus/trojan may have added to the instability of the server; the virus may have been in the process of infecting binaries in the /usr/bin directory when it was last shut down, causing the many deleted files and the unclean unmount of the system disks. Also, the intruder's activities on the system caused it to reboot several times, possibly as a result of instabilities caused by the SucKIT kernel rootkit (Farrow, p. 13). After the server rebooted, no further evidence is found of the intruder.

When the machine rebooted, it brought itself to the attention of the Agency X staff. Agency staff logged onto the machine and were able to determine that something was wrong with it. After initial investigations, they isolated the server from the network and spent a few days trying to determine exactly what had happened on it before retiring the server from service. Agency X staff acted quickly and made good decisions that stopped further intrusion into the server or its network, which the presence of passwords captured by the SucKIT kernel rootkit would have enabled if the intruder had been allowed to return.

Agency X personnel's investigations were not without impact to later forensic examination, however. Because clean, uninfected media were not used during those initial investigations, system damage was caused by further proliferation of the RST.b virus/trojan. Also, some of the intruder's activities were obscured by

the initial investigations. One clear example of this is found on the system timeline where file searching activities after the system compromise have reset MAC times to later values, masking changes that the intruder may have made. It is probable that Agency X activities also caused reallocation of inode and file sectors, overwriting data left by the intruder and making recovery of information more difficult.

This forensic investigation will have been successful if it helps teach Agency X personnel better response techniques and improves my organization's overall ability to protect itself against, and respond quickly and correctly to, a system compromise incident. It will have been doubly successful if it clearly demonstrates my understanding of the forensic investigation techniques taught in the SANS System Forensics, Investigation & Response track.

## Additional Information

I used many sources of information while investigating this incident. Here is a listing of those sources and why they were important.

http://www.google.com – Google, font of all information. I used Google searches to locate almost all of the following sources.

http://www.honeynet.org/scans/scan29/sol/wbehounek/ and http://www.sleuthkit.org/case/sotm_29/ - Two analyses of the Honeynet Project Scan of the Month #29 by Wolfgang Behounek and Brian Carrier, respectively. The case in the Scan of the Month #29 was very similar to the case I investigated and the attacker in that case used many of the same tools and techniques as the one I was investigating. I was able to learn from their investigations and they helped me draw the conclusions I did. They also both illustrate good forensic technique and reporting.

http://www.giac.org/practical/GCIH/Heather_Larrieu_GCIH.pdf – This paper, submitted by Heather Larrieu for a GCIH practical assignment, contains detailed descriptions of how a particular exploit against weaknesses in OpenSSL works and of how a "ptrace" vulnerability exploit works. It also describes some of the rootkit tools that I encountered.

http://www.cert.org/advisories/CA-2002-23.html and http://rhn.redhat.com/errata/RHSA-2003-098.html – these two pages describe the OpenSSL vulnerability and the "ptrace" kernel vulnerability that the server I was investigating was susceptible to. They include which versions of the software are vulnerable and what the fixes are.

http://www.security-focus.com/archive/100/247640 – A discussion of the RST.b Root Shell Trojan, variant "b." This includes an analysis of the RST.b virus/trojan and describes the author's attempt to contact the administrators of the system that RST.b connects to. This article includes a copy of the analysis, available at http://www.lockeddown.net/rst-expl.txt, and it refers to a GDB disassembly of the virus available at http://www.lockeddown.net/rst-variant.txt.

For tracking location of IP addresses discovered during the course of this investigation I used:

http://www.ratite.com/whois/whois.html – A global whois search tool,

http://whois.nic.or.kr/english/index.html – KRNIC, a Korean Internet Registry where I located Korea-based IP addresses,

http://www.geobytes.com/IpLocator.htm – Geobytes, Inc. IP Address Map lookup service. Given an IP address, this tool will display a map showing where it's located. According to the site, they use BGP router data and "seed data" from web browsing customer responses that locate IP addresses to specific locations, in addition to WHOIS data.

## References


Behounek, Wolfgang. “Honeynet Scan of the Month 29 Challenge.” September 2003. 3 April 2005. <http://www.honeynet.org/scans/scan29/sol/wbehounek/>.

Carrier, Brian. “Honeynet Project – Scan of the Month #29.” September/October 2003. 3 April 2005. <http://www.sleuthkit.org/case/sotm_29/>.

CERT Coordination Center. “CERT Advisory CA-2002-23 Multiple Vulnerabilities in OpenSSL.” 30 July, 2002. 3 April 2005. <http://www.cert.org/advisories/CA-2002-23.html>.

e-fense, Inc. “Chain of Custody.” 29 Oct. 2003. 8 Jan. 2005. <http://www.efense.com/Chain of Custody.pdf>.

e-fense, Inc. “Helix home page.” 2004. 3 April 2005. <http://www.efense.com/helix/index2.html>.

Farrow, Rik. “Musings.” ;login:. April 2005: 11-14.

geobytes, inc. “IP Address Locator Tool.” 22 March 2005. <http://www.geobytes.com/IpLocator.htm>.

Larrieu, Heather M. “A J0k3r Takes Over.” 7 October 2003. 1 Jan. 2005 <http://www.giac.org/practical/GCIH/Heather_Larrieu_GCIH.pdf>.

lockdown. “RST-b commented asm dump.” 19 December 2001. 11 Jan. 2005. <http://www.lockeddown.net/rst-variant.txt>.

Red Hat Network. “RHSA-2003:098-24 – Updated 2.4 kernel fixes vulnerability.” 17 March, 2003. 3 April 2005. <http://rhn.redhat.com/errata/RHSA-2003-098.html>

Russel, Ryan. “RST.b.” Security Focus FOCUS-VIRUS Archive. 28 December 2001. 3 April 2005. <http://www.security-focus.com/archive/100/247640>.

SANS Institute. Track 8 – System Forensics, Investigation & Response. Volume 8.1. SANS Press, Nov. 18, 2004.

sd and devik. “Linux on-the-fly kernel patching without LKM.” Phrack. Issue 58, 28 December 2001. 3 April 2005. <http://www.phrack.org/show.php?p=58&a=7>.

# Appendices

## *Appendix A – Forensics Letter of Agreement*

January 6, 2005

This is an agreement between the Agency Network Technical Services Unit of the Oregon Agency X and John Ritchie, Information Security Technician at the Oregon Agency Y. Agency X has agreed to loan a Dell dual 450 CPU with IDE service tag # uyq0d, Agency X inventory number X123X-27675 with Red Hat OS installed and primary installed services of Apache and RSA agent to John for the purposes of completing a forensics analysis of the system for use in his GIAC Certified Forensic Analyst Practical Assignment.

The GIAC Certified Forensic Analyst Practical assignment requires the following:
[ GCFA assignment not replicated in accordance with Administrivia ]
John agrees to the following Agency X requirements for utilizing the system for this purpose:

1. John will sign the Agency X Secrecy Agreement and abide by all state and federal laws regarding disclosure of sensitive information.
2. Any copies of the system will be either encrypted, burned or government wiped 7 times using a tool such as DD. We are open to a discussion of possible long term retention in a secured storage facility once we know more about what may be on the machine.
3. The Agency X Information Security Officer, Information Security Tech, and Disclosure Officer will all review John's paper prior to submission to GIAC.
4. John will work with Agency X' Network staff to share his findings and methods so they may learn from this process.
5. The data may be kept and utilized for up to six months beginning on January 6, 2005.

The following parties agree to these requirements.

| | |
|---|---|
| Agency X Security Officer, ISO<br>Agency X<br>January 6, 2005 | John Ritchie<br>Agency Y<br>January 6, 2005 |

## *Appendix B – Chain of Custody Form*

**ESO Case #:** SoOESO2005_01

**Page: ___1___ of __2___ pages.**

### Chain of Custody

| | | |
|---|---|---|
| Item #: SoOESO2005_01-01 | Description: Dell Precision 210 workstation, Agency X inventory # X123X-27675. | |
| Make: Dell Precision 210 | Model #: WCM | Serial #: UMG10D |

| Date/Time | Released By | Received By | Reason |
|---|---|---|---|
| Date<br>1/7/2005 | Name/Agency<br>Agency Tech 2/ Agency X | Name/Agency<br>John Ritchie / Agency Y | For the purpose of forensic analysis for GCFA practical assignment. |
| Time<br>12:20 PM | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |

Form design based on e-fense Chain of Custody form http://www.efense.com/Chain of Custody.pdf

**ESO Case #:** SoOESO2005_01

**Page: 2 of 2 pages.**

## Chain of Custody

| Item #: | Description: | |
|---|---|---|
| SoOESO2005_01-02 | Quantum Fireball lct 30.0 GB hard drive contained in Dell Precision 210 workstation (Item # SoOESO2005_01-02) labeled as WEB1.AgencyX | |
| Make: | Model #: | Serial #: |
| Quantum Fireball lct | 30.0 GB AT hard drive, Part # LB30A011 Rev 01-A | 176491530427 TAZXX |

| Date/Time | Released By | Received By | Reason |
|---|---|---|---|
| Date<br>1/7/2005 | Name/Agency<br>Agency Tech2/ Agency X | Name/Agency<br>John Ritchie / Agency Y | For the purpose of forensic analysis for GCFA practical assignment. |
| Time<br>12:20 PM | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |
| Date | Name/Agency | Name/Agency | |
| Time | Signature | Signature | |

Form design based on e-fense Chain of Custody form http://www.efense.com/Chain of Custody.pdf

## *Appendix C – Tools Created For GCFA*

These tools are “quick-n-dirty” but they helped me a lot. Using a combination of the output from filefind.pl to perform batch-mode string searches and ***Autopsy*** to keep track of and view unallocated blocks, I was able to process string searches much more efficiently than I otherwise would have been able to.

**groupgrep.sh** – takes an input file of search strings and digs them out of the disk image ASCII strings file. Uses sub-scripts to generate ***Autopsy***-ready string search results and to automate associating matching file fragments with file names.

**mkautfromgrep.pl** – converts ***grep*** results to a file format that ***Autopsy*** uses.

**filefind.pl** – uses the ***Sleuthkit*** tools ***dstat***, ***ifind***, ***ffind*** and ***istat*** to attempt to find the filename associated with a disk block number if the block is allocated to a file.

### groupgrep.sh Listing

```
#!/bin/sh
#  groupgrep.sh - given an input file of strings to search for, this
shell script
#  does a single grep for all the keywords in one pass through the
"strings" file
#  then it separates the output into individual autopsy-format files.
Calls
#  tool mkautfromgrep.pl to convert grep output into autopsy-form
results.
#  Calls tool filefind.pl to generate file information about findings.

TOOLDIR=/home/ritchiej/priv/projects/GCFA/tools

#  file containing strings to search for
INFILE=$TOOLDIR/wordsearch.in

#  gigantic strings output from disk image
ASCFILE=/forensics/GCFA/web1/output/web1_hda2.img.asc

#  where does this go; I put it into autopsy's output directory
OUTDIR=/forensics/GCFA/web1/output

# output file of all grep results
GREPFILE=$OUTDIR/grep_results.allwords

# COUNT is incremental starting point for autopsy's string search
results filename:
#  e.g. "images-web1_hda2.img-3.srch"  Set this to one bigger than the
previous highest
#  result number
COUNT=3
```

```
#  first grep for all the words so we don't have to read through the
giganti-file more than once
echo "Starting mega-grep at `date`"
grep -i -f $INFILE $ASCFILE > $GREPFILE

#  now cut the big grep results files into little ones and process them
further
for i in `cat $INFILE`
do
      echo "Starting $i at `date`"
      grep -i $i $GREPFILE > $OUTDIR/grep_results.$i
      echo "      mkautfromgrep.pl on $i (search count $COUNT) at
`date`"
      $TOOLDIR/mkautfromgrep.pl $OUTDIR/grep_results.$i $OUTDIR/images-
web1_hda2.img-${COUNT}.srch $i
      echo "      filefind.pl on $i at `date`"
      $TOOLDIR/filefind.pl $OUTDIR/images-web1_hda2.img-${COUNT}.srch >
$OUTDIR/files_containing_${i}.txt
      COUNT=`expr $COUNT + 1`
done
```

## mkautfromgrep.pl Listing

```
#!/usr/bin/perl -w

########################################################
# takes raw grep output (grepping from strings file)
#  and builds an autopsy search file out of it
####################

use strict;

#  blocksize (bytes/block)
my $b_size = 4096;

unless ((scalar @ARGV) == 3)
{
      print "Requires 3 args: infile, outfile, searchterm\n";
      exit;
}

unless (open INFILE, "$ARGV[0]")
{
      die "Can't open $ARGV[0] for reading: $!";
}

unless (open OUTFILE, ">$ARGV[1]")
{
      close INFILE;
      die "Can't open $ARGV[1] for writing: $!";
}

my $searchterm = $ARGV[2];

my @infile = (<INFILE>);
close INFILE;
```

```
my $lines = scalar @infile;

print OUTFILE "$lines||$searchterm|ascii\n";

foreach (@infile)
{
      /^\s*(\d+) (.*)/;

      my $byte = $1;
      my $string = $2;

      #  get only up to 5 characters on each side of search term
      $string =~ s/.*?(.{0,5}$searchterm.{0,5}).*/$1/i;
      my $offset = $byte % $b_size;

      my $block = ($byte - $offset) / $b_size;
      print OUTFILE "$block|$offset|$string\n";
}

close OUTFILE;
```

## filefind.pl Listing

```
#!/usr/bin/perl -w
############################################################
# perl script to automate finding filename from block number.
#  input is an autopsy-generated textfile that shows block number
#  of matching string.
#####

use strict;

#  directory where sleuthkit lives
my $sk_dir = '/usr/local/bin/sleuthkit/bin';
#  image that we're digging through
my $image = '/forensics/GCFA/web1/images/web1_hda2.img';

#  read textfile in as file parameter
my @results = (<>);

# get rid of the first one
shift @results;

chomp @results; # get rid of newlines

my $old_b_num = 0;
foreach (@results)
{
      # get block number of result
      my $b_num = (split /\|/, $_)[0];
      next if ($b_num == $old_b_num);
      $old_b_num = $b_num;
      #  now get the stats for this fragment
      my @dstat = `$sk_dir/dstat -f linux-ext3 $image $b_num`;
```

```
        if (grep /^Allocated/, @dstat)
        {
              #  get inode number from block number
              my @ifind = `$sk_dir/ifind -f linux-ext3 -d $b_num $image`;
              chomp @ifind;
              #  get filename from inode number
              my @ffind = `$sk_dir/ffind -f linux-ext3 -a $image $ifind[0]
`;
              chomp @ffind;
              print "Block $b_num is file: $ffind[0]\n";
              my @istat = `$sk_dir/istat -f linux-ext3 $image $ifind[0]`;
              chomp @istat;
              foreach (@istat)
              {
                    print "\t$_\n";
              }
        }
        else
        {
              print "$b_num not allocated\n";
        }

}
```

Last Updated: February 25th, 2020

# Upcoming SANS Forensics Training


| | | | |
|---|---|---|---|
| SANS Northern VA - Reston Spring 2020 | Reston, VA | Mar 02, 2020 - Mar 07, 2020 | Live Event |
| SANS Secure Japan 2020 | Tokyo, Japan | Mar 02, 2020 - Mar 14, 2020 | Live Event |
| SANS Munich March 2020 | Munich, Germany | Mar 02, 2020 - Mar 07, 2020 | Live Event |
| SANS St. Louis 2020 | St. Louis, MO | Mar 08, 2020 - Mar 13, 2020 | Live Event |
| SANS Dallas 2020 | Dallas, TX | Mar 09, 2020 - Mar 14, 2020 | Live Event |
| Dallas 2020 - FOR500: Windows Forensic Analysis | Dallas, TX | Mar 09, 2020 - Mar 14, 2020 | vLive |
| SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques | FOR610 - 202003, | Mar 09, 2020 - Apr 22, 2020 | vLive |
| SANS Paris March 2020 | Paris, France | Mar 09, 2020 - Mar 14, 2020 | Live Event |
| SANS Secure Singapore 2020 | Singapore, Singapore | Mar 16, 2020 - Mar 28, 2020 | Live Event |
| SANS London March 2020 | London, United Kingdom | Mar 16, 2020 - Mar 21, 2020 | Live Event |
| SANS Norfolk 2020 | Norfolk, VA | Mar 16, 2020 - Mar 21, 2020 | Live Event |
| SANS San Francisco Spring 2020 | San Francisco, CA | Mar 16, 2020 - Mar 27, 2020 | Live Event |
| SANS Oslo March 2020 | Oslo, Norway | Mar 23, 2020 - Mar 28, 2020 | Live Event |
| SANS Seattle Spring 2020 | Seattle, WA | Mar 23, 2020 - Mar 28, 2020 | Live Event |
| SANS Madrid March 2020 | Madrid, Spain | Mar 23, 2020 - Mar 28, 2020 | Live Event |
| SANS Secure Canberra 2020 | Canberra, Australia | Mar 23, 2020 - Mar 28, 2020 | Live Event |
| Mentor Session - FOR508 | Sao Paulo, Brazil | Mar 25, 2020 - Mar 28, 2020 | Mentor |
| SANS Abu Dhabi March 2020 | Abu Dhabi, United Arab Emirates | Mar 28, 2020 - Apr 02, 2020 | Live Event |
| SANS FOR585 Rome March 2020 (In Italian) | Rome, Italy | Mar 30, 2020 - Apr 04, 2020 | Live Event |
| SANS Frankfurt March 2020 | Frankfurt, Germany | Mar 30, 2020 - Apr 04, 2020 | Live Event |
| SANS vLive - FOR508: Advanced Incident Response, Threat Hunting, and Digital Forensics | FOR508 - 202003, | Mar 31, 2020 - May 07, 2020 | vLive |
| SANS 2020 | Orlando, FL | Apr 03, 2020 - Apr 10, 2020 | Live Event |
| SANS Riyadh April 2020 | Riyadh, Kingdom Of Saudi Arabia | Apr 04, 2020 - Apr 16, 2020 | Live Event |
| SANS Bethesda 2020 | Bethesda, MD | Apr 14, 2020 - Apr 19, 2020 | Live Event |
| SANS Minneapolis 2020 | Minneapolis, MN | Apr 14, 2020 - Apr 19, 2020 | Live Event |
| SANS London April 2020 | London, United Kingdom | Apr 20, 2020 - Apr 25, 2020 | Live Event |
| SANS Brussels April 2020 | Brussels, Belgium | Apr 20, 2020 - Apr 25, 2020 | Live Event |
| SANS Baltimore Spring 2020 | Baltimore, MD | Apr 27, 2020 - May 02, 2020 | Live Event |
| SANS Bucharest May 2020 | Bucharest, Romania | May 04, 2020 - May 09, 2020 | Live Event |
| SANS Security West 2020 | San Diego, CA | May 06, 2020 - May 13, 2020 | Live Event |
| Security West 2020 - FOR572: Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response | San Diego, CA | May 08, 2020 - May 13, 2020 | vLive |