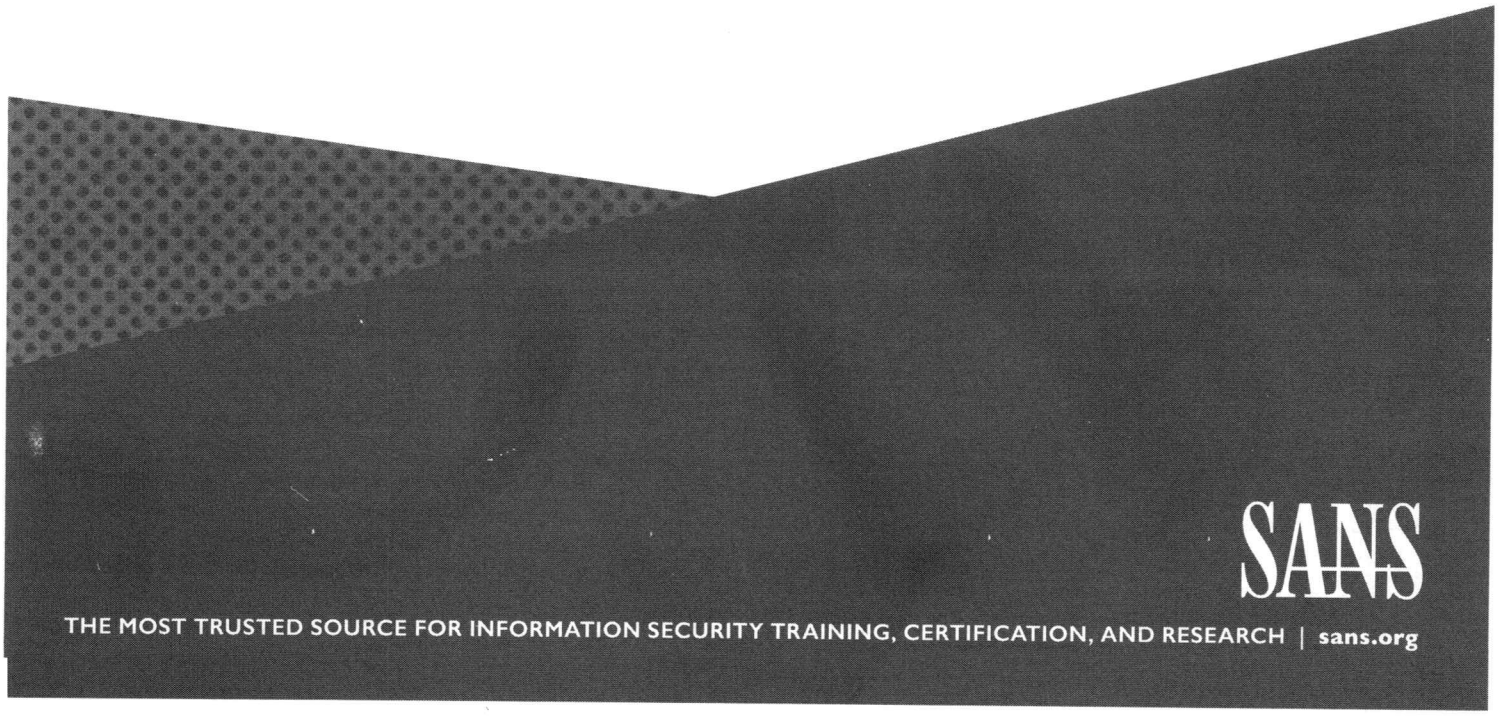
506.6
Digital Forensics for Linux/Unix



SANS

Copyright © 2017, The SANS Institute. All rights reserved. The entire contents of this publication are the property of the SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

SEC506.6 | Securing Linux/Unix

SANS

Digital Forensics for Linux/Unix

© 2017 Hal Pomeranz and Deer Run Associates | All rights reserved | Version C02_01

Digital Forensics for Linux/Unix

Welcome to the world of forensics. There's a tremendous amount of information that comprises this field, and we'll do our best to give you a comprehensive overview of the digital forensic specialty as we progress through this material.

Get ready, because of the tremendous amount that we have to cover, we are going to move along at a very high rate of speed.

Hal Pomeranz
hal@deer-run.com
http://www.deer-run.com/~hal/
@hal_pomeranz on Twitter

This course derived from materials originally created by John Green, used with permission.

Course Outline

- File system basics
- Introduction to forensics
- What can be done to prepare?
- Importance of validating compromise
- Process for collecting evidence
- Creating images and proving integrity
- Analyzing what you've collected
- Reporting and wrap-up

Course Outline

At a very high level, these are the items that we'll be covering today.

© 2017 Hal Pomeranz and Deer Run Associates

Linux File System Basics

Before we dive into forensic investigations, let's take a moment to ensure we understand the layout and structures of a file system.

Linux File System Basics

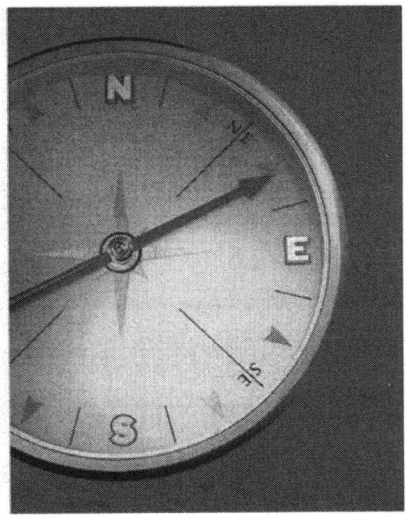
Much of your success investigating the compromise of a Linux or Unix system will depend on your understanding of the file system under investigation. File systems are quite complex, and every single one of them is different in terms of structure, features, and performance. That said, the "standard" Unix file systems (such as EXT under Linux, UFS in Solaris, FFS for the BSD OSes, etc.) have a common ancestor—the Fast File System designed by Kirk McKusick for 4.2BSD. Thus, these file systems have certain common features that can be exploited by specialized forensic tools.

So, before we dive into the depths of forensic investigations, let's take a moment to ensure an understanding of the layout and structures that are common to most Unix-based file systems.

© 2017 Hal Pomeranz and Deer Run Associates

File System Objectives

- File system concepts
- Data organization
- Conceptual layers
- Critical data structures
- Sleuth Kit tools



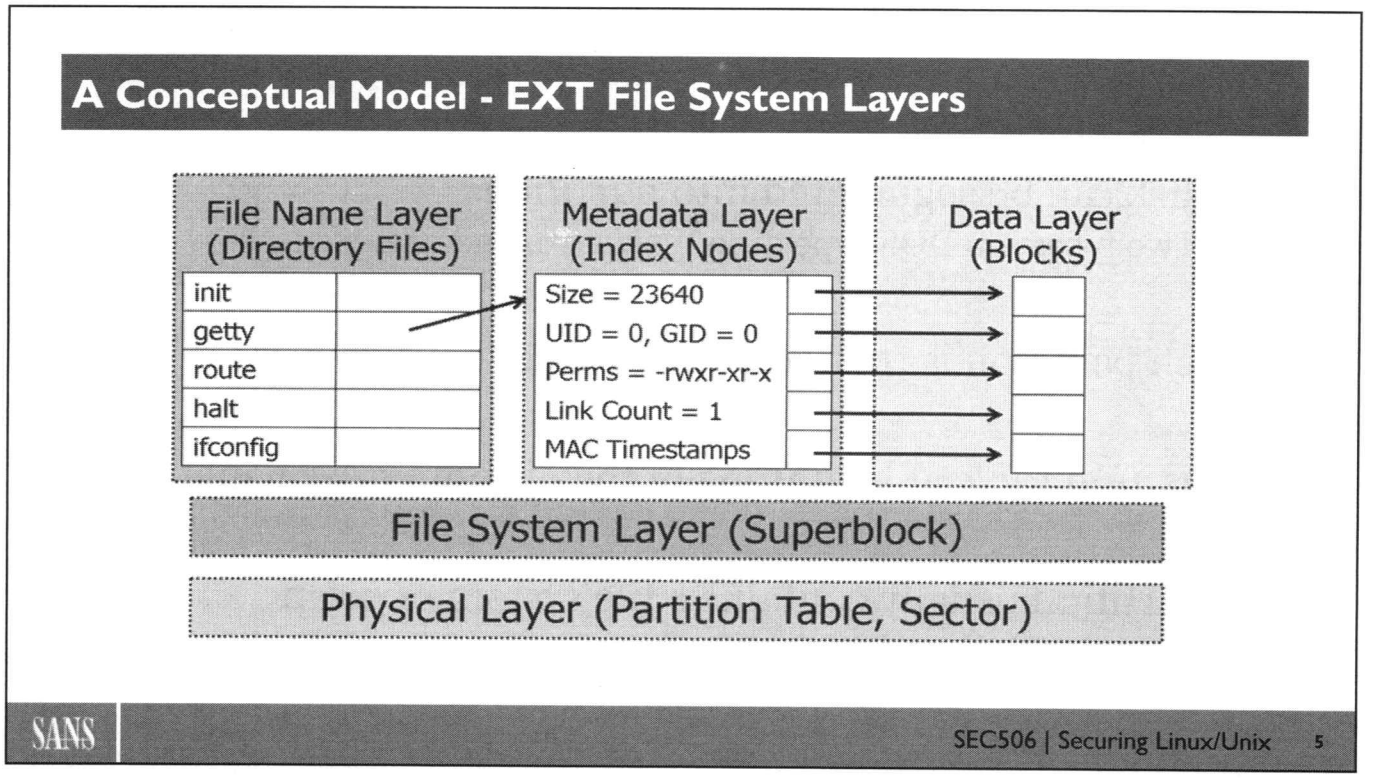
SANS | SEC506 | Securing Linux/Unix 4

File System Objectives

The file system is the medium in which compromises reside. In order to perform a detailed investigation of a system, it is important for the investigator to have a solid understanding of the file system.

This section will introduce the student to the basic concepts of Linux-based file systems. The file system and its data are arranged in layers, and a few important data structures actually tie it all together. The student will be introduced to those data structures, why they're important, and some interesting tools that can be used to analyze them.

© 2017 Hal Pomeranz and Deer Run Associates



A Conceptual Model - EXT File System Layers

You're probably familiar with the "seven layer" OSI model for describing network communications. The file systems can be conceptualized as a "five layer" model:

- Physical Layer: The physical drive or device and the partitions on it. Partition geometry is described by a *partition table* at the beginning of the disk—sometimes referred to as a *Volume Table of Contents* (VToC) or *disk label*. *Sectors* are the smallest unit of storage addressable by the disk controller.
- File System Layer: Contains all the configuration and management data associated with the file systems in each partition on the disk. For Unix file systems, the primary structure of interest at this layer is an object called a *superblock*.
- The File Name Layer (AKA Human Interface Layer) is responsible for mapping human-readable file names to metadata addresses. In Unix file systems, this is accomplished with special *directory files* that map file names to *index node* (*inode*) numbers in the layer below.
- Metadata Layer: Contains all of the data structures that are responsible for the definition and delineation of files. In Unix file systems, we use objects called *index nodes* (*inodes* for short) that store meta-data about files and pointers to the disk blocks that make up the contents of the file
- Data Layer: Contains the actual data units of disk storage—commonly referred to as *blocks* in Unix file systems.

© 2017 Hal Pomeranz and Deer Run Associates

Physical Layer: Disk Partitions

- A disk can be segmented into *partitions*
  - Two types (on DOS-type disks): *primary* and *extended*
- Each partition is treated as an independent device
- *Partition table* at beginning of the disk provides a map
- A partition usually contains a file system or swap

The Physical Layer: Disk Partitions

The physical layer consists of the physical disk device and the structures that define it.

A disk drive (with a file system) must contain at least one partition, though it may be segmented into many. The first sector of each disk contains a *partition table* or *Volume Table of Contents* (VToC, also known as a disk label). On many PC-BIOS systems, the partition information is stored in the *Master Boot Record* (*MBR*) in the first sector of the disk. The MBR only has space to store information about four partitions. These are what is referred to as the *primary* partitions on the device. Obviously, four partitions aren't enough for modern systems, so a primary partition can be subdivided into multiple partitions—this makes the partition an "extended" partition. The extended partition contains a linked list of partition tables that may describe any number of sub-partitions.

GPT (GUID Partition Tables) is an alternate disk partitioning scheme designed to overcome many of the limitations of traditional MBR-style partition tables. GPT allows for more and larger partitions and has other advanced features such as a backup partition table and error-detecting checksums.

Even though multiple partitions may exist on the same disk, the Unix operating system treats them as independent devices and performs file I/O via individual entries in the `/dev` directory.

Typically, each partition is formatted with a file system like EXT. Sometimes a file system is not formatted—for example, Unix swap partitions will typically use "raw" partitions, and some databases use raw partitions to try to improve performance.

© 2017 Hal Pomeranz and Deer Run Associates

File System Layer: Superblock

The File System Layer contains data that describes the file system within a partition

Unix uses a *superblock* which contains the following data:

- FS type/size, block size, number of blocks/inodes, etc.
- Modification time, last mounted on, clean/dirty status
- Pointer to inode for file system journal (EXT3 and above)

The File System Layer: Superblock

When a file system is created in a logical partition, a data structure is created at the beginning of the partition to define the attributes of the file system that resides there. For Unix file systems, this data structure is called a *superblock*. The superblock contains basic file system information including items like the file system type, block size, the number of blocks and inodes in the file system, the number of unallocated blocks and inodes, and so on. It also contains information about the usage of the file system, like when it was last mounted, where it was mounted, whether it was unmounted cleanly, and so on. The superblock is always replicated, to provide fault tolerance against disk failure in the first superblock.

For EXT3 and later file systems, the superblock contains a pointer to the inode of the file system journal, though this is almost always inode number 8. The superblock *does not* contain a pointer to the inode of the root of the file system, but by convention inode 2 is reserved for the root directory (in older versions of the Unix file system, the "file" at inode number 1 was used to store bad block addresses).

© 2017 Hal Pomeranz and Deer Run Associates

Data Layer: Blocks

Data Layer is where the zeros and ones are actually written

The basic storage unit is a *block*

- Blocks are composed of sectors (usually 8 in EXT)
- This is the smallest unit of file I/O in the file system

For efficiency, the blocks that make up a file are allocated consecutively when possible

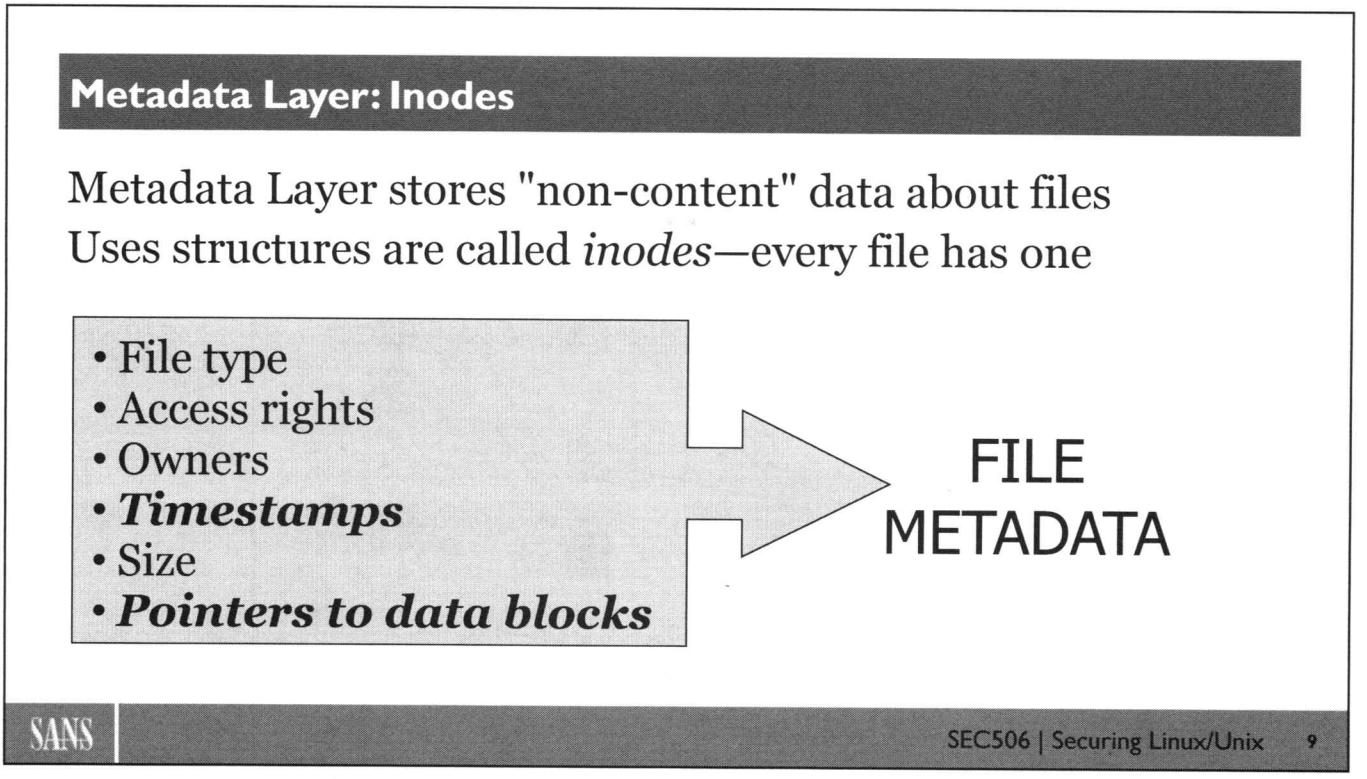
The Data Layer: Blocks

The data layer is where the binary information is actually stored on disk. The smallest storage unit addressable by the disk device is a *sector* which is usually 512 bytes. However, to improve I/O performance, EXT file systems will normally perform reads/writes in 4K chunks called *blocks*.

On modern Linux file systems, the standard 4K block size is the minimum amount of data that will be allocated to any file (if the file is smaller than 4K, the remainder of the block is wasted). The original BSD FFS implementations, in an effort to use disk space more effectively, sub-divided the 4K block into 4 1K *fragments*. Each fragment could then independently store a small file. But as disk space got cheaper, it wasn't worth the performance overhead to pack multiple small files into blocks. So, on modern Linux systems, you'll typically see the fragment size set to the same value as the block size—thus you can only have a single file per block.

When writing a large file that spans multiple blocks, the file system will tend to allocate consecutive blocks where possible. This will increase the read efficiency because the file system can "read ahead" in large swaths. But this tendency also turns out to be useful when we're trying to recover deleted data. If you can locate a suspicious string in the middle of a "deleted" block of data, you may be able to recover the entire deleted file by capturing the blocks immediately before and after the "interesting" block.

© 2017 Hal Pomeranz and Deer Run Associates



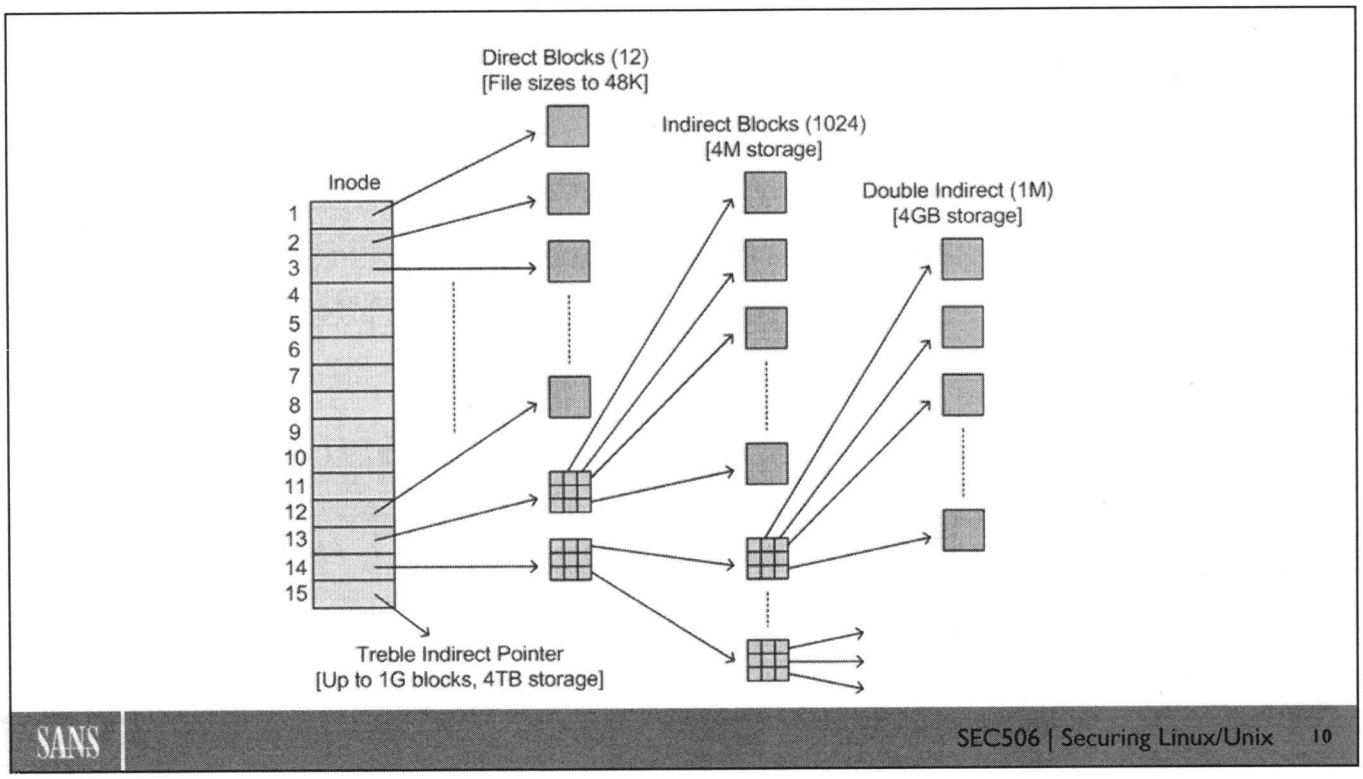
The Metadata Layer: Inodes

All file systems have structures that are used to describe or represent files. The metadata layer contains these structures. They are called different things in different file systems, but in the Unix world, they are known as *inodes* (which is a contraction of "index nodes").

An inode contains descriptive information such as timestamps, access controls or permissions, file owner's user id, and the file's size—everything you're used to seeing in the output of "`ls -l`" *except* for the file name. An inode also has pointers to the data blocks that make up the contents of the file.

Each inode has an address—they are simply numbered sequentially. Although inodes are typically hidden from the end-user, their addresses can be accessed ("`ls -i`") and a wealth of information can be retrieved using some specialized tools we'll be looking at shortly.

© 2017 Hal Pomeranz and Deer Run Associates



Indirect Block Pointers

While the inode has pointers to the data blocks that hold the file contents, the inode data structure is of fixed size (128K for EXT3 and earlier) and only has space for 15 block pointers. Assuming 4K data blocks, this would only allow you to have at most a 60K file—obviously not nearly a sufficient maximum file size.

Indirect block pointers allow the file system to use regular data blocks to be used as additional storage for block pointers rather than file data. This works as follows in the typical Unix file system:

- The first 12 block pointers in the inode point directly to actual data blocks. This means you can have files of up to 48K just using the direct block pointers in the inode.
- The thirteenth pointer in the inode is the *indirect block pointer*. It points to a block which contains block pointers to the indirect data blocks. You can put 1024 4-byte block pointers in a 4K block, each of which points to a 4K data block. Using the indirect block pointer plus the direct block pointers in the inode, you can address files of up to 4MB + 48K in size.
- The fourteenth pointer in the inode is the *double indirect block pointer*—it points to a data block that points to up to 1K additional blocks that in turn each point to up to 1K data blocks. This would allow you to address up to 4GB of additional storage.
- The fifteenth pointer in the inode is the *treble indirect pointer*, which gives you the ability to address another 4TB of storage.

So, you would think that 4T+4G+4M+48K is the maximum file size in a typical Unix file system. However, 32-bit partition size fields in the MBR limit the maximum size of a standard partition to 2TB. Also, some older Unix file systems only used 32-bit values for the file size field in the inode and so were limited to 4GB file sizes. EXT file systems use 64-bit file sizes and so do not have this limitation.

© 2017 Hal Pomeranz and Deer Run Associates

What's New in EXT4?

- 48-bit address space
- Extents instead of indirect block chains
- 64-bit nanosecond resolution timestamps
- File creation time timestamp ("btime")

What's New in EXT4?

EXT4 has actually done away with this (fairly inefficient) indirect block strategy and instead uses *extents*—defining files as one or more runs of consecutive blocks. EXT4 also uses 48-bit block addresses so you can have much larger files and filesystems.

In EXT3 and earlier, timestamps are 32-bit values which track the number of seconds since Jan 1, 1970 (the start of the *Unix epoch*). EXT4 has moved to 64-bit timestamps with nanosecond resolution. EXT4 also adds a file creation timestamp, which earlier Unix file systems have not had.

© 2017 Hal Pomeranz and Deer Run Associates

Backwards Compatibility

Backwards compatibility was a design goal

Inodes expanded to 256 bytes:

- Much of first 128 bytes like EXT[23] ...
- ... except that block pointers replaced by extents
- Extended timestamps, etc. in upper 128 bytes

Backwards Compatibility

The EXT4 developers tried very hard to maintain backward compatibility with earlier EXT file systems. This leads to some strange inode configurations. First, the EXT4 developers doubled the size of the inode, preserving much of the lower 128 bytes in the same configuration as the EXT3 inode. The one major change is the 60 bytes of the inode that once held block pointers now hold extent information. The new 128 bytes in the upper part of the inode hold additional data added by EXT4—like the additional bytes for longer timestamps, etc.

For thorough coverage see: *http://computer-forensics.sans.org/blog/tags/ext4*

© 2017 Hal Pomeranz and Deer Run Associates

Human Interface Layer: File Names

- Partition (major/minor device number) and inode number are how the kernel tracks files
- Humans don't like accessing files via sequences of numbers
- *Directory files* associate file names with inode numbers

The Human Interface Layer: File Names

The Unix operating system tracks files using the inode number and the device numbers associated with the disk partition that holds the file. But human beings don't find a series of numbers convenient for naming files, so some interface layer between humans and machines is necessary.

The Human Interface (or File Name) Layer contains special file system objects whose purpose is to associate human-readable file names with the inode numbers used by the OS. The "special objects" are what we call directories…

© 2017 Hal Pomeranz and Deer Run Associates

Directories

Inode contains all of the information about a file EXCEPT its name

Directories are special files that map inode numbers to file names

- Inode number
- File name
- File name length
- Size of entry

Byte Offset in Directory	Inode Number	File Name
0	84	.
16	6	..
32	1854	fdisk
48	1232	fsck
64	87	halt
80	1789	lsmod
96	1523	mkfs
112	74	mount
128	1466	reboot
144	132	umount
160	90	syslogd
176	1656	ifconfig

Directories

Directories are simply special files that associate file names with inodes. In the traditional Unix file systems, a directory "file" is just a sequential list of file names along with their corresponding inode. When you list a directory, you are basically just dumping the contents of the directory "file".

Directories give the file system its hierarchical structure. Consider what happens in the operating system when you try to access a file like `/home/hal/.profile`:

- Remember that inode 2 is reserved for the root of the file system, so the file system driver begins by opening this "file"
- The OS reads the contents of the root directory "file" pointed to by this inode until it finds the entry for "`home`" and the associated inode with this entry
- The OS then opens the inode from the "`home`" directory entry—this is another directory "file" and the OS scans through the contents until it finds the "`hal`" entry and its inode
- Now we have the inode for `/home/hal`—yet another directory, so the OS has to scan through the directory to find the entry for "`.profile`"
- Finally, the OS has the inode for `/home/hal/.profile` so it can open this file and read its contents

© 2017 Hal Pomeranz and Deer Run Associates

The Sleuth Kit (TSK)

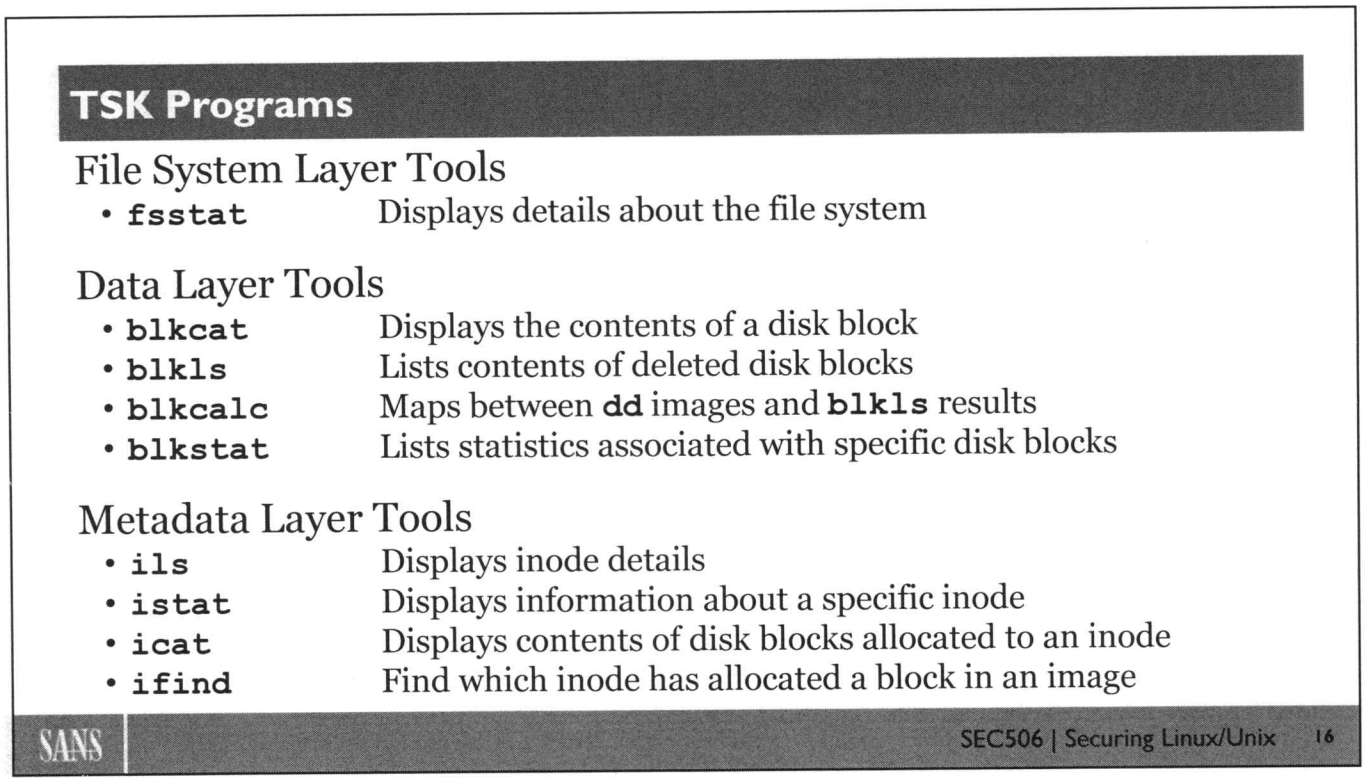
Collection of 16 highly specialized file system analysis tools

- Enhances collection/analysis tools from earlier packages
- Tools are organized by file system layers
- Follow common syntax and naming convention

The Sleuth Kit (TSK)

The Sleuth Kit was written by Brian Carrier and has undergone a bit of an evolution. Brian originally wrote a set of tools called TCTUTILS to expand the capabilities of an older package called The Coroner's Toolkit (TCT). However, over time TCTUTILS diverged from TCT and became a stand-alone package. The Sleuth Kit (TSK) was born.

© 2017 Hal Pomeranz and Deer Run Associates



TSK Programs

Having to learn so many different tools sounds daunting, but the file system tools are organized by file system layers according to the conceptual model that we discussed. The prefix letter(s) of each tool is based on the layer name. The remainder of the name uses standard names, such as `find`, `stat`, and `ls`.

The `cat` tools display data (like the `cat` utility in Unix). The `stat` tools display statistics or information about an address, the `ls` tools list details about many addresses, and the `find` tools map between different layers.

The tools for the File System Layer start with '`fs`'. The tools for the content (data) layer start with '`blk`' (in older versions of TSK these tools were prefixed with '`d`'). The tools for the metadata layer start with '`i`' because the original TCT tools were designed only for Unix inodes.

© 2017 Hal Pomeranz and Deer Run Associates

More TSK Programs

Human Interface (File Name) Layer

- **`fls`** Displays file and directory entries in a directory inode
- **`ffind`** Determine which file has allocated an inode in an image

Media Management (partitions)

- **`mmls`** Displays list of partitions in a disk image

Other miscellaneous tools:

Hash database tools, timeline tools, etc.

More TSK Programs

The Human Interface layer commands all start with the letter '`f`'. The `fls` tool allows us to list files like the `ls` command does in Unix, but has multiple additional features, including showing deleted files.

The `mmls` tool can be used to read the partition map from a bit image of an entire disk.

The remaining six tools do miscellaneous support options. The `file` tool is analogous to the Unix `file` command, determining the type of an unknown file. The `sorter` command will process files and sort them into bins of similar type. The `mactime` tool will take a specially formatted file and create a nicely formatted timeline of file modification, access, and attribute change, and finally, the hash database tools permit the comparison of the files in an image against the hashes of known good files in an attempt to reduce the number of unknown files that need to potentially be looked at.

© 2017 Hal Pomeranz and Deer Run Associates

A Layer / TSK Case Study

String search is one way of locating evidence

How do you find the file that a "hit" belongs to?

1. Use **`grep`** with a byte offset flag
2. Divide byte offset by block size to compute block number
3. Display a hexdump of block to see the string in context
4. Check if the block is allocated to a file
5. If it is, see which inode number has that block allocated
6. Search directories to find file name associated with inode number

TSK can help, let's walk through it!

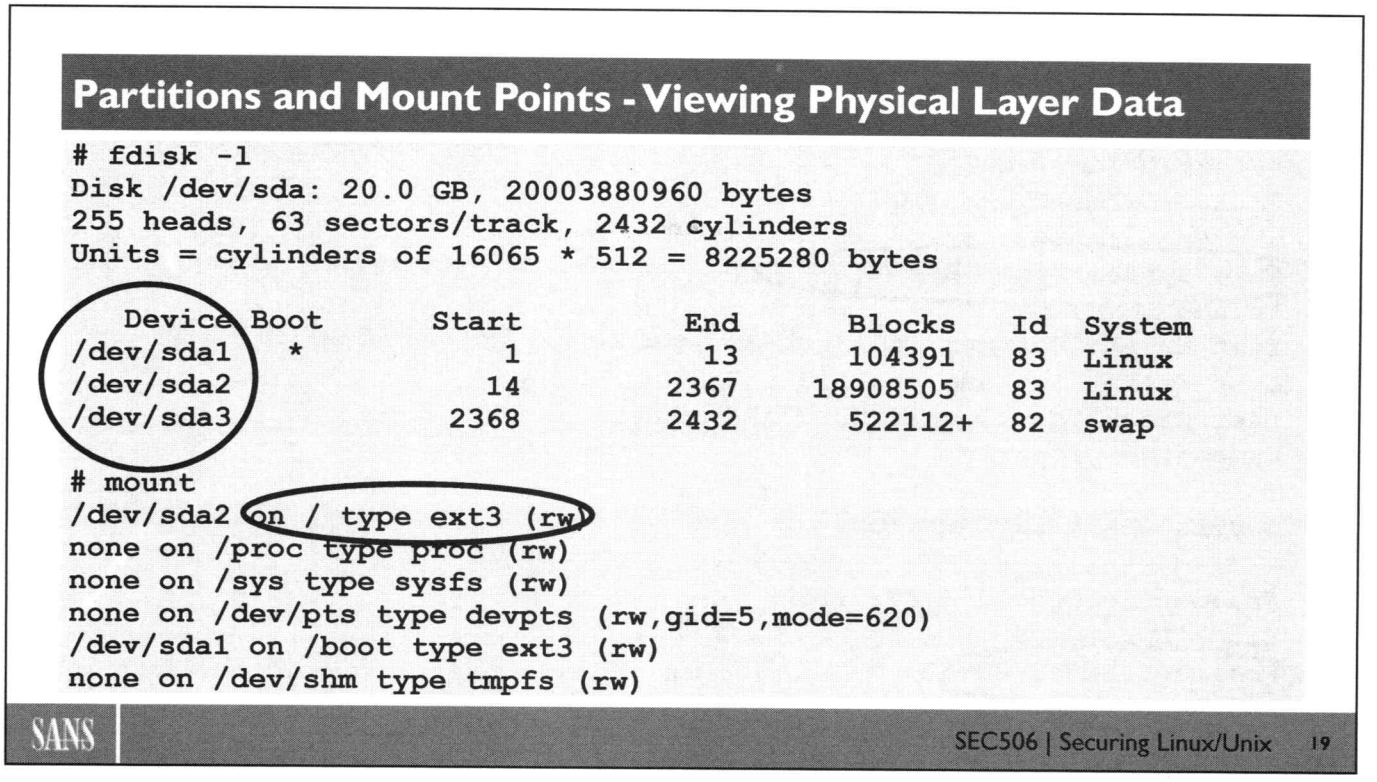
A Layer / TSK Case Study

Let's take a moment to understand how the concepts that we've just presented can be used in a real-world investigation. One method investigators often use is to search a disk partition for "strings of interest." We can use `grep -b` to provide the byte offset of the string, indicating how many bytes into the partition it actually is, but as you can imagine, that doesn't do us a lot of good.

If I find a string that might indicate evidence, then I want to know if it resides in a file and if so, what is the file's name? It is possible to figure this out, but it takes a bit of work and a progression through the different layers that we've seen.

Let's do an example, and introduce some tools along the way.

© 2017 Hal Pomeranz and Deer Run Associates



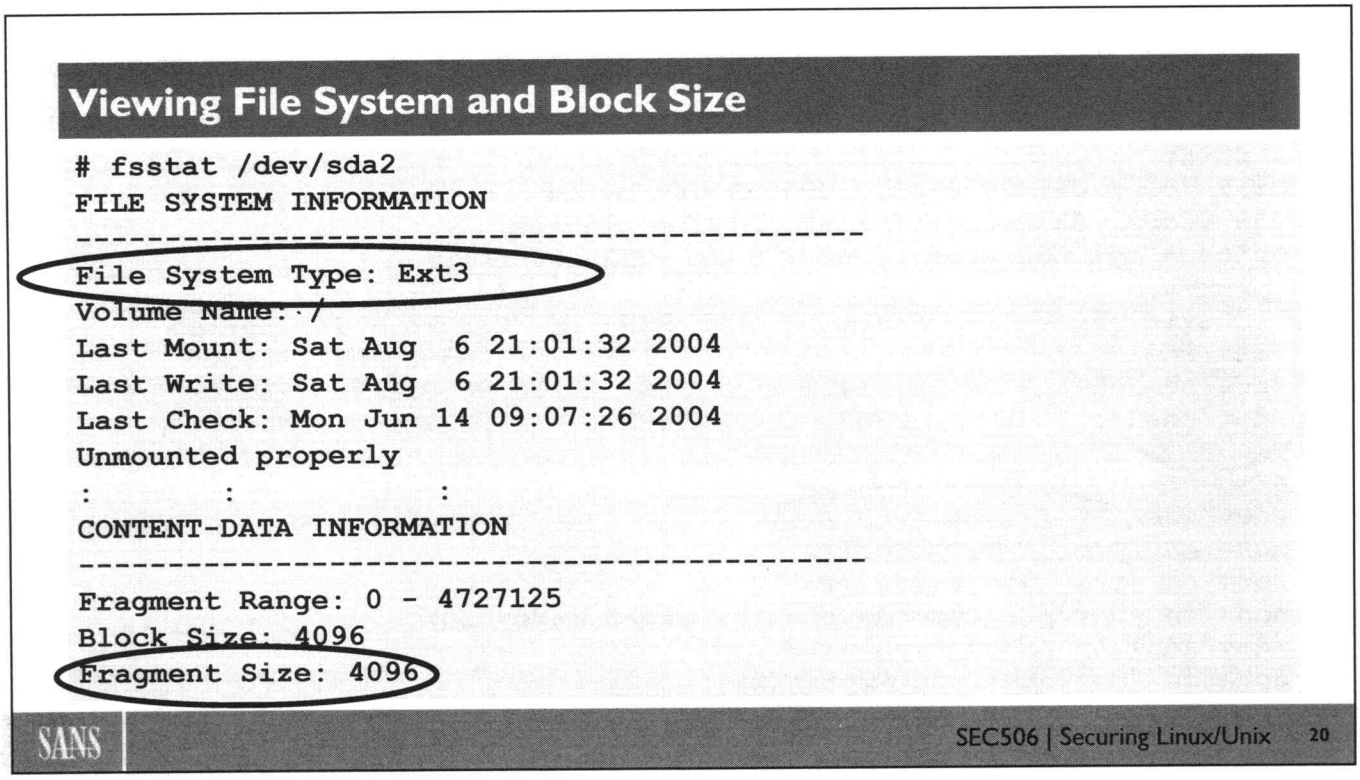
Partitions and Mount Points - Viewing Physical Layer Data

For any disk or image that you might be investigating, information from the physical layer is very important. Many of the tools we will use will need to know what the file system is, and where a particular partition may be mounted.

In this case, we have used `fdisk -l` to safely display the disk's partition table. We also used the `mount` command to see where these partitions were actually mounted into the file system. Although the `df` command could be used to list partition information, the advantage that the `mount` command provides is that, in addition to partition and mount information, it also shows us the file system type as it is mounted on the system.

Of course, in a normal forensic investigation, we wouldn't be running these sorts of commands on the live system. Instead, we would use the `mmls` tool from TSK to access the partition information from a *disk image* that was created from the original drive. We'll discuss disk imaging and meet `mmls` later in this course.

© 2017 Hal Pomeranz and Deer Run Associates



Viewing File System and Block Size

Once we have the physical layer data, we'll need to get a feel for the File System layer information. Although all of the data here is interesting to us, the most important for the purpose of this walkthrough is the File System Type (Ext3), and the Block Size (4096).

```
# fsstat /dev/sda2
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext3
Volume Name: /
Last Mount: Sat Aug  6 21:01:32 2004
Last Write: Sat Aug  6 21:01:32 2004
Last Check: Mon Jun 14 09:07:26 2004
Unmounted properly
Last mounted on:
Operating System: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Dir Index
InCompat Features: Filetype, Recover,
Read Only Compat Features: Sparse Super,

CONTENT-DATA INFORMATION
--------------------------------------------
Fragment Range: 0 - 4727125
Block Size: 4096
Fragment Size: 4096
```


© 2017 Hal Pomeranz and Deer Run Associates

Where's Waldo? Block 170338!

```
# grep -abi waldo /dev/sda2 > /tmp/found_waldo.txt
# less /tmp/found_waldo.txt
:   :   :
689702819:^@^@<8D^@submit@bugs.kde.org^@(c) 2003 Waldo
Bastian^@Author^@bastian@kde.org^@No
696076095:Do you want to save the changes or discard
them?^@editor^@0.5^@submit@bugs.kde.org^@KDE Menu
Editor^@kmenuedit^@bastian@kde.org^@Waldo
Bastian^@sandrini@kde.org(C) 2000-2003, Waldo 697911472:Waldo
697911478:Waldorf
:   :   :
```

697911472(byte offset) / 4096(blocksize) = 170388(block)

Now, Where's Waldo? Block 170338!

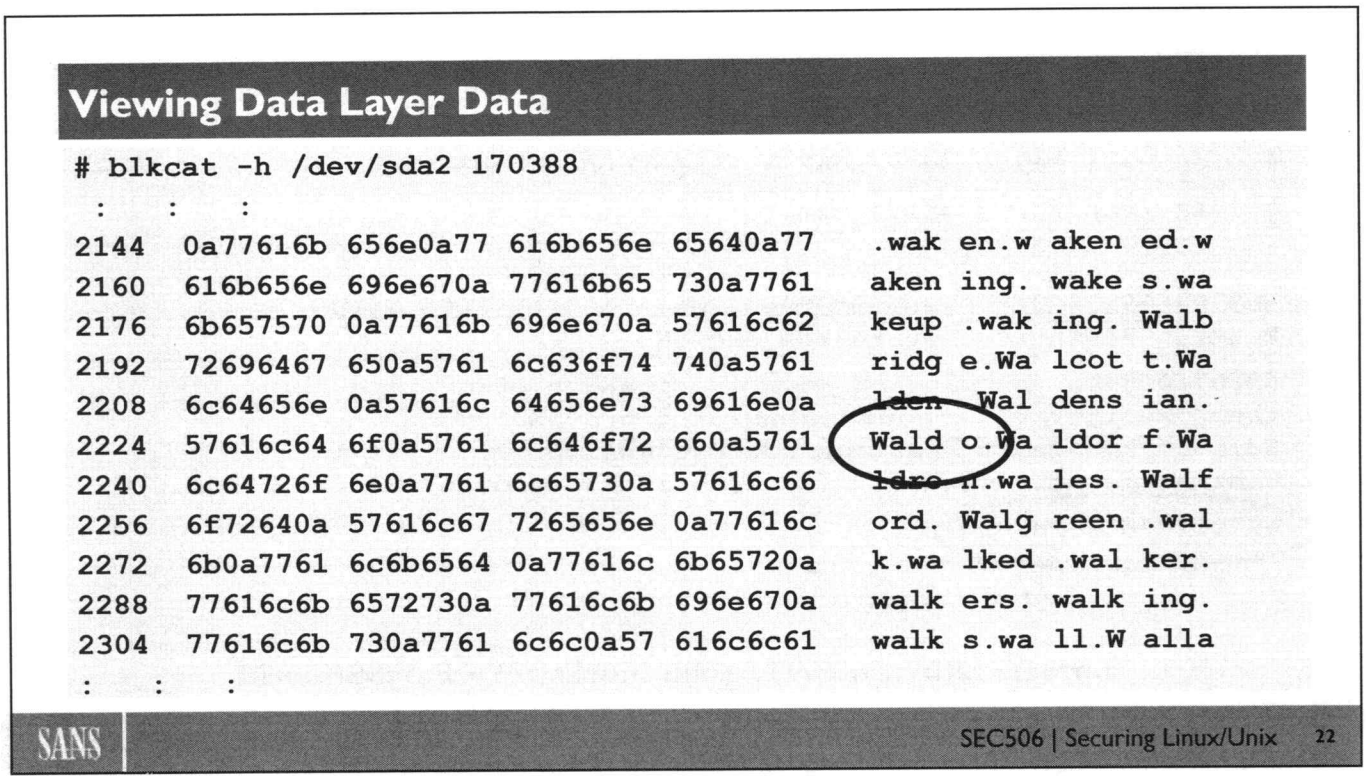
At this point, we've got all of the information that we need to start the fun part. In this example, we are going to use the string "Waldo" because it's more fun, but in an investigation, you will be searching for strings that are commonly associated with malware, stolen or illicit data, etc.

To begin, we can use the `grep` command with some specific flags: the `-a` flag says to treat everything as ASCII data (so `grep` doesn't choke on the raw binary file system data), the `-b` flag says to provide a byte offset at the beginning of each matching line, and the `-i` flag says to ignore case when searching for the string in question (you've probably used this option many times). To make things easy for this example, I have redirected the output to a temporary file.

When that finishes, we can examine the resulting file to see if Waldo was found, and indeed, he was—many times. Choosing an interesting line, I record the byte offset.

Now to calculate which disk block that occurrence of Waldo was in, I divide the byte offset (697911472) by the block size (4096) we got from the `fsstat` output on the previous slide and truncate the result to an integer (if you use the `expr` command in Unix, it will automatically give you the integer output). My result is that Waldo is hiding in block number 170388 on the disk.

© 2017 Hal Pomeranz and Deer Run Associates



Viewing Data Layer Data

Before I go too far, I want to know if this is an "interesting" Waldo. It is often very helpful to see the string of interest in the context of surrounding data, and then make a decision about whether it may be related to the case under investigation.

So, we can use information from the data layer and a neat tool called `blkcat`. `blkcat` will display the contents of a disk block to STDOUT. Be careful though, listing binary contents to STDOUT can get messy. Instead, we'll make `blkcat` give use results in a hexdump-like display format using the `-h` flag. You must also specify the partition name and block number on the `blkcat` command line.

In the slide above, you see the hexdump results from `blkcat`, and in fact, Waldo appears twice! It looks like in this case, Waldo may just be part of a dictionary or wordlist, but we'll keep going with our example anyhow.

© 2017 Hal Pomeranz and Deer Run Associates

Allocated to a File? Which One? Viewing the Meta Data Layer

Is it allocated to a file?

```
# blkstat /dev/sda2 170388
Fragment: 170388
Allocated
Group: 5
```

Which inode does it belong to?

```
# ifind -d 170388 /dev/sda2
69739
```


Allocated to a File? Which One? Viewing the Metadata Layer

Just because we found a block that Waldo resides in, doesn't mean that this block is still allocated to an existing file. It could've been deleted long ago. So, to check to see whether or not it belongs to an existing file, we can query the metadata layer to see if there are any inodes that have this disk block allocated to them.

The `blkstat` command provides allocation statistics about a given block on disk. It requires the same parameters that we saw earlier: a file system image or partition, and the address of a disk block. Here, we see that it is indeed allocated to an existing file.

Remember the goal of this exercise was to find the name of the file that contains this instance of the string Waldo, so we're going to need to figure out which inode has this block allocated. For this, we'll use the `ifind` command. Given a block number, this command will search the inodes to find which one has that block allocated. Again, the parameters are very similar to earlier commands we've seen. The exception is that to specify the disk blocks address, we must use the `-d` flag.

`ifind` determines that the block is allocated to inode number 69739.

© 2017 Hal Pomeranz and Deer Run Associates

How About the File Metadata? Viewing Metadata Layer Data

```
# istat /dev/sda2 69739
inode: 69739
Allocated
Group: 4
uid / gid: 0 / 0
mode: -rw-r--r--
size: 409305
num of links: 1

Inode Times:
Accessed:        Tue Feb 17 19:47:53 2004
File Modified:   Tue Feb 17 19:47:53 2004
Inode Modified:  Sun Jun 13 23:13:18 2004

Direct Blocks:
170290 170291 170292 170293 170294 170295 170296 170297
:      :      :
```


How About the File Metadata? Viewing Metadata Layer Data

Now that we have the inode number that allocated the block that OUR Waldo resides in, it might be informative to take a look at the data contained in an inode to get a feel for the size of the file, the owner, permissions, and MAC times. To do this, we can use the `istat` command. We can see that it is quite a large file, owned by root, and readable by everyone. It has been quite a while since the file was access or modified though.

TSK also provides a tool called `icat` which would enable us to dump the contents of this file. However, we're more interested in figuring out what file this actually is, so let's proceed.

© 2017 Hal Pomeranz and Deer Run Associates

Viewing Human Interface Layer Data. And Now the File Name ...

```
# ffind -a /dev/sda2 69739
/usr/share/dict/linux.words
```

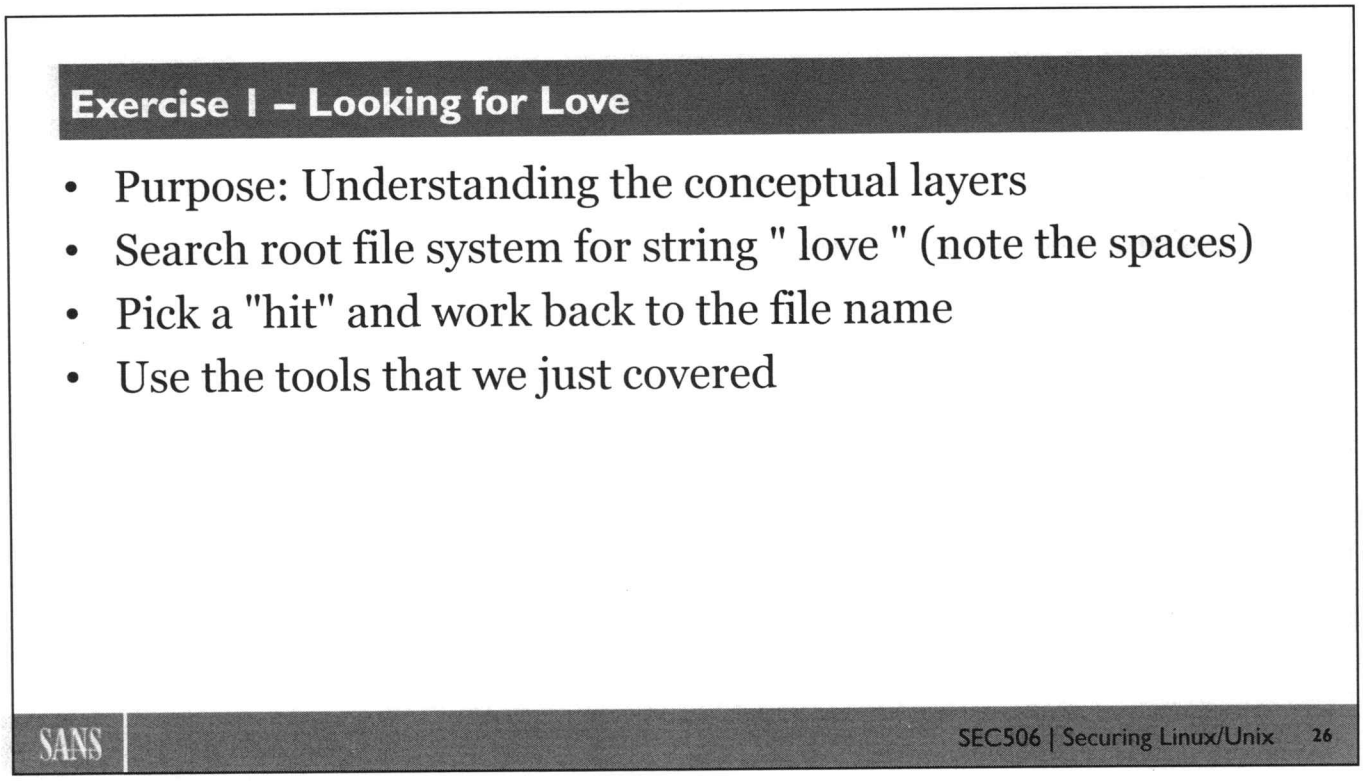

Viewing Human Interface Layer Data. And Now the File Name ...

Finally! We've reached the point that we can determine the name of the file where Waldo was hiding. Now that we have the inode number, we can use the `ffind` command to determine the file name. Given an inode number, `ffind` looks through the directory entries finds the inode number and displays the corresponding file name.

The `-a` flag tells `ffind` to display all of the entries or names associated with this inode—after all, there may be multiple hard links pointing to a single inode. However, if you look carefully at the `istat` output on the previous slide you'll see that the "num of links" value is 1, so as soon as we find the first directory entry that points to this inode that's it. But the `-a` option will force `ffind` to scan the entire file system rather than stopping on the first hit—a waste of time in this case.

So, Waldo was hiding in the file: `/usr/share/dict/linux.words`. Whew!

© 2017 Hal Pomeranz and Deer Run Associates



Exercise 1 – Looking for Love

Exercises are in HTML files in two places: on the course USB media, and in the home directory of the SANS user in the lab virtual machine. These two locations contain identical copies of the exercise files—it just depends on whether you prefer to look at them from a web browser inside the lab VM (because you're working in full-screen mode) or from your host operating system.

1. Open your web browser
2. Select "File … Open …" from the browser menu (Ctrl-O is the usual keyboard shortcut)
3. This is Day 6, Exercise 1, so navigate to ... */Exercises/Day_6/index.html* and open this HTML file
4. Click on the hyperlink which is the title of Exercise 1
5. Follow the instructions in the exercise

© 2017 Hal Pomeranz and Deer Run Associates

Introduction to Forensics

SANS | SEC506 | Securing Linux/Unix 27

Introduction to Forensics

Forensic investigations are not rocket science. Though they do require a firm understanding of the operating system and file system of the victimized computer. It also helps a tremendous amount to have a grasp of the techniques that attackers use to compromise systems and conceal their activities. Finally knowing what tools are available to help the process along is invaluable.

Forensics requires patience, tenacity, and a little bit of luck. Combine those qualities with a bit of process and a standardized methodology and you're well on your way to being a forensic analyst.

Although we'll be walking through the forensic process in a day, be aware that the investigation of a compromised system may easily take forty or even eighty hours of work. Now that's patience and tenacity!

© 2017 Hal Pomeranz and Deer Run Associates

Forensics in a Nutshell

- Validate compromise
- Evidence seizure
- Investigation and analysis
- Reporting Results

"*Gathering and analyzing data in a manner as free from distortion or bias as Possible to reconstruct data or what has happened in the past on a system.*"

–Farmer and Venema, 1999 http://www.fish.com/security/forensics.html

Forensics in a Nutshell

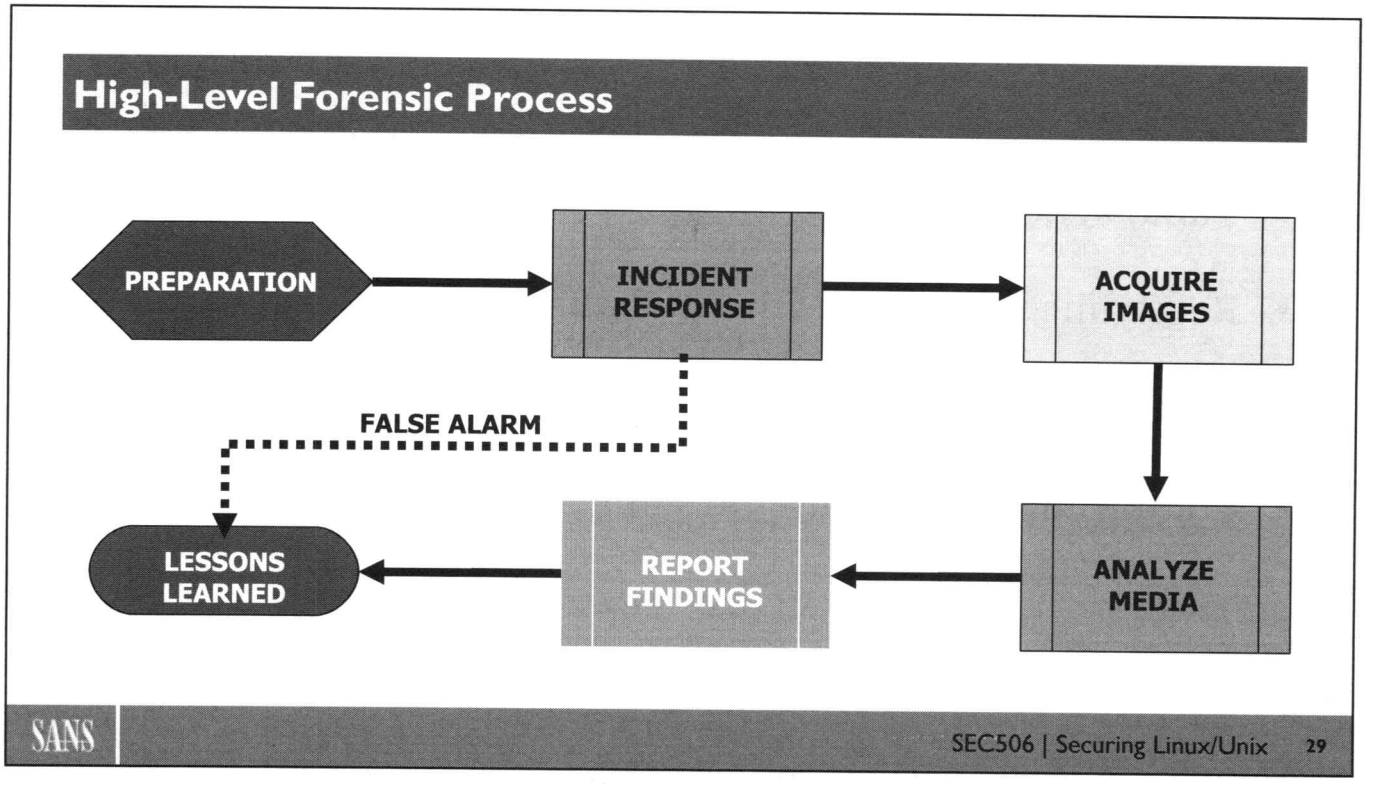
Computer forensics is more than just analyzing blocks of data. It is the effective gathering, examination, and reporting of your actions and findings. A seasoned investigator knows that if one step is overlooked, his case will not yield the results he or she may desire.

Evidence seizure generally occurs during the incident response phase where you must verify the incident, but you also begin your work to collect volatile and non-volatile data. Data that is volatile is lost if the system is adjusted prior to the collecting of that data, a memory dump of a process that contains key IP addresses of the attacker or subject. Non-volatile data is a hard drive that is powered off, static CD-ROMs. There may also be information such as backups and log files stored on other systems.

Investigation and analysis occur when the investigator takes what is collected and analyzes it to form a clear picture of the incident. This analysis uses tools and techniques that require data recovery, piecing together the puzzle of what happened, and forming a timeline of events.

Reporting your results becomes the most important step. Without accurate reporting, the investigator often finds himself unable to find anyone willing to prosecute his case or take action. Without action, why perform the investigation? Reporting is key.

© 2017 Hal Pomeranz and Deer Run Associates



High-Level Forensic Process

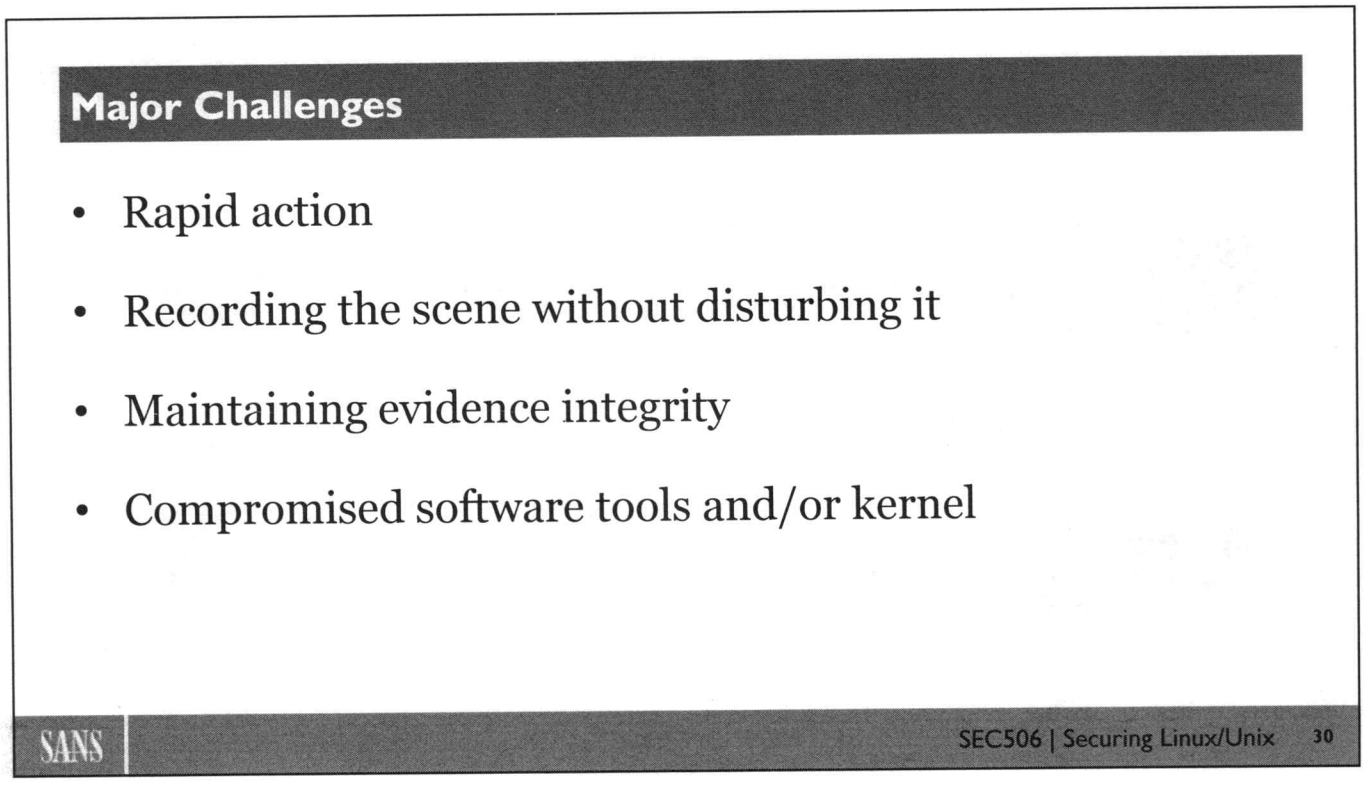
Here you see a simplified flowchart of the forensic investigation process. We'll discuss each of these steps in detail throughout the course of the material, but the process is worth mentioning here at a high level. Being adequately prepared to respond is absolutely critical. If you spend enough time in this proactive phase, you can greatly reduce the stress that you encounter and the chance of making costly mistakes during the reactive process of incident response.

It is important to mention that not every incident is an intrusion or compromise, so we have incorporated a dotted line to indicate that false alarms do occur. In these cases, the incident response team is activated to handle what appears to be an incident, but it later turns out to be something less serious—such as an improperly configured system or over-zealous user/administrator.

Therefore, the first step of incident response is usually to validate that an intrusion has occurred. Beyond that, incident response is a carefully choreographed dance that balances the investigation of a system with the requirement of minimized data loss due to the investigation. Image acquisition is the process of creating true bit images of the disks and removable media that may be associated with the system in question. It is an art-form unto itself, albeit an easy one to master. Once these images have been acquired, the original evidence (system disk) can be locked in a container for safe keeping.

You will conduct your investigation against these images, performing media analysis. Examining the images for small pieces of evidence that will ultimately help you to reconstruct what happened in the past on a system. The two final pieces are the ones that are most often overlooked. The creation of an accurate and effective incident report, and incorporating any lessons learned back into the security processes and policies of the organization.

© 2017 Hal Pomeranz and Deer Run Associates



Major Challenges

As I see it, there are several major challenges facing the forensic investigator.

The first challenge is rapid action. When an incident occurs, an immense amount of pressure is placed on the incident handling team to investigate and recover from the compromise as soon as possible. This can make it very difficult if not impossible, to collect evidence in a thorough manner.

Another problem is making sure that once you've collected and analyzed the data, it can be submitted as evidence in court and will hold up under scrutiny. Remember, the defense's goal is to convince just one juror that you didn't have all of your I's dotted and T's crossed when you were investigating the incident.

A key piece of the investigation strategy is to avoid disturbing the crime scene. The question is. "How does one record the scene of a computer incident without disturbing it?" Merely by logging on to the system, the "state" of the computer is altered.

And finally, how do you investigate a computer, using the tools that may themselves be compromised, or changed to hide the intruder's activities and presence. I hope to show you how to overcome these obstacles during the course of the lecture today.

© 2017 Hal Pomeranz and Deer Run Associates

Guiding Principles

- Work to minimize evidence loss
- Take great notes in excruciating detail
- Collect all the evidence that you can
- Analyze everything you collect
- Prove that evidence/process integrity has been maintained
- Learn lessons from each incident

Guiding Principles

There are a few basic principles to assure success when performing a forensic analysis on a computer. The number one rule - attempt to minimize the loss of evidence. There is no way to NOT affect the system. You will lose evidence. The key to managing this problem is understanding what effects your actions have and what data is preferable to lose. Armed with this knowledge, you can take steps in order to minimize that loss and maximize the data available for collection.

Record everything. Record what you do. Record computer memory, disks, and everything else about the computer. Record the scene. There is nothing that you should not record while at the scene. Remember, once you leave the scene or power down the system, much of the evidence is gone.

Analyze all the data that you collect. You'll likely end up with a huge amount of data, but don't let that deter you. Forensics can be a tedious process; thorough investigation of a single system can take days or even weeks depending on what you find when you start looking.

Maintaining the integrity of the evidence is critical. You will learn what you can do to prove that the integrity of the evidence has not been lost, and at what points in the investigation you can apply these concepts.

© 2017 Hal Pomeranz and Deer Run Associates

Forensic Preparation

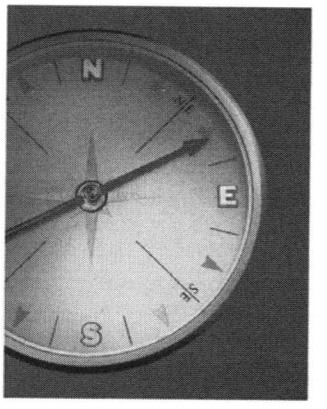
Forensic Preparation

Much of the work involved in incident handling and forensic investigations can, and should be done ahead of time. These proactive measures will help you every step of the way when actually responding and investigating an incident. Having plans, call lists, response policies, an adequate detection infrastructure, and a trusted set of tools may ultimately make the difference between a successful response/investigation and one that is botched.

© 2017 Hal Pomeranz and Deer Run Associates

Preparation Objectives

- Methodologies guide you
- Infrastructure alerts you
- Forensics labs enable you
- Toolkits assist you
- Policies protect you



Preparation is the key to effective and efficient DFIR

Preparation Objectives

In this section, we'll take a quick look at some of the things that you should consider doing ahead of time, and during the non-response hours of your day. Understand that each of the areas on the slide above provide great benefits during the investigation.

If the case that you are investigating goes to court, one of the things that you will be questioned about is the investigation methodology that was used. Obviously, a well thought-out and planned methodology is superior to an ad-hoc response. A good methodology will be a general and flexible framework that provides guidance on the steps of the investigation. It must be flexible though, to account for the fact that no two incidents are the same, and so, responses may need to vary slightly. Be careful though, if your organization has developed and documented a rigid methodology then you had better follow it, or else that could become a point of contention.

If correctly configured, your network and computing infrastructure contribute supporting data to the investigation. This primarily revolves around network time, audit logs and IDS/firewall captures, but can also include the creation of a trusted baseline image or the creation and storage of a hash database.

The forensic lab is your office though it need not have walls. It is the thing that enables you to perform your job. For the purposes of this course, our forensic lab will be limited to the scope of a bag of critical items that are typically needed for response, including a laptop system that has been loaded with the forensic tools of your choice.

Much of the forensic process could be extremely laborious, repetitive, and prone to error if left to the very low-level tools that exist to manipulate the data on a disk. Luckily, some really smart people have created some great tools over the last few years that all but eliminate these hardships.

Finally, in a world where critical decisions that may adversely affect business must be made quickly, response policies are invaluable for protecting you: the responder/investigator.

© 2017 Hal Pomeranz and Deer Run Associates

Two Key Questions

1. Will all incidents will lead to prosecution?
2. Who's on the incident response team?

SANS | SEC506 | Securing Linux/Unix 34

Two Key Questions

Before you start your planning, it's worth considering the answers to two key questions. First, as a matter of general policy, does your organization go into incidents assuming that the end result will be prosecution via the legal system? If yes, then you will tend to be much more careful about collecting, storing, and examining evidence than if your goal was simply to understand the compromise and get your systems functioning again as quickly as possible. Prosecution also means additional time away from work for the investigators, system administrators, and other potential witnesses and may involve additional costs for storing and producing evidence.

The other question to answer is, "Who is involved in handling incidents in your organization?" You need to know this information so that (a) you can publish it within the organization so people know who to call, and (b) you can as a matter of policy grant the individuals involved in incident response extraordinary powers when dealing with incidents. Answering this question also lets you document who's *not* involved in handling incidents, which can help stop well-intentioned but unauthorized/untrained personnel from messing up your investigations.

© 2017 Hal Pomeranz and Deer Run Associates

Do You Have a Plan?

- Do you have an incident response plan in place?
- Plans should be different for different types of incidents
  - External Incident
    Intrusions, viruses, denial-of-service, theft of service
  - Internal Incidents
    Intellectual property theft, malicious intent, policy abuse
- Documented as part of your incident response policy

Do You Have a Plan?

Does your organization have a specific computer incident response management and policy guide? While it may be boring for the typical person to work out the details of these plans, nothing is more important than this document that will outline who does what, and which actions can and will be taken during the course of an incident.

Incidents create chaos; you need to mitigate that chaos by having a well thought out plan prior to the incident. There should be different plans for the two major categories of incidents: internal versus external. Well prepared organizations will even have different plans for intrusions versus viral outbreaks versus DoS attacks. Work with the person responsible for your corporate disaster recovery and business continuity planning, they will be grateful for the help and may have a lot of valuable advice.

While external incidents get the majority of the press, and will often consume much of an incident response team's time, it's the internal incident that usually does the most damage to an organization. They key thing to understand is that your response will likely be drastically different in these two cases. The external incident is likely to elicit the "no holds barred" style response: get in there, get it cleaned up, and get us back into business as quickly as possible. On the other hand, the responding to, and investigating an internal incident is likely to involve an entirely different set of players (Human Resources for example) and will be conducted in much more secrecy perhaps over a longer period of time.

The one truth is that regardless of the incident type, having a well thought-out plan in place for handling it will eliminate most of the guesswork and hesitation that frequently slows down or halts an investigation.

© 2017 Hal Pomeranz and Deer Run Associates

Policies Remove Doubt

Responders often called-in outside of normal working hours

- Decision Makers may be hard to contact
- IR Team may need elevated authority during incident
- Clearly document this authority

Incidents are high-stress situations

- *Make incident handling decisions ahead of time and document them to avoid paralysis*

SANS | SEC506 | Securing Linux/Unix 36

Policies Remove Doubt

Unfortunately, attackers don't compromise systems on a 9am to 5pm basis. In fact, it seems as if the most major incidents occur around 5pm on a Friday. It's a common running joke: "It's 5pm, do you know where your compromise is?"

During these non-business hours, it can be difficult to contact those people in the organization who are normally responsible for making critical business decisions. Like what? Like taking your e-commerce web server offline, and essentially disabling a major income vector of the business.

Forward thinking organizations plan for this eventuality and provide a lead responder with elevated levels of authority when the normal call escalation process is exhausted. If this type of countermeasure hasn't been put into place and documented, it can be a frightening experience for the lead responder: Do I pull the plug and bring the business to a halt? Do I wait till I can get authorization and risk the loss of reputation or intellectual property? Will I lose my job as a result of my actions? You can understand the stress that such a situation creates. That stress usually results in responder paralysis.

However, if an escalation process has been documented and includes the ability of the lead incident handler to make those tough calls when a decision maker can't be reached then one can feel a bit more confident in proceeding. You see, this works even better if incident response plans have been put into place, and you have a general idea of how the organization SHOULD respond given an incident of type X.

If management is uncomfortable giving that kind of authority to a single person, then perhaps a majority 2 out of 3 votes of the three senior members of the incident response team would be an acceptable alternative for them.

© 2017 Hal Pomeranz and Deer Run Associates

Prepare Your Infrastructure

Sync system time across the enterprise
- Use Network Time Protocol
- Consider GMT if you are a global organization

Ensure logs are being generated and reviewed
- Firewalls, IDS, e-mail, file servers, system logs

Make sure back-ups are created and safely stored
- Critical servers and tertiary servers

Create hash databases for deployed platforms

Prepare Your Infrastructure

Before anything ever happens on your network, you can work to ensure that your infrastructure is prepared to perform its role in the investigation process.

Make sure that every machine on the network is time-synched. Usually, simply adding a network time server or using one on the Internet will suffice. If your organization is geographically dispersed or even global, then it is recommended that your system times are reflected in GMT (Greenwich Mean Time) or synched to the location of corporate headquarters.

You should have a policy in place to regularly store and backup your network log files. This is especially critical in the case of your firewalls and IDS (Intrusion Detection System) log files. A properly-tuned IDS and firewall log will generally be easy to store for a long time since the log files will be smaller.

Ensuring your network servers have a backup is very important. If you have to take a server offline for an investigation, what will send your e-mail to employees or serve your customers on the web? Having a backup is a good idea and usually, most larger companies do as it is truly the LAST line of defense and the last resort of response.

If I have a critical system and create a hash for each binary on that system, I can use those hashes to detect changes to any of those system binaries—indicating a possible compromise. This is exactly how file integrity tools work (e.g., Tripwire, AIDE, etc.), and is an excellent idea to implement organizationally.

© 2017 Hal Pomeranz and Deer Run Associates

Pre-Built Forensic Distros

SIFT Workstation

http://digital-forensics.sans.org/community/downloads

Paladin

http://sumuri.com/product-category/paladin/

CAINE

http://www.caine-live.net/

DEFT

http://www.deftlinux.net/

Pre-Built Forensic Distros

You'll also want a collection of specialized tools to help you investigate the incident. There are several different distros available that are specifically designed for incident response and forensics. SIFT is a Linux distro originally created by Rob Lee for the SANS Forensics Curriculum, but now a fully functional forensic distribution based on Ubuntu Linux. CAINE, DEFT, and Paladin Linux are other well-known Linux Forensic distros.

Kali Linux (formerly BackTrack) is generally thought of as a hacking or penetration testing platform, but it also happens to contain many forensic tools as well.

© 2017 Hal Pomeranz and Deer Run Associates

Incident Response Process

Incident Response Process

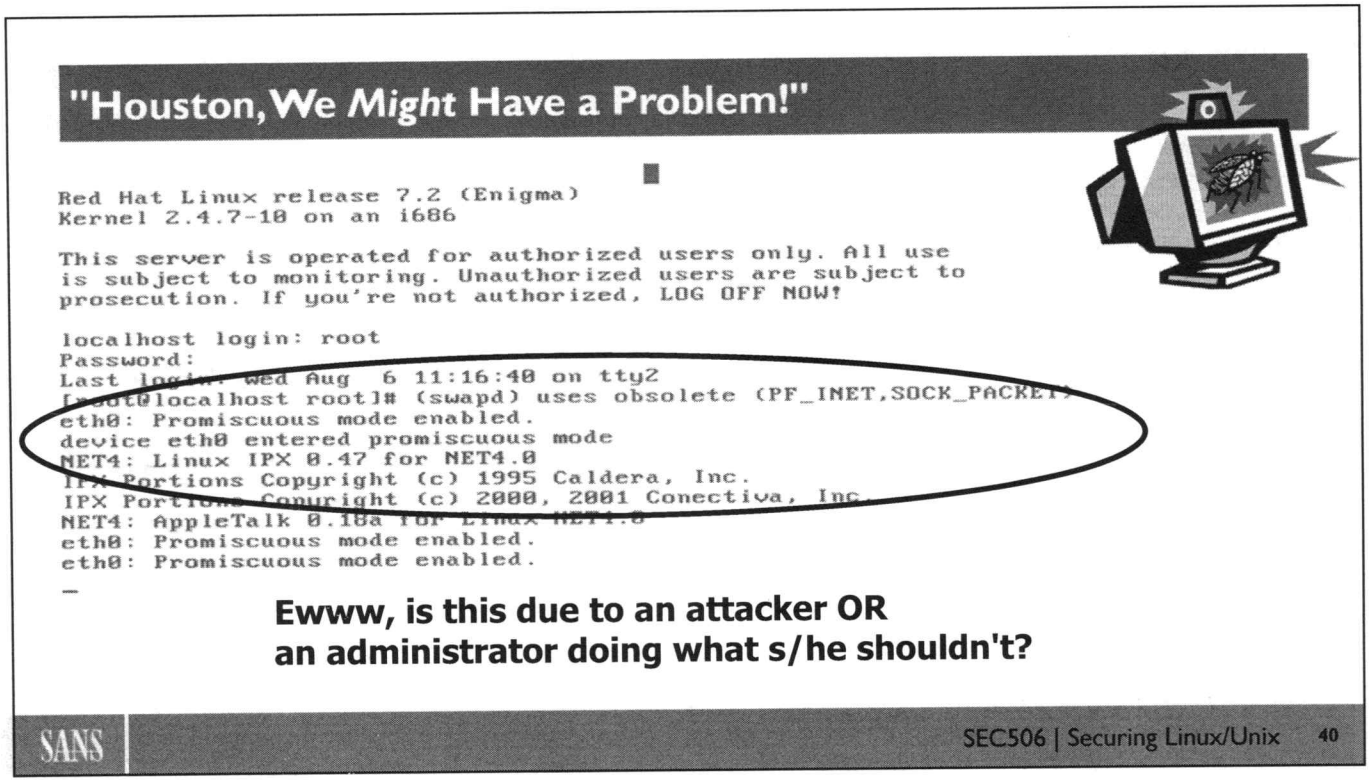
The practice of incident response is an art form unto itself, regardless of the incident. In theory, one can certainly extract forensics from the incident response process and look at it alone, but that provides little help or guidance to those who must implement the knowledge that they gain.

Instead, for the purposes of this course, we have chosen to look at how incident response and forensics are used in tandem, in the real world. The incident response team is likely the same team that will collect and ultimately analyze the evidence so we will be approaching the problem holistically. Though there is no true standard incident response process when it comes to the forensic side of things, there are certainly some acknowledged best practices and guidelines. This course is an attempt to present these in a Unix context, at a high level.

In general, the incident response process as it applies to forensics involves the collection of evidence. This collection must be performed in a forensically sound manner. Care must be taken to collect evidence that accurately reflects the state of the system as it was discovered. This is a time critical process, and one in which the order of collection is of utmost importance.

Although there are specialized tools for the collection and analysis of digital evidence, many of them are operating system commands that Unix system administrators should already be familiar with.

© 2017 Hal Pomeranz and Deer Run Associates



"Houston, We *Might* Have a Problem!"

Imagine being called to the console of this system. "Hey, John! Come take a look at this, something weird is on the screen!" The first thing noticed is that the screen is displaying a message indicating the "*device eth0 entered promiscuous mode*".

When a Network Interface Card (NIC) enters promiscuous mode, traffic on the local LAN segment is being sniffed. The knee-jerk response is that this is bad, really bad. But, has the system actually been compromised? It could be one of your network admins running `tcpdump`. Or is it an over-zealous administrator or user playing with new toys? Either way, it's an incident, but a sniffer as part of a compromise is much worse.

A quick interview with the users and administrator would probably be enough to say for certain whether or not this system has been compromised. However, this is a good time to introduce the fact that all incidents and suspected compromises should be validated before activating the entire incident response team.

© 2017 Hal Pomeranz and Deer Run Associates

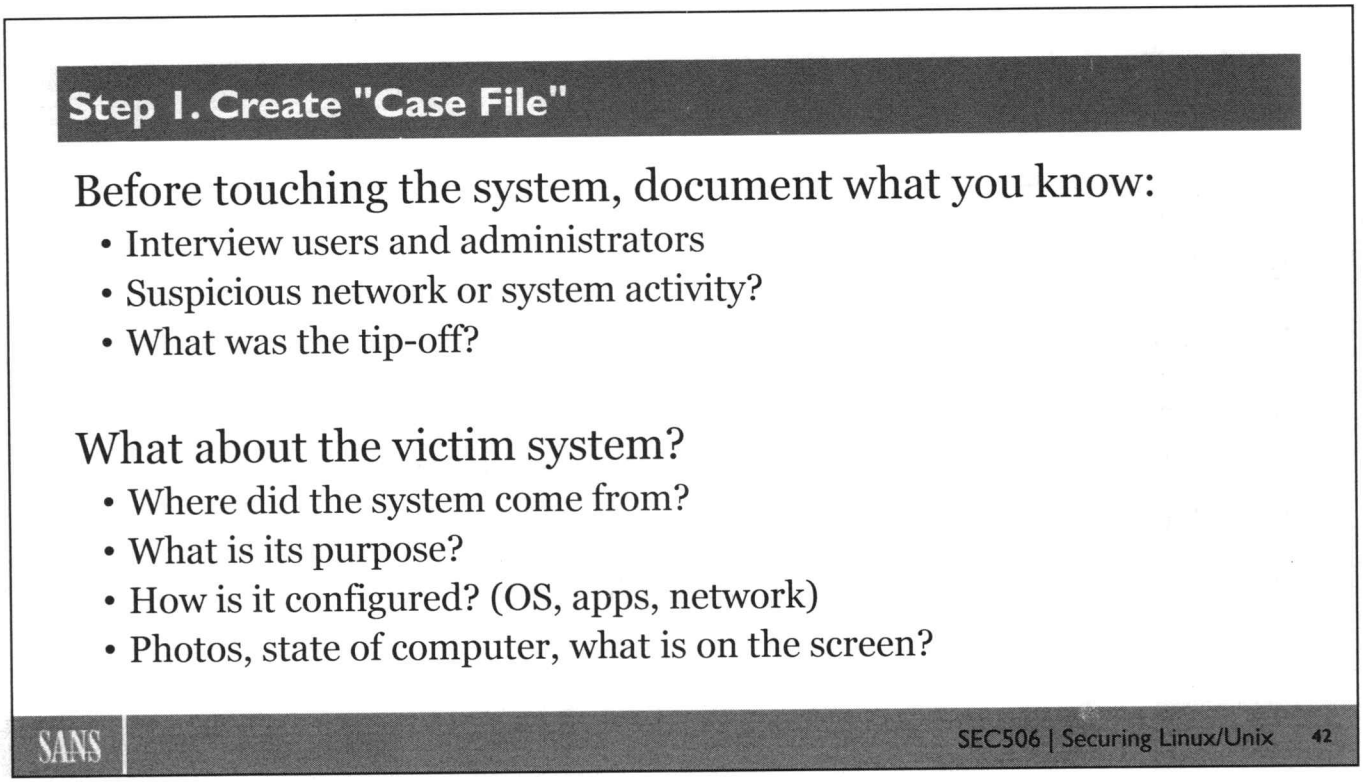
Incident Response Methodology

1. Open "case file", snapshot scene
2. Validate compromise
3. Collect different types of evidence
4. Create timeline, perform MAC analysis
5. Analyze file system, recover deleted data
6. Create incident report
7. Document and apply lessons learned

Incident Response Methodology

The overall forensic investigation methodology will remain the same from operating system to operating system. You will probably conduct some, if not all, of these steps in every investigation. Despite the tool that you use, your overall process will and should remain the same. If you use a commercial tool versus and open source toolset, you will still gather evidence, obtain investigation leads, and perform data recovery. Regardless of what your tools are, your methodology will aim to reconstruct what has occurred in the past on a system, and hopefully, point to the intruder.

© 2017 Hal Pomeranz and Deer Run Associates



Step 1. Create "Case File"

This is pretty straightforward. Just like in a museum. "DON'T TOUCH!" and especially don't let others touch.

You aren't looking for actual fingerprints in the local area. You are trying to limit things that could "change the state" of the system. Even choosing to limit activity on the system ALSO changes the state of the system. So, you are trying to keep the changes on the system to a minimum. These methods might not be the best, but it is ONE way you could go about completing this goal!

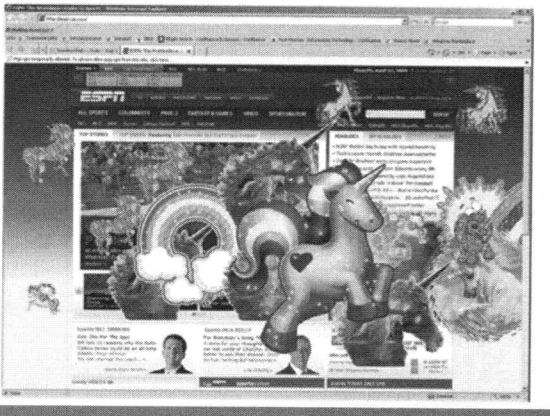
In general, describe the system you are analyzing. Where did you acquire the system? What is/was it used for? What is the configuration of the system (OS, network)? Include any other information you feel may be necessary to perform the investigation.

Your system description will affect the way your investigation is executed. If this machine is a critical server, you may not be able to shut the server down. If the machine is a workstation you may need to determine what the workstation is utilized for. Being able to predict the type of information that the system stores it would aid in your ability to perform your collection and your analysis.

© 2017 Hal Pomeranz and Deer Run Associates

Step 2. Validate the Compromise

GOAL—Find evidence system has been compromised *before* activating the entire incident response team



Validation sometimes easy:

- Defaced web pages
- Obvious DoS attacks
- IDS captures

SANS | SEC506 | Securing Linux/Unix 43

Step 2. Validate the Compromise

Occasionally, validation is a no-brainer. A pink pony on the homepage of your businesses website is a big clue that something has gone very wrong. IDS captures may provide conclusive evidence that a system's security has been breached, or that a DoS attack is the reason that your site has lost Internet connectivity. However, the clues that something has gone wrong are not always this obvious. In fact, they rarely are.

Therefore, once you have been notified of a suspected incident and have opened a case file, it is wise to work to validate that a system has actually been compromised. There will be numerous incidents where the compromise wasn't really a compromise after all—and there's the incident response team, all psyched up and nowhere to go.

© 2017 Hal Pomeranz and Deer Run Associates

What Are We Looking For?

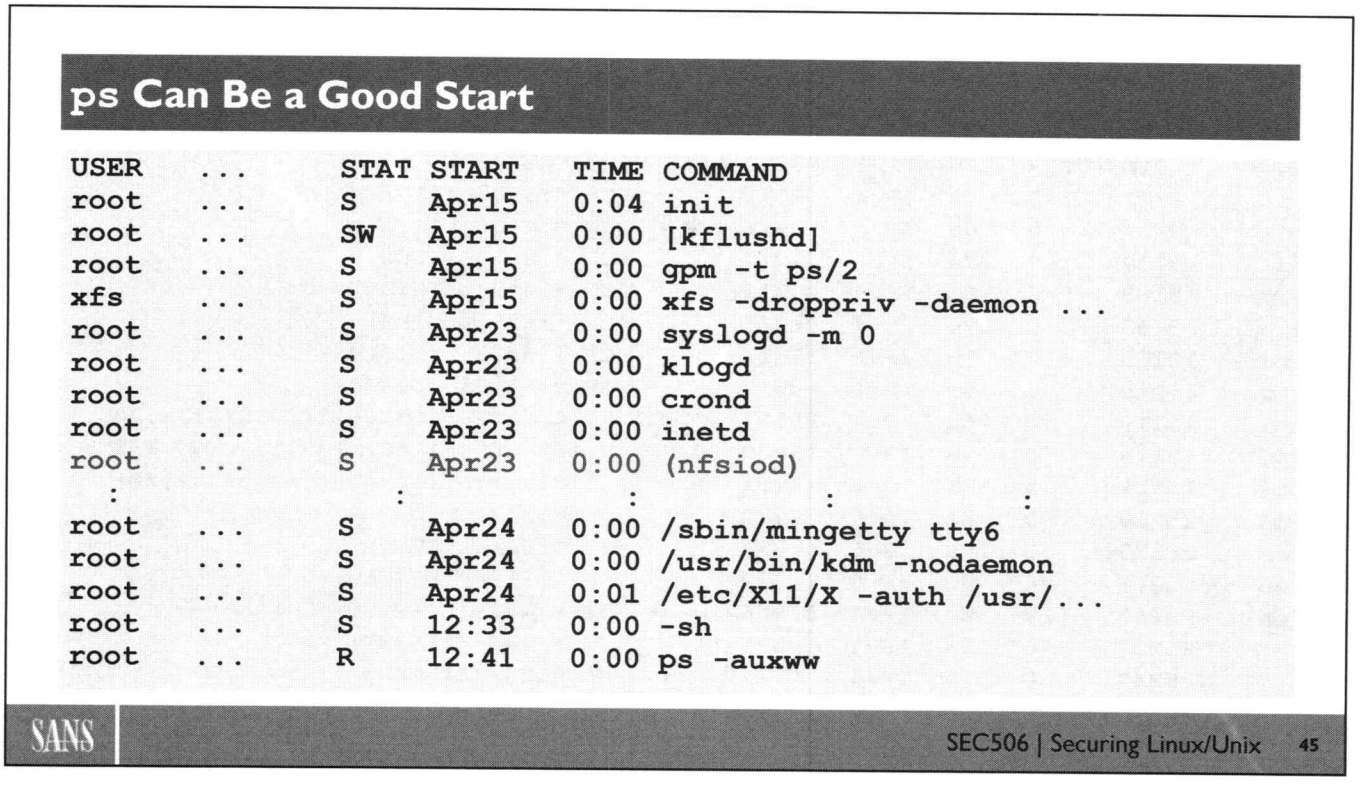
Example evidence might include:

- Backdoors on open ports
- Hidden files or directories (root kit)
- Unusual processes
- Promiscuous mode NIC
- Altered files (password, shadow, binaries)
- Suspicious (or non-existent) log entries

What Are We Looking For?

Basically, the idea at this stage is to take a quick look at the system—as non-invasively as possible—and try to find signs that something has gone wrong. Some good ideas of places to look are listed on this slide and we'll talk about a few of them in more detail in the upcoming slides.

© 2017 Hal Pomeranz and Deer Run Associates



ps Can Be a Good Start

A careful examination of the output from an appropriately formed `ps` command can show quite a few things. Obviously, this machine was booted on April 15 (look at the time on the `init` process), but many of the logging daemons were apparently restarted on the 23rd. Why? Do you recognize all of the processes? What about that suspicious "`(nfsiod)`" process? Are any using an extraordinary amount of memory? Knowing the processes that should be running on the OS versions in your organization can be useful.

© 2017 Hal Pomeranz and Deer Run Associates

Backdoor Evidence

```
COMMAND     PID  USER   FD   TYPE DEVICE SIZE NODE NAME
smbd       3137     0   6u   IPv4   4571       TCP *:2003 (LISTEN)
smbd       3137     0  16u   IPv4    976       TCP *:443 (LISTEN)
smbd       3137     0  17u   IPv4    977       TCP *:80 (LISTEN)
(swapd)    3153     0  16u   IPv4    976       TCP *:443 (LISTEN)
(swapd)    3153     0  17u   IPv4    977       TCP *:80 (LISTEN)
initd     15119     0   3u   IPv4  15617       TCP *:65336 (LISTEN)
initd     15119     0   5u   IPv4  15619       TCP *:65436 (LISTEN)
initd     15119     0   6u   IPv4  16157  TCP 192.168.1.79:65336->213.154.118.200:1188
initd     15119     0   9u   IPv4  15909  TCP 192.168.1.79:1146->199.184.165.133:6667
initd     15119     0  12u   IPv4  16191  TCP 192.168.1.79:1149->64.62.96.42:6667
xopen     25239     0   8u   IPv4   9972       UDP *:3049
xopen     25239     0  17u   IPv4    977       TCP *:80 (LISTEN)
xopen     25241     0   8u   IPv4  12302       TCP *:3128 (LISTEN)
xopen     25241     0  16u   IPv4    976       TCP *:443 (LISTEN)
lsn       25247     0  16u   IPv4    976       TCP *:443 (LISTEN)
lsn       25247     0  17u   IPv4    977       TCP *:80 (LISTEN)
```


Backdoor Evidence

This `lsof` output shows evidence of multiple backdoors. At a minimum, `lsn`, `xopen`, `swapd`, `initd`, and `smbd` are included in the list of possible backdoors because of the strange port numbers they're listening on.

© 2017 Hal Pomeranz and Deer Run Associates

Unusual Processes

NAME	PID	PORTS	WORKING DIRECTORY
smbd	3137	TCP 80, 443, 2003	/tmp/sand
(swapd)	3153	TCP 80, 443	/usr/bin
initd	15119	TCP 65336, 65436	/etc/opt/psybnc
xopen	25239	TCP 80, 443, UDP 3049	/lib/.x/s
xopen	25241	TCP ports 80, 443, 3128	/lib/.x/s
lsn	25247	TCP ports 80, 443	/lib/.x/s

Examination of **ps** and **lsof** output yields results summarized above...

Unusual Processes

```
140 S root       845     1  0  69   0    -   814 do_sel Aug09 ?          0:00 smbd -D
PWD=/ HOSTNAME=localhost.localdomain CONSOLE=/dev/console PREVLEVEL=N CONFIRM=
runlevel=3 MACHTYPE=i386-redhat-linux-gnu LANG=en_US SHLVL=2 previous=N
SHELL=/bin/bash HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/ TERM=linux
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin RUNLEVEL=3 INIT_VERSION=sysvinit-
2.78 _=/sbin/initlog

040 S root      3137     1  0  69   0    -   475 do_sel 13:33 ?          0:03 smbd -D
PWD=/tmp/sand HOSTNAME=localhost.localdomain MACHTYPE=i386-redhat-linux-gnu SHLVL=5
SHELL=/bin/false HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/ TERM=dumb
PATH=/usr/local/bin:/bin:/usr/bin _=/usr/bin/smbd -D

100 S root      3153     1  0  69   0    -   416 wait_f 13:33 ?          0:00 (swapd)
PWD=/usr/bin HOSTNAME=localhost.localdomain MACHTYPE=i386-redhat-linux-gnu
OLDPWD=/tmp/sand SHLVL=5 SHELL=/bin/false HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/
TERM=dumb PATH=/usr/local/bin:/bin:/usr/bin _=/usr/bin/(swapd)

040 S root     25239     1  0  69   0    -   470 wait_f 15:32 ?          0:00
/lib/.x/s/xopen -q -p 3128 PWD=/lib/.x/s HOSTNAME=localhost.localdomain MACHTYPE=i386-
redhat-linux-gnu SHLVL=4 SHELL=/bin/false HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/
TERM=dumb PATH=/usr/local/bin:/bin:/usr/bin _=/lib/.x/s/xopen OLDPWD=/lib/.x

040 S root     25241     1  0  69   0    -   472 do_sel 15:32 ?          0:00
/lib/.x/s/xopen -q -p 3128 PWD=/lib/.x/s HOSTNAME=localhost.localdomain MACHTYPE=i386-
redhat-linux-gnu SHLVL=4 SHELL=/bin/false HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/
TERM=dumb PATH=/usr/local/bin:/bin:/usr/bin _=/lib/.x/s/xopen OLDPWD=/lib/.x

140 S root     25247     1  0  69   0    -   417 wait_f 15:32 ?          0:00
/lib/.x/s/lsn PWD=/lib/.x/s HOSTNAME=localhost.localdomain MACHTYPE=i386-redhat-linux-
gnu SHLVL=4 SHELL=/bin/false HOSTTYPE=i386 OSTYPE=linux-gnu HOME=/ TERM=dumb
PATH=/usr/local/bin:/bin:/usr/bin _=/lib/.x/s/lsn OLDPWD=/lib/.x

040 S root     15119     1  0  69   0    -   574 do_sel 16:02 ?          0:00 initd
PWD=/etc/opt/psybnc HOSTNAME=sbm79.dtc.apu.edu LESSOPEN=|/usr/bin/lesspipe.sh %s
USER=root …
```

© 2017 Hal Pomeranz and Deer Run Associates

Memory

Memory analysis saves time, more effective

Memory and swap may contain details of:

- Process and network data
- Command history
- Encryption keys and passwords
- Recently accessed files and directories

Bad news: Linux kernel devs broke /dev/mem

- "**`dd if=/dev/mem`**" trick doesn't work on modern kernels
- Wietse's **`memdump`** command doesn't work either

Memory

Memory analysis can be enormously useful in an investigation. As we'll see, the information we got from command output (previous slides) can also be obtained from a memory dump. Since memory is analyzed on a different machine from the compromised system, it is less prone to interference from the attacker. Memory analysis may be the only way to obtain encryption keys necessary to unlock protected file systems.

On older Linux kernels you could simply use `dd` to dump `/dev/mem` to a file, but it is no longer possible to capture all of the memory in this way on modern Linux kernels (2.6.x kernel versions). This is because, in 2008, the Linux kernel developers deliberately changed the behavior of `/dev/mem` so that it did not have access to all of the memory (only the first 256MB now). The reason they cited for doing this is that full memory access to `/dev/mem` was frequently used by attackers in their exploits. See: *http://lwn.net/Articles/267427/*

Wietse Venema created a `memdump` that used to work on many different OS platforms. But due to the same changes in the Linux kernel, `memdump` doesn't work anymore either. Wietse has indicated that he's not motivated to fix `memdump` due to what he considers poor programming choices on the part of the Linux kernel developers, so it's likely that `memdump` will not work on Linux in the future.

© 2017 Hal Pomeranz and Deer Run Associates

You Put the LiME in the Kernel!

```
# cd /mnt/usb
# insmod lime-*.ko "format=lime path=/mnt/usb/mem.lime"
# ls -lh mem.lime
-r--r--r--. 1 root root 512M Dec  9 06:25 mem.lime
```

- Can also export memory over the network:
  - Use "**path=tcp:4321**"
  - Connect to selected port with netcat

You Put the LiME in the Kernel!

So, the trick to capturing Linux memory dumps is to install a new device that works like the old `/dev/mem` device did before the Linux kernel developers changed the behavior. Joe Sylve created LiME to facilitate memory extraction from Linux devices (including Android phones!).

The trick is that you have to compile the LiME kernel module for the specific version of the Linux kernel on the system where you want to grab the memory. You'd rather not do this on the system you're investigating because bringing the source code onto the machine and compiling it will modify the state of the system. Hopefully, you can find or build another system with the same kernel and compile your LiME module there. Another option to minimize your impact would be building the LiME module on a USB device connected to the system you're investigating. The LiME source code and documentation is available from *http://code.google.com/p/lime-forensics/*

Once you've built the module, you simply use `insmod` to load it into the running kernel. The "`format`" parameter says what type of memory dump to take. Your choices are "`lime`" and "`raw.`" The Volatility tool we'll be using later wants "`lime`" format. Other tools may only work with "`raw`".

The "`path`" parameter specifies where to dump the memory image. In the example on the slide, we're dumping the memory capture to a USB device we've plugged into the system. This is how we brought the LiME kernel module onto the machine, so we may as well write our memory dump here as well.

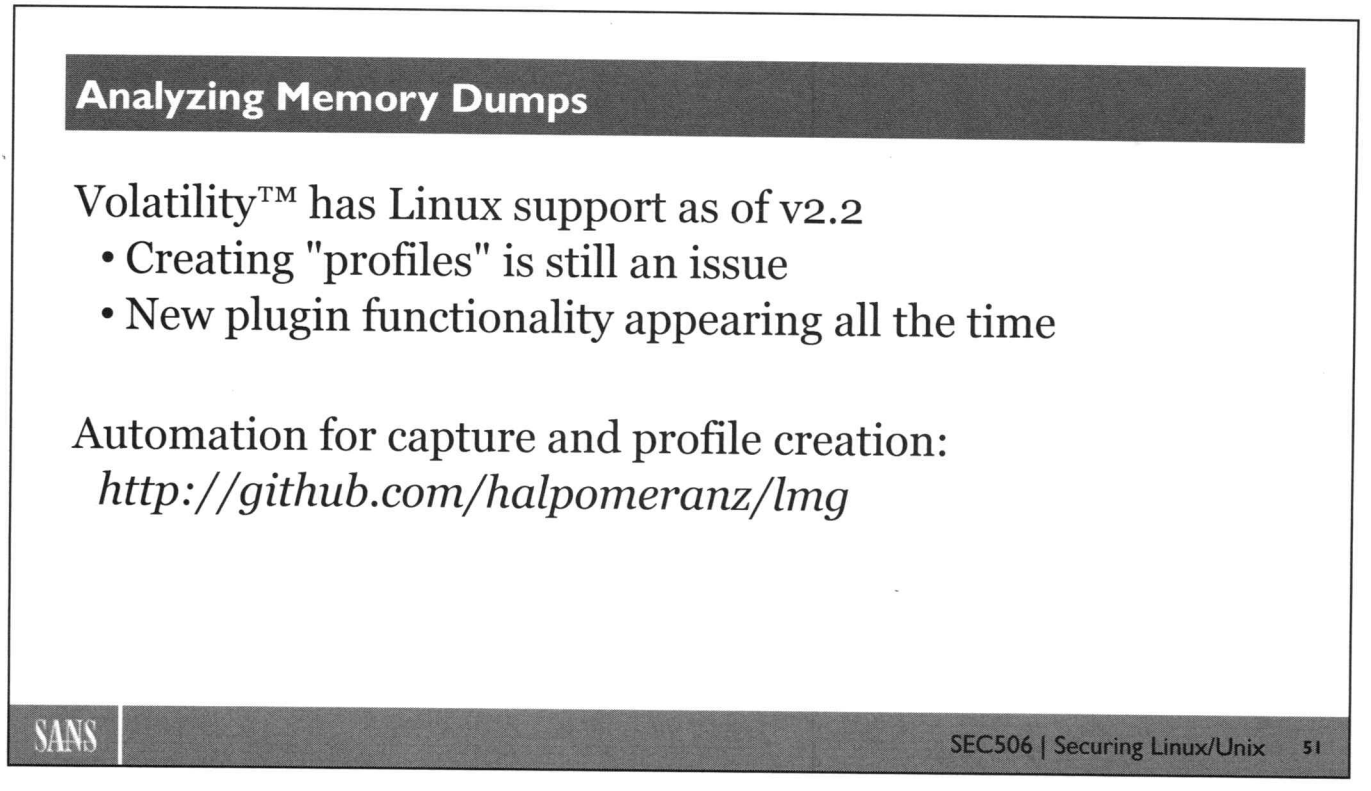
Instead of dumping to a file, you can also do "`path=tcp:`*<port>*", where *<port>* can be any available TCP port. Then from some other system, you would connect to this port using netcat ("`nc` *<ip>* *<port>* `>mem.lime`") and the LiME kernel module will deliver the contents of memory over the netcat connection.

© 2017 Hal Pomeranz and Deer Run Associates

Note that there are some other options besides LiME, though they all involve installing a new kernel module that creates a device with similar behavior to the old `/dev/mem` device. `fmem` is one popular option (*http://hysteria.sk/~niekt0/foriana/fmem_current.tgz*). Jamie Levy has ported Red Hat's "crash" driver to other Linux flavors and used it to dump memory (see *http://gleeda.blogspot.com/2009/08/devcrash-driver.html*).

If you're dealing with a virtual machine, obtaining a memory dump might be even easier. Some virtual machine technologies store their guest memory images in a memory-mapped file on the host OS. For example, you can find the memory image of a VMware virtual machine in the `*.vmem` file in the VMware directory.

© 2017 Hal Pomeranz and Deer Run Associates



Analyzing Memory Dumps

In the Windows forensics world, Volatility™ is a great tool for analyzing memory dumps. As of Volatility™ 2.2, Linux support is included in the mainline "stable" release. The major problem is that the wide variety of Linux kernel versions means that you need to create a "profile" for the specific kernel release that you're investigating. This process is documented at *http://code.google.com/p/volatility/wiki/LinuxMemoryForensics*. Future releases promise to make this easier, but basic functionality is there.

Hal Pomeranz has created a script for automating the process of capturing Linux memory and creating Volatility™ profiles for the target system. The script can be installed on a portable USB device and will capture all data to the device it is executed from. More details at *http://github.com/halpomeranz/lmg*

The command reference for Linux Volatility™ modules is at:

http://code.google.com/p/volatility/wiki/LinuxCommandReference22

For some examples of using this new Volatility™ functionality for Linux, see:

http://volatility-labs.blogspot.com/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html

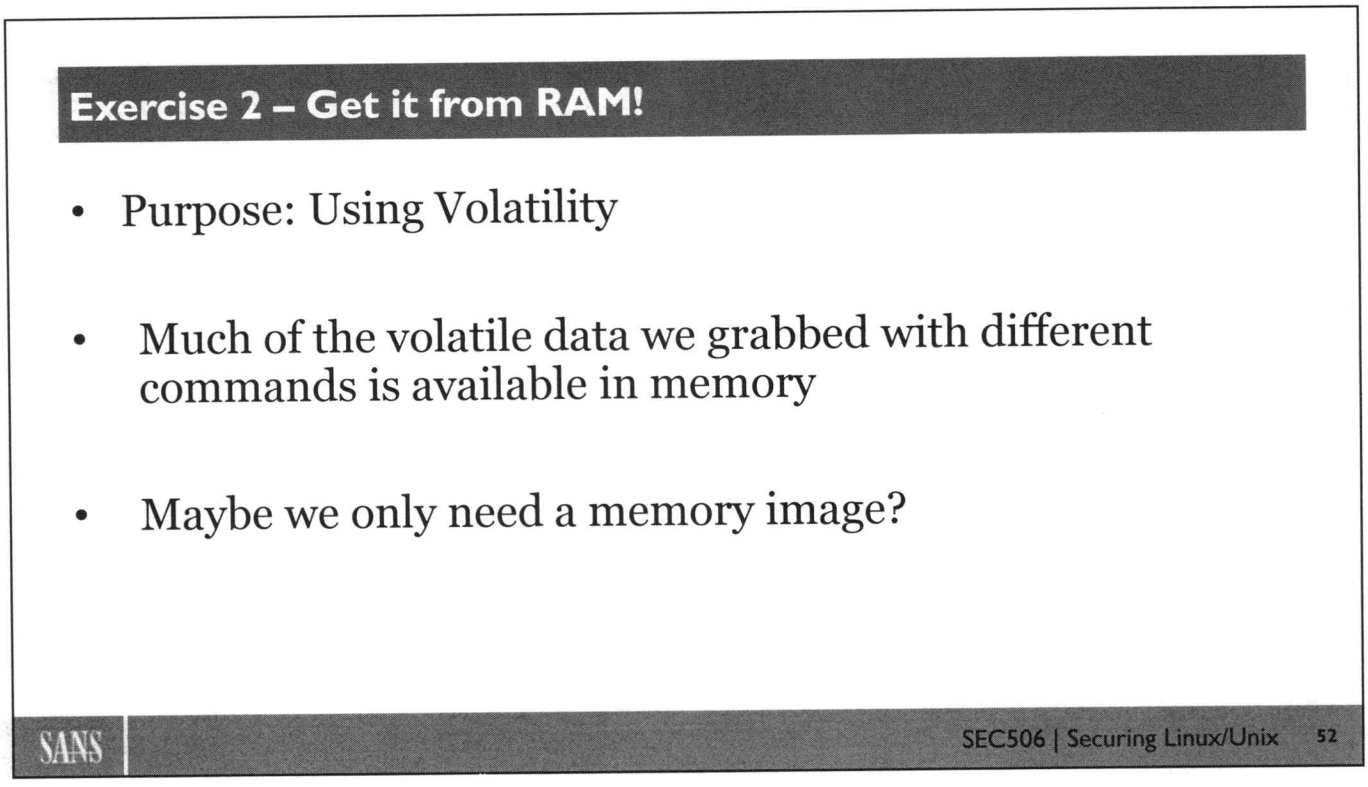
http://volatility-labs.blogspot.com/2012/09/movp-24-analyzing-jynx-rootkit-and.html

http://volatility-labs.blogspot.com/2012/09/movp-25-investigating-in-memory-network.html

http://volatility-labs.blogspot.com/2012/09/movp-35-analyzing-2008-dfrws-challenge.html

http://volatility-labs.blogspot.com/2012/10/phalanx-2-revealed-using-volatility-to.html

© 2017 Hal Pomeranz and Deer Run Associates



Exercise 2 – Get it from RAM!

Exercises are in HTML files in two places: on the course USB media, and in the home directory of the SANS user in the lab virtual machine. These two locations contain identical copies of the exercise files—it just depends on whether you prefer to look at them from a web browser inside the lab VM (because you're working in full-screen mode) or from your host operating system.

1. Open your web browser
2. Select "File… Open…" from the browser menu (Ctrl-O is the usual keyboard shortcut)
3. This is Day 6, Exercise 2, so navigate to *.../Exercises/Day_6/index.html* and open this HTML file
4. Click on the hyperlink which is the title of Exercise 2
5. Follow the instructions in the exercise

© 2017 Hal Pomeranz and Deer Run Associates

Definitely Compromised! Now What?

What to do once system is **known** to be compromised:

- Pull the power cord?
- Unplug network cable?
- Shutdown gracefully?
- Leave as is?

ANSWER: It depends on your policy...

Definitely Compromised! Now What?

First, stop and consider that while you've found one compromised system, there are likely others. If you give in to the knee-jerk reaction of taking this system down, you are signaling your adversary that you're on to them. This will cause your opponent to change their tools/techniques/practices so that they can remain embedded in your network. You may want to leave the compromised system operational while you look for the attackers throughout your enterprise.

If you encounter a live system imagine how many things will be lost if you pull the plug? Granted, you will retain more of a perfect image, a snapshot in time, of that hard drive, but you will also destroy evidence that may exist in system memory and the running processes of that system. Also, if full disk encryption is being used, you may find yourself unable to recover data from the "cold" system. You may also have situations where the attackers are using malware that is purely memory resident and leaves little or no footprint on the disk drives. If you shut down the system, you've lost your chance to capture and analyze that malware.

On the other hand, leaving the system running could give the attacker an opportunity to wipe the system or otherwise tamper with the evidence. Also, the system could potentially be part of an attack against other systems, either within your organization or at some external organization. What are your downstream liability issues if the system is used to attack another company?

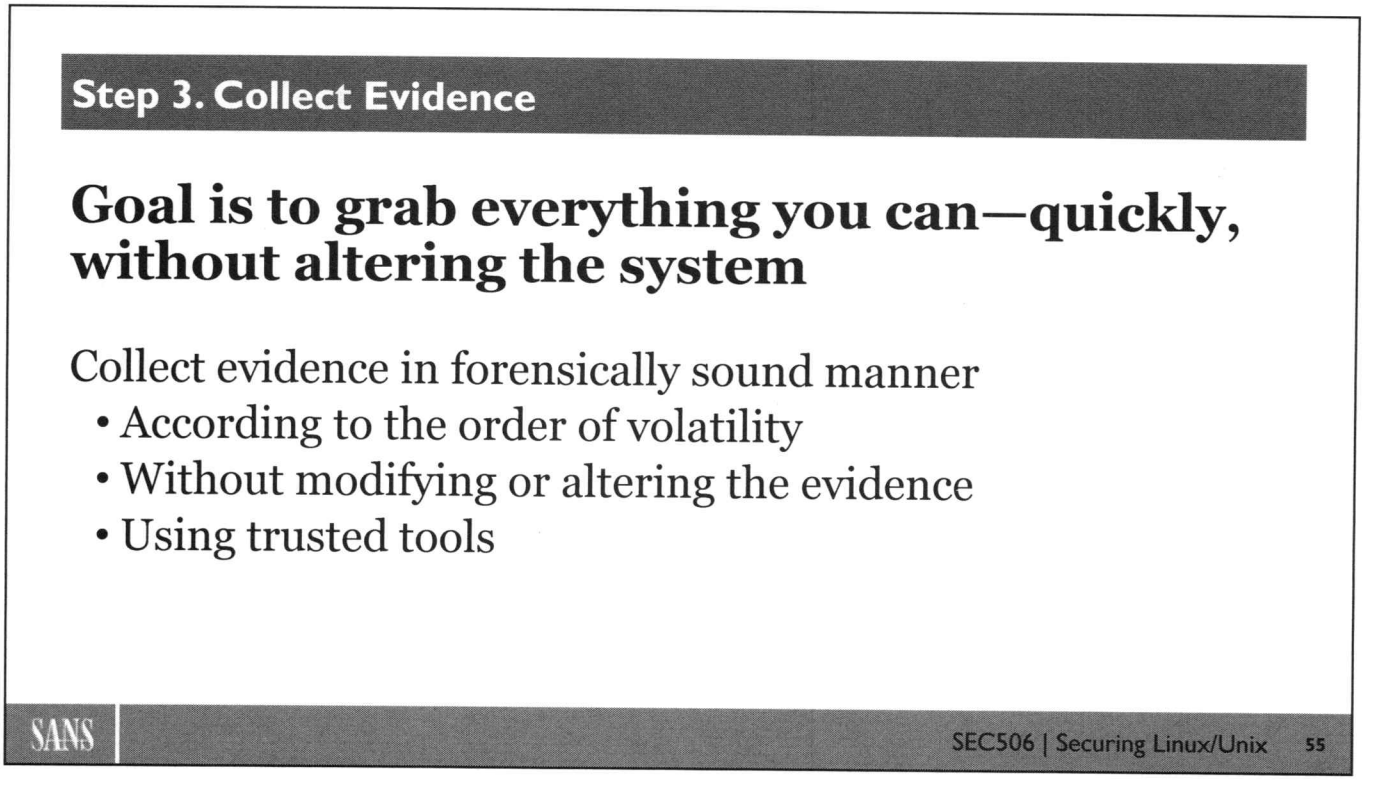
© 2017 Hal Pomeranz and Deer Run Associates

Evidence Collection

Evidence Collection

Armed with the basic tools and knowledge that you will need to succeed, we can now begin the discussion of evidence collection.

© 2017 Hal Pomeranz and Deer Run Associates

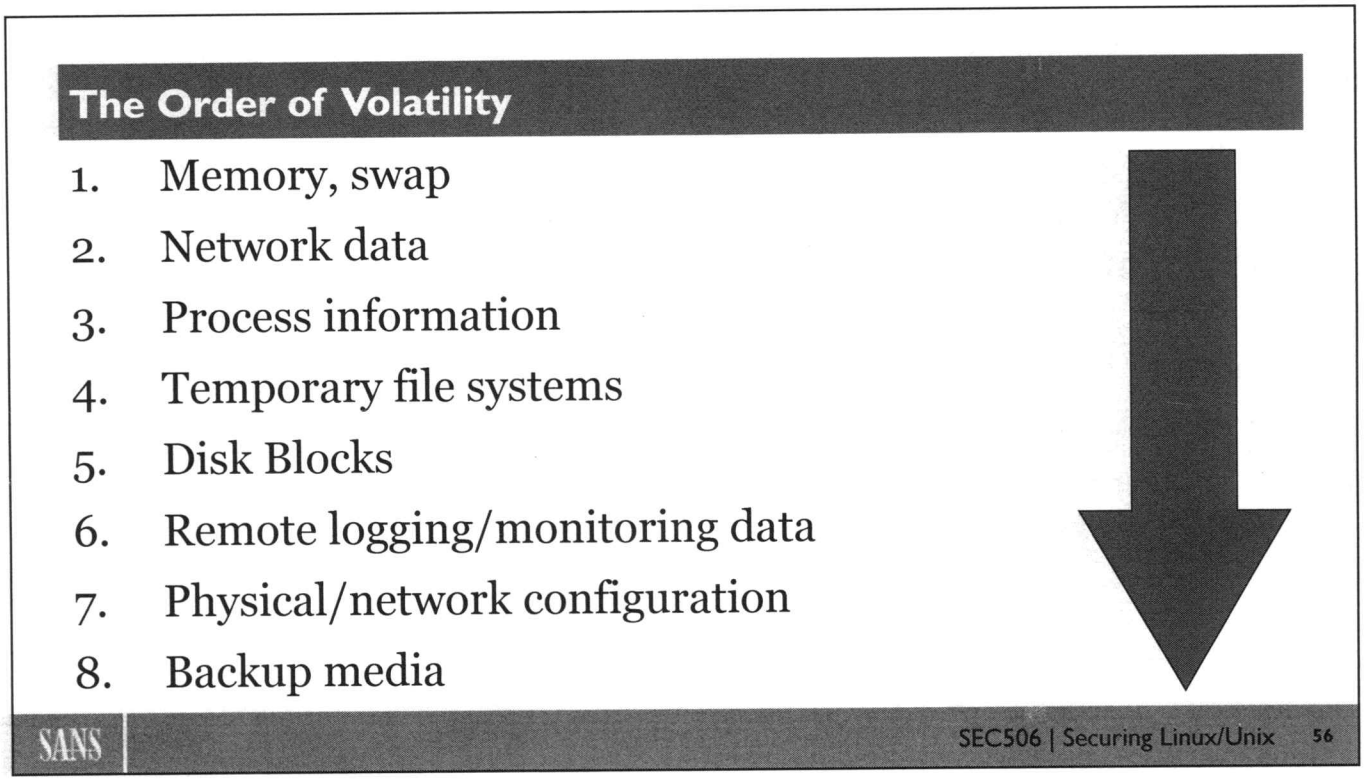


Step 3. Collect Evidence

Evidence is defined as anything that can be collected from the system under investigation. It does not necessarily have to be an image. It could include process information, network connections, log files, and user information. It is best to obtain evidence in as forensically sound a method as possible. This means trying to avoid the loss of evidence during any actions that you perform. Take your actions into consideration when you collect evidence and in what order it is collected.

Evidence collection is a step that needs to be performed and your results from this step need to be clear in your documentation. Not only do you need to describe the tool you used to collect the evidence, but you also need to describe how you ensure that the integrity of collected evidence has been maintained.

© 2017 Hal Pomeranz and Deer Run Associates



The Order of Volatility

Computer evidence is very volatile. You have to be very careful about completely understanding which evidence to gather and in what order. In some cases, gathering one piece of evidence will accidentally modify another.

In some examples, this may have to be OK as there is no way to completely take a snapshot of a running system at a moment in time. Nevertheless, collecting evidence according to the **order of volatility** is a defacto-standard approach. The idea is simple: some data on a system is more volatile than other data. You should work to collect the most volatile data first. In cases where you feel there may be a "tie", then collect the most important evidence first.

Undoubtedly, memory is the most volatile evidence on a system, followed by swap space. Swap exists on a system's media but is constantly changing. You have no guarantee what state swap will be in once the system is powered off, so you should gather it separately.

Network connections are important. What if the subject had a print queue or was in the middle of a secure shell session while you powered down? How would you know what was in the queue or even that there was one? How would you know what system he had tried to secure shell to? Is it guaranteed to be in a system log?

Running processes should also be gathered to testify as to the state of the system. Perhaps during your media analysis phase, you determine that the system scheduler had been Trojaned and had a sniffer wiretap wrapped into it. You now can check and see if that process was running at the time of seizure, who had executed it, and hopefully enough information to wrap an illegal wiretap in the list of potential charges.

Note: volatile evidence above is potentially destroyed if the system is powered off.

© 2017 Hal Pomeranz and Deer Run Associates

Collect Evidence Remotely!

Collect evidence remotely—avoid overwriting potential evidence on the local disk!

Can use **`netcat`** to transfer data:

- Copies stdin to host/port specified on command-line
- Also acts as a "listener" (server) on remote machine

Can use **`cryptcat`** if you want to encrypt data

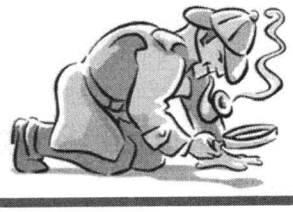
Collect Evidence Remotely!

Files are created in the course of collecting evidence. Although most evidence files are quite small, the potential exists for the creation of rather large files. One example might be the collection of system memory. If a system contains 8 gigabytes of physical memory, then the resulting capture will be of an equal size.

Evidence may exist in disk blocks that are no longer allocated to files. As a result of their unallocated state, these disk blocks may be reused at any time. It would be a shame if the collection of one set of evidence (i.e. collecting physical memory) were to cause the destruction of other evidence that might reside in these blocks. Therefore, it is important to create files that hold evidence on some disk other than the disks of the victim system. Said differently, the output of the investigation should be captured in files on another system.

There are (expensive) commercial tools that allow investigators to capture evidence remotely: F-Response, EnCase Enterprise, etc. For a cheap, low-tech solution, there's always netcat. netcat can act either as a network server or a network client. The investigator will start a netcat listener (server) on the analysis system, and then use netcat as a client to "pipe" the results of commands over the network to the listening server. If the system that is being investigated contains sensitive data then it might be wise to use cryptcat, an encrypting version of netcat (or tunnel the data over SSH).

© 2017 Hal Pomeranz and Deer Run Associates

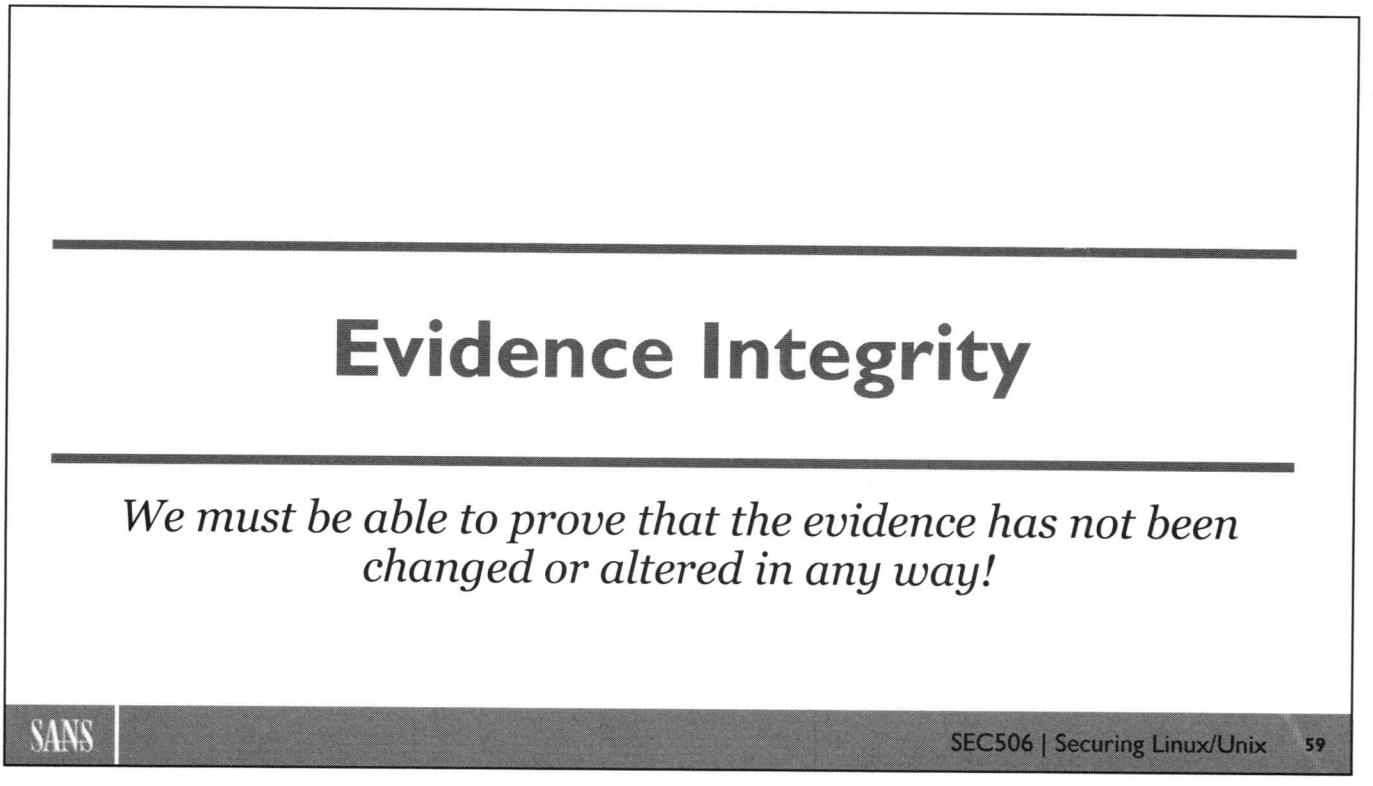


Disk Image Acquisition

Disk Image Acquisition

In this section, you will learn the basic techniques of gathering disk images and being able to prove their integrity.

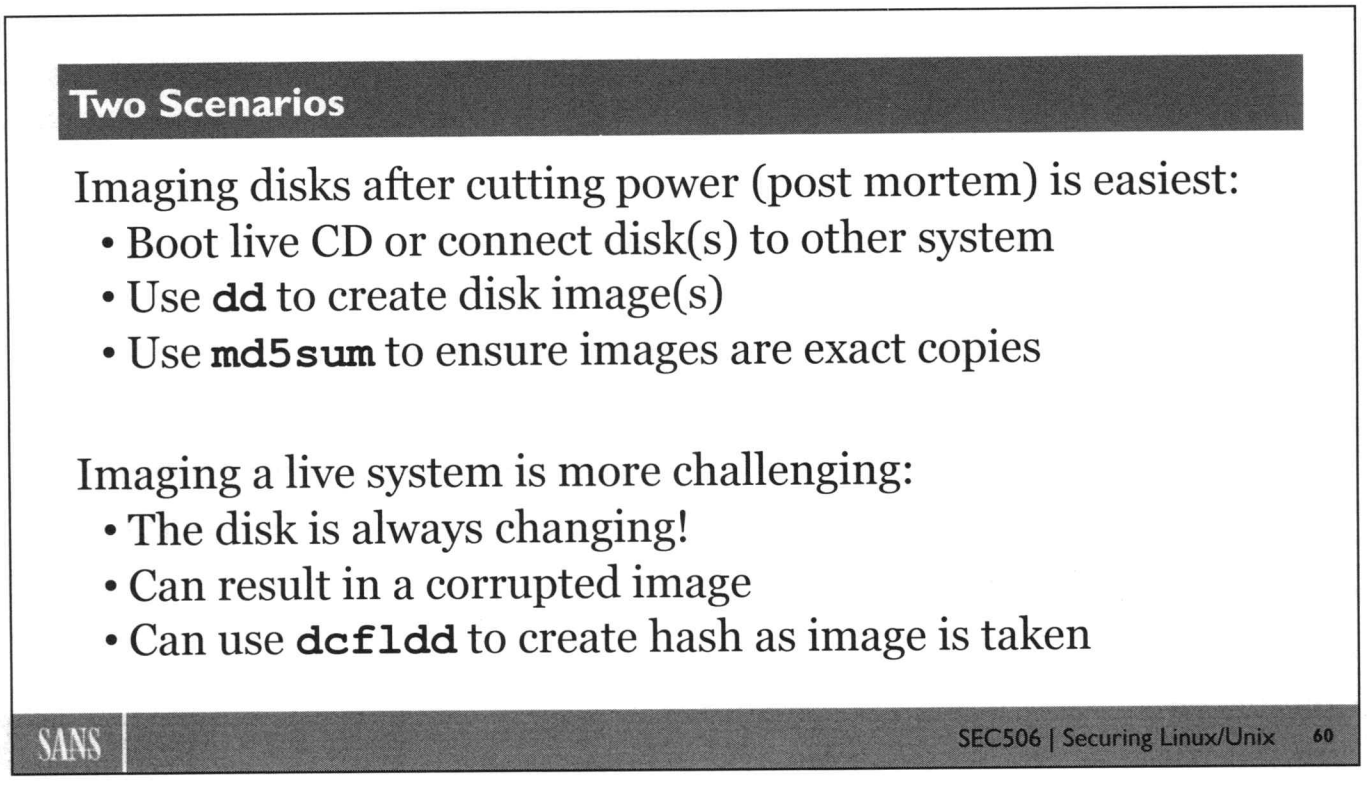
© 2017 Hal Pomeranz and Deer Run Associates



Evidence Integrity

When you take a disk image, you want to be able to verify later that the image hasn't been modified or corrupted since it was taken. Usually, this is done by comparing hash values. However, it's a slow process to image a multi-terabyte drive and then re-read the image file to create an initial hash. So, it's better to use a disk imaging tool that computes the hash value for the image while the image is being created. Commercial disk duplicators do this automatically. We'll also look at a software tool called `dcfldd` that can hash on the fly.

© 2017 Hal Pomeranz and Deer Run Associates



Two Scenarios

You will usually encounter one of two scenarios. Either the system administrator will have already killed the power to the system or you will be facing a live system with a changing file system. In the case of a post-mortem image collection, you've got it pretty easy. You may be able to physically remove the storage from the machine and connect it to a forensic disk duplicator, which would be the fastest copying option. Or you could connect the disk to another machine, being careful that the system you're plugging the disk into doesn't auto-mount the disk and change its state. If you're unable to remove the storage, you could boot the system using a Linux forensic distro like Paladin and image the disk that way.

However, occasionally you will run into the critical system that for whatever reason CAN NOT be powered down. Trying to collect a disk image while the file system is changing can result in a corrupted image. You may be limited to just doing a logical acquisition with a tool like `tar`.

If the system you are imaging is a virtual machine, consider using snapshots to gather disk images. If you're dealing with an enterprise class system with mirrored drives, you may be able to "break" the mirror and remove one of the drives for imaging.

© 2017 Hal Pomeranz and Deer Run Associates

Use dd for Post-Mortem Images

Can use **dd** to create bit-for-bit disk images:

```
dd if=/dev/sda of=- | nc 192.168.1.1 9000
```

I/O can be a file, disk device, partition or tape

Useful Options

bs=N	(sets the transfer size to N)
count=N	(copy only N blocks from input)
skip=N	(skip ahead N blocks in input)
conv=noerror,sync	(skip over unreadable sections)

Use dd for Post-Mortem Images

`dd` is a utility that reads input files block by block. If you specify a disk device, you can capture file system metadata. This includes unallocated blocks that could contain deleted file data. This data will be missed if you use a logical file system collection tool like `tar`. Note, however, that the images you capture will be as large as the total physical size of the drive, so we're talking about a lot of data here (`gzip` is your friend).

The input file for `dd` can be disk device name or any other type of device or file in the operating system. Here we're using "`of=-`" to send the output to stdout and then using netcat to relay the data across the network to another system for collection.

The "`conv=noerror,sync`" option is particularly useful when working with damaged media. "`noerror`" tells `dd` to simply skip bad blocks and "`sync`" says to replace the skipped block with a block of nulls in the output. Note that one problem with this approach is that if you have a 4K block size but only a single bad sector in the middle of the block, the `dd` command will skip the other 7 undamaged sectors and just output nulls. You could tell `dd` to read 512-byte sectors using the "`bs=512`" option, but this makes your image captures run MUCH slower.

© 2017 Hal Pomeranz and Deer Run Associates

dcfldd Is a Better Option

Improved **dd** with several enhancements:

- Collection status bar
- Dynamic MD5 integrity checks and logging
- Hashwindow defines hash "chunk" size in BYTES
  - It will perform an MD5 hash on every BYTES amount of data
  - Usually use **hashwindow=0** for single checksum over entire image

```
# dcfldd if=/dev/hda hashwindow=0 | netcat ...
```

SANS | SEC506 | Securing Linux/Unix 62

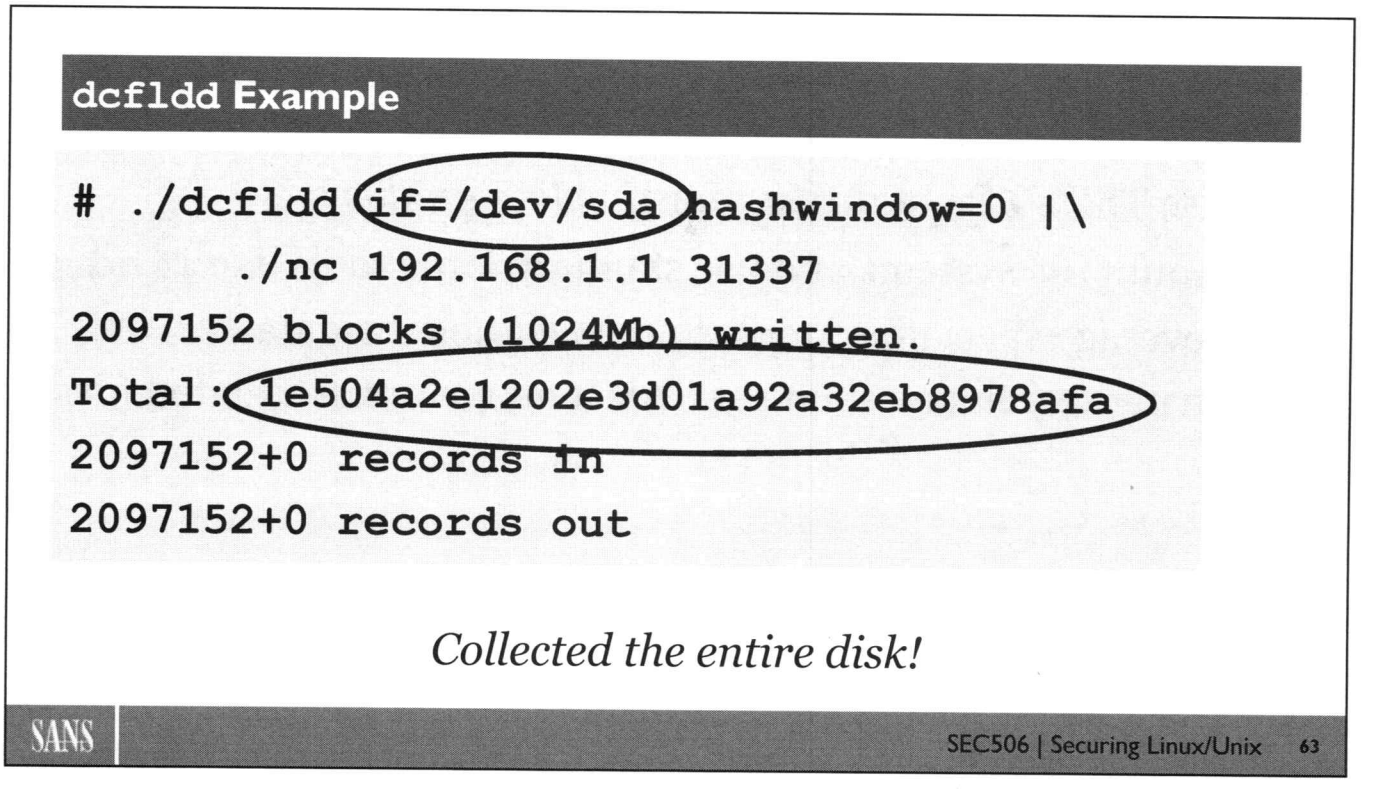
dcfldd Is a Better Option

`dcfldd` is a modified version of the `dd` tool that will provide for updates as well as provide the ability to performing hashing on the raw data that it is collecting (rather than having to collect the data and then run `md5sum` on it separately). It works and is implemented just like the normal `dd` tool but with two extra options. The first is to take a hash of the data collected which can be called (using `hashwindow=...`) as well as write that output to a designated logfile (`hashlog=filename`).

The `hashwindow` contains the minimum amount of data to grab before a hash is taken. Typically, the most common option is to set the size of the `hashwindow` to zero. This will take the hash of the entire data that was collected.

From this point on in the class, we recommend your using `dcfldd` to obtain any images gathered. However, we may refer to `dd` as a generic tool description for `dcfldd` or similar tools.

© 2017 Hal Pomeranz and Deer Run Associates



dcfldd Example

In this screenshot, we see the results of running `dcfldd` on a live system and piping the image through `netcat` to a remote system for collection. As this image was collected, the line just below the command dynamically counted the number of blocks and megabytes acting as a status bar. This helps the investigator because we can see progress. When the tool completes its collection of disk blocks, the hash of everything it read and sent across `netcat` is displayed. In this case, MAKE SURE to record (or take a screenshot) of this hash value. It is the value that you will use later to prove your image's integrity.

Notice that we imaged the entire disk across the network into one big file. This is certainly handy from a collection standpoint, as we don't have to fumble with `fdisk` and grabbing individual partitions. But if we collect the entire disk, then how do we examine the individual partitions?

© 2017 Hal Pomeranz and Deer Run Associates

What Can You Do with These Images?

- Use TSK's **`fls`**/**`mactime`** for *timeline analysis*
- Mount file systems and use standard Unix tools like **`find`**
- Leverage specialized tools to recover deleted data
- String search and then use TSK tools to find associated files

What Can You Do with These Images?

There are many advantages to creating images in a file rather than actually physically duplicating a disk. They are easier to handle, easier to hash, easier to distribute, easier to archive, and in general easier to work with.

Even though we've collected the entire disk as a single image, with a few special tools and tricks, we can work with them just as if they were actual partitions on a disk. That means that we can mount them into the file system (taking additional precautions to protect the integrity of these images). We can also use various TSK tools to analyze data in these images, as well as other specialized tools specifically designed for recovering deleted data.

We'll spend the rest of the course discussing all of these different approaches.

© 2017 Hal Pomeranz and Deer Run Associates

Use `mmls` to List Partitions

- A tool from the Open Source "Sleuthkit"
- **`mmls`** reads partition table, lists partitions
- Also displays the unallocated portions of the disk

Use `mmls` to List Partitions

In order to start working with our disk images, we need a tool to examine the image and tell us where the partition boundaries are. The tool `mmls` from the Sleuthkit (TSK) is useful in determining which partitions are located on a physical disk, the size, and the location in the physical disk image.

© 2017 Hal Pomeranz and Deer Run Associates

mmls Output

```
# md5sum dev_sda.dd
1e504a2e1202e3d01a92a32eb8978afa dev_sda.dd

# mmls -t dos dev_sda.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0001884159   0001884128   Linux (0x83)
03:  00:01   0001884160   0002097151   0000212992   Linux Swap (0x82)
```

- Check **md5sum** against **dcfldd** output
- We see Linux file system/swap plus some unused space

mmls Output

Here we see `mmls` being run on a disk image. The `Start` column shows the starting sector of each partition, the `End` column shows the final sector of the partition and the `Length` shows how many sectors are in the partition (which you could calculate using the other two values if you wanted to). We'll see how to use this information in later sections of the course.

Note that the partition table only uses one sector, but there are 31 sectors in between the partition table and the first Linux partition. Data or an old file system could be found there.

© 2017 Hal Pomeranz and Deer Run Associates

MAC Timelines

SANS | SEC506 | Securing Linux/Unix 67

MAC Timelines

The timeline is usually the bedrock of an investigation. Everything revolves around it. Often, the investigator will have an idea of the window of time in which the compromise occurred once basic evidence has been obtained and analyzed or log file analysis has been performed. The timing of events—when files were accessed, changed, modified will directly point to the events that occurred on the system.

© 2017 Hal Pomeranz and Deer Run Associates

File System Metadata

A HUGE amount can be discovered from the analysis of file attributes (inode data)

Of particular interest are MAC Times:

- ***Modification*** time: last time a file was written
- ***Access*** time: last time a file was read
- ***Change*** time: last time the inode contents were written
- ***Born-on*** time: file creation time (EXT4 only)

SANS | SEC506 | Securing Linux/Unix 68

File System Metadata

In terms of a timeline of events, we can learn a lot by looking at the MAC times associated with files. MAC stands for modification, access, and change. These times are recorded in the inode for each file and directory.

The modification time tells us when a file was last modified. This is helpful for determining what has changed since a given time. For example, I like to get a listing of all of the executable files (owned by root) that have changed since the OS was installed.

The access time tells us when a file was last accessed. This is sometimes helpful for determining which commands were run, and when. This can be enlightening, but it's not a history of commands because only the last occurrence of a file's access is recorded in the inode. So, if gzip is run twice, we will only be able to see the last time that is was run. Note, however, that modern Linux systems are starting to use the `relatime` option, which means that last access times are only updated if the previous atime was older than the current mtime or ctime. This means you can't necessarily trust atime information on file systems that are mounted with this option.

The (metadata) change time records the last time that the contents of an inode were written. The inode contains information such as permissions and ownerships.

EXT4 has added a file creation timestamp (usually referred to as the "born on" date or *btime*) indicating when the file was created on disk.

While MAC times are very useful in forensic investigations, be very careful. MAC times can be modified quite easily. Using the `debugfs` command, any one of these can be set to an appropriate time so as to fool the investigator. The `touch` command allows the root user to quickly modify any atime or mtime values.

© 2017 Hal Pomeranz and Deer Run Associates

MAC Time Data

MAC Times can be used to create a **timeline**.

Quickly locate additions and modifications to the file system:

- Command usage
- Configuration or log file alterations
- Root kits
- Program compilation
- Loadable kernel modules

Track attackers with "footprints" through the file system

MAC Time Data

Timelines are very beneficial if you are trying to track an attacker in a system. If you collect the timeline quickly enough after the incident, you should be able to easily see the actions of the attacker reflected in the MAC times of files in the file system. However, remember that the MAC times only retain the *last* access/modify/change time. Normal system activity can update these timestamps and inadvertently mask the attacker's activity.

It's like footprints on a beach: if you run down the beach early in the morning, it's easy to look back and see where you've been. But several hours later, other people's footprints will have stomped all over yours, making it harder to track you.

© 2017 Hal Pomeranz and Deer Run Associates

Step 4a. Create a Timeline

Use collected file system data *(and some cool tools)* to create a timeline of activity on the system based on MACB times

- Can be performed on live systems or partition images

Two-part process:

- Part 1, Grab inode metadata for the entire file system:
  - Allocated files, deleted file names, unallocated inodes
- Part 2, use mactime to create a time-ordered, formatted "timeline"
  - Can filter based on date and time or directory

Step 4a. Create a Timeline

The first step in making a timeline is to collect the MAC time data from all of the partitions in your image. Historically, this was called a "body file" by the `grave-robber` tool, and this name has stuck. TSK includes a tool called `fls` for collecting the MACtime information, including data from inodes associated with deleted files.

The second step is to take the data in the body file that has just been created (or a subset of it) and sort/format it. We will use another TSK tool called `mactime`. The subset can be determined by a date range (typically the dates of the incident) or by a directory of interest.

You could also use Kristinn Gudjonsson's log2timeline tool (*http://log2timeline.net/*), or the more recent version Plaso (*http://plaso.kiddaland.net/*), which handle both steps. At the moment, however, these tools are much more geared towards Windows systems than Linux machines.

© 2017 Hal Pomeranz and Deer Run Associates

Creating "Body Files"

```
# mmls -t dos dev_sda.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table
(#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0001884159   0001884128   Linux (0x83)
03:  00:01   0001884160   0002097151   0000212992   Linux Swap
(0x82)

# fls -o 32 -m / dev_sda.dd > dev_sda1.body
```

Note: "-m /" specifies the original mount point of the partition

Creating "Body Files"

`fls` is a TSK tool that will extract inode data and present it in a variety of formats. With the "`-m`" option it produces a dump format specifically designed to be consumed by the `mactime` tool we'll be using to produce the human-readable timeline format. Along with the "`-m`" option you need to specify the mount point where this file system is normally mounted (this information may be available in the output of `fsstat` or in the system `/etc/fstab` file from the image). `fls` will prepend this mountpoint information to each file path it outputs so that the resulting file name will be rooted at the correct location. In this simple example, we only have a single partition that's mounted on "`/`", but you can imagine having to run `fls` multiple times for different partitions in the disk image.

The other piece of information we need to run `fls` is the starting sector offset of the partition we want to dump. We get this information from the "`Start`" column in the `mmls` output. Use "`-o`" with any TSK tool to specify the starting sector offset.

© 2017 Hal Pomeranz and Deer Run Associates

mactime

Perl script that uses a body file as input

Cover the entire time range or select a range of dates

Flags include:

- b: Body file location (data file) (STDIN is default)
- p: password file location (to replace UID)
- g: group file location (to replace GID)
- y: Dates use the year first
- z: specify the timezone
- Date Range

mactime

TSK's `mactime` tool sorts the data file by time instead of directory and path name and produces clean, human-readable output.

The timeline contains columns for the user and group info. You will need to extract the `passwd` and `group` files from the disk image you collected in order to use them with `mactime`. If the password and group file are not given on the command line, then just the numerical ID is given.

© 2017 Hal Pomeranz and Deer Run Associates

mactime Examples

Run **mactime** on the entire data file:

```
# mactime -b dev_sda1.body \
     > timeline-dev_sda1-all
```

Run **mactime** starting at 2008:

```
# mactime -b dev_sda1.body 01/01/2008 \
     > timeline-dev_sda1-2008_onwards
```

View only the **/dev** directory:

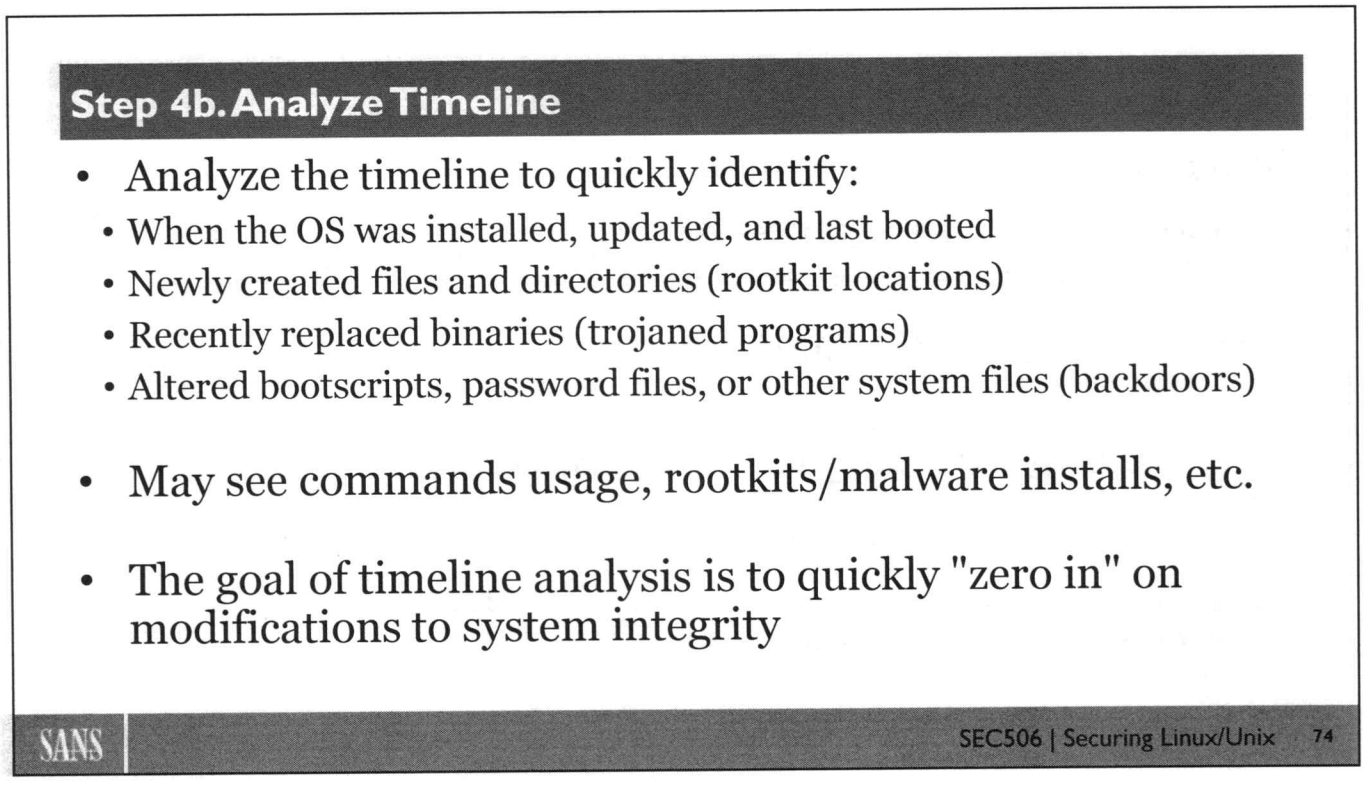
```
# grep /dev/ dev_sda1.body > dev_sda1-dev.body
# mactime -b dev_sda1-dev.body > tl-dev_sda1-dev
```


mactime Examples

The first example takes the `dev_sda1.body` file and makes a timeline of all data in it. The second example takes the same file, but only shows entries from Jan 1, 2008, and later. The last version extracts out entries from the `/dev` directory and makes a timeline of those files.

Looking at smaller timeline subsets is easier if you already know when the attack occurred or where the attacker was operating. When you're creating a report from your analysis, extracting just a subset of the timeline can make your report much more readable.

© 2017 Hal Pomeranz and Deer Run Associates



Step 4b. Analyze Timeline

Now that the timeline has been created, it can be used to quickly zero in on rootkits, trojan programs, and possibly even the commands that were used on the system and in which order.

© 2017 Hal Pomeranz and Deer Run Associates

Timeline Forensics—It Was FTP!

```
Oct 03 16:01:30     484 .a. -rw------- root  root  /etc/ftpaccess
                 153488 .a. -rwxr-xr-x root  root  /usr/sbin/in.ftpd
Oct 03 16:01:33     456 .a. -rw------- root  root  /etc/ftpconversi...
Oct 03 16:01:34     104 .a. -rw------- root  root  /etc/ftphosts
                     79 .a. -rw------- root  root  /etc/ftpusers
                   4096 mac -rw-r--r-- root  root  /var/run/ftp.pids
Oct 03 16:01:54   42736 .a. -rwxr-xr-x root  root  /sbin/ifconfig
                  11868 .a. -rwxr-xr-x root  root  /usr/bin/cut
Oct 03 16:01:55    3070 m.c -rw-r--r-- root  root  /etc/inetd.conf
                  10160 .a. -rwxr-xr-x root  root  /usr/bin/killall
                   8860 .a. -r-xr-xr-x root  root  /usr/bin/w
Oct 03 16:20:37   20452 m.c -rwxr-xr-x root  root  /bin/systat
```

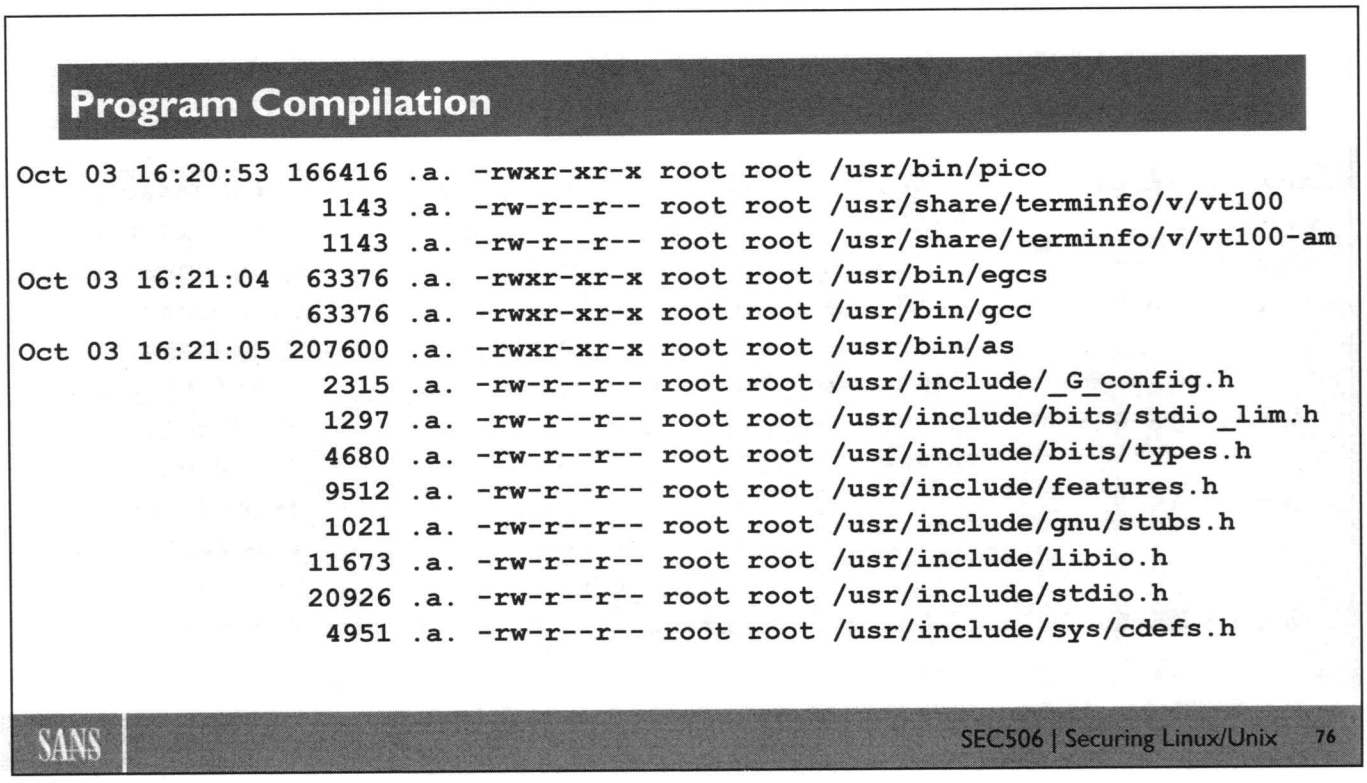

Timeline Forensics—It Was FTP!

Here is a sample of the `mactime` output. The output is sorted chronologically (look at the increasing timestamps in the first column). The second column gives file size. The third column indicates which of the MAC time values were modified. Then you see output that looks at lot like "`ls -l`".

Note that timestamps only have a one-second granularity. In the first interval shown on the slide, it's most likely that `in.ftpd` was executed first and it, in turn, read the `ftpaccess` file. But since these events happened in the same second, all `mactime` can do is list the files in alphabetical order. In other words, there is some interpretation on the part of the forensic investigator that must happen when reviewing the output of `mactime`.

This is the output from an intrusion that used the `SITE EXEC` remote buffer overflow in WU-FTPD to break into the computer system. You can see that immediately after `in.ftpd` was executed, weird things started happening on the system. The attacker edited `/etc/inetd.conf`—it turns out they added a remote "`sh -i`" to a high numbered port (you would review `inetd.conf` as soon as you saw that it had been modified). You can then see the attacker entered the system 20 minutes later and ran `systat` and a number of other commands documented on the following pages.

© 2017 Hal Pomeranz and Deer Run Associates



Program Compilation

You can clearly see `gcc` being executed and header files under `/usr/include` being accessed while a program is being compiled.

© 2017 Hal Pomeranz and Deer Run Associates

Oh Look! A New /bin/login ...

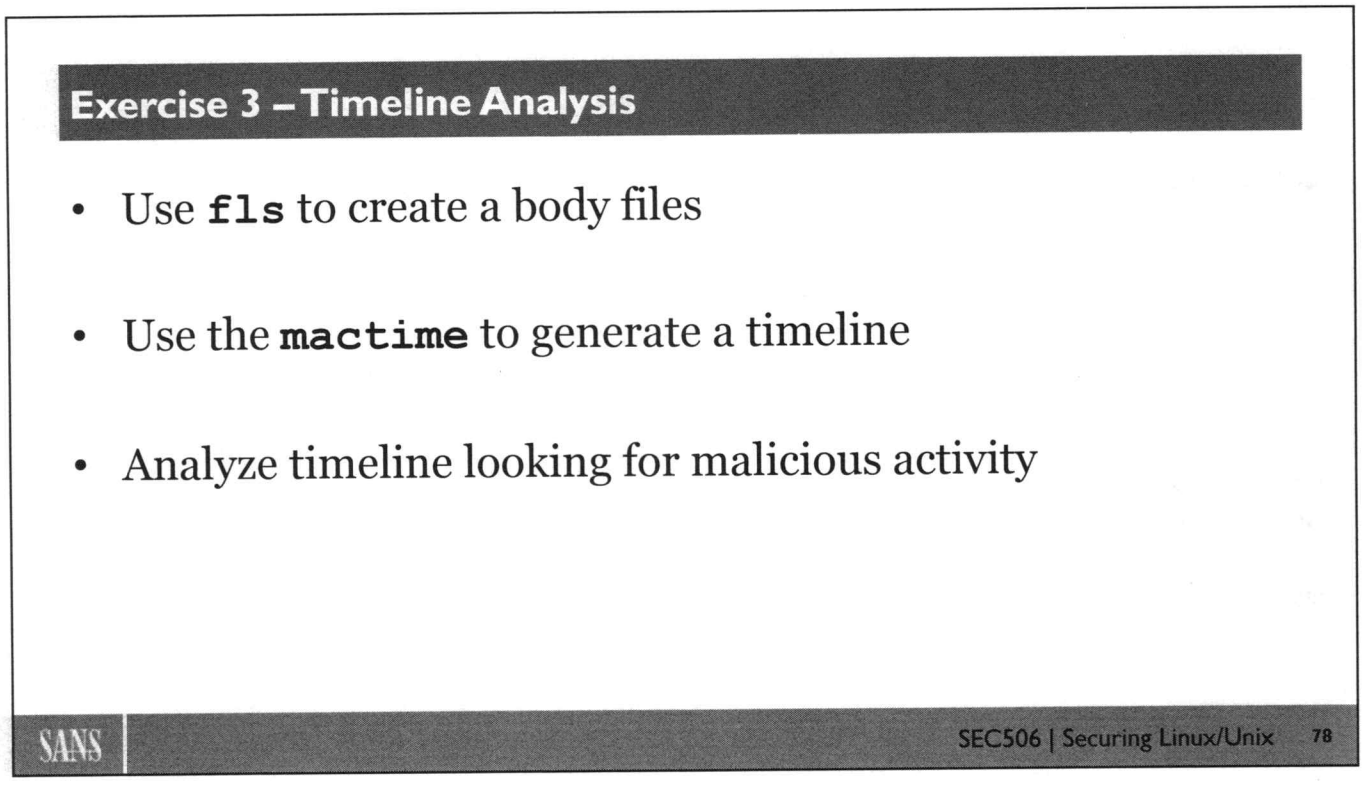
```
Oct 03 16:21:06  12343 m.c -rwxr-xr-x root  root  /bin/login
                205136 .a. -rwxr-xr-x root  root  /usr/bin/l
                  8512 .a. -rw-r--r-- root  root  /usr/lib/crt1.o
                  1124 .a. -rw-r--r-- root  root  /usr/lib/crti.o
                   874 .a. -rw-r--r-- root  root  /usr/lib/crtn.o
                314936 .a. -rwxr-xr-x root  root  /usr/lib/libbfd-2.9.5.so
                   178 .a. -rw-r--r-- root  root  /usr/lib/libc.so
                 69994 .a. -rw-r--r-- root  root  /usr/lib/libc_nonshared.a
Oct 03 16:21:08   4096 m.c drwxr-xr-x root  root  /bin
Oct 03 16:38:54   6196 .a. -rwxr-xr-x root  root  /bin/uname
                  5728 .a. -rwxr-xr-x root  root  /usr/bin/dirname
Oct 03 16:38:55  75600 .a. -rwxr-xr-x root  root  /bin/egrep
```


Oh, Look! A New **/bin/login** ...

`/bin/login` program was added to the system at 16:21:06. This is another case where `mactime` is showing a bunch of files that were accessed during the same one-second interval, but getting the order wrong. What we're seeing here is the `/bin/login` program being linked by the compiler—all of the `.o` and `.so` files are combined to create the final `/bin/login` image. Anyway, you would now go grab the `/bin/login` program out of your disk image and analyze the heck out of it to see if you can figure out what backdoors the attacker has added.

By the way, why aren't we seeing the C source code file that was compiled to create the `/bin/login` program? Most likely this file was later deleted by the attacker to cover their tracks.

© 2017 Hal Pomeranz and Deer Run Associates



Exercise 3 – Timeline Analysis

Exercises are in HTML files in two places: on the course USB media, and in the home directory of the SANS user in the lab virtual machine. These two locations contain identical copies of the exercise files—it just depends on whether you prefer to look at them from a web browser inside the lab VM (because you're working in full-screen mode) or from your host operating system.

1. Open your web browser
2. Select "File … Open …" from the browser menu (Ctrl-O is the usual keyboard shortcut)
3. This is Day 6, Exercise 3, so navigate to *.../Exercises/Day_6/index.html* and open this HTML file
4. Click on the hyperlink which is the title of Exercise 3
5. Follow the instructions in the exercise

© 2017 Hal Pomeranz and Deer Run Associates

Analyze Media

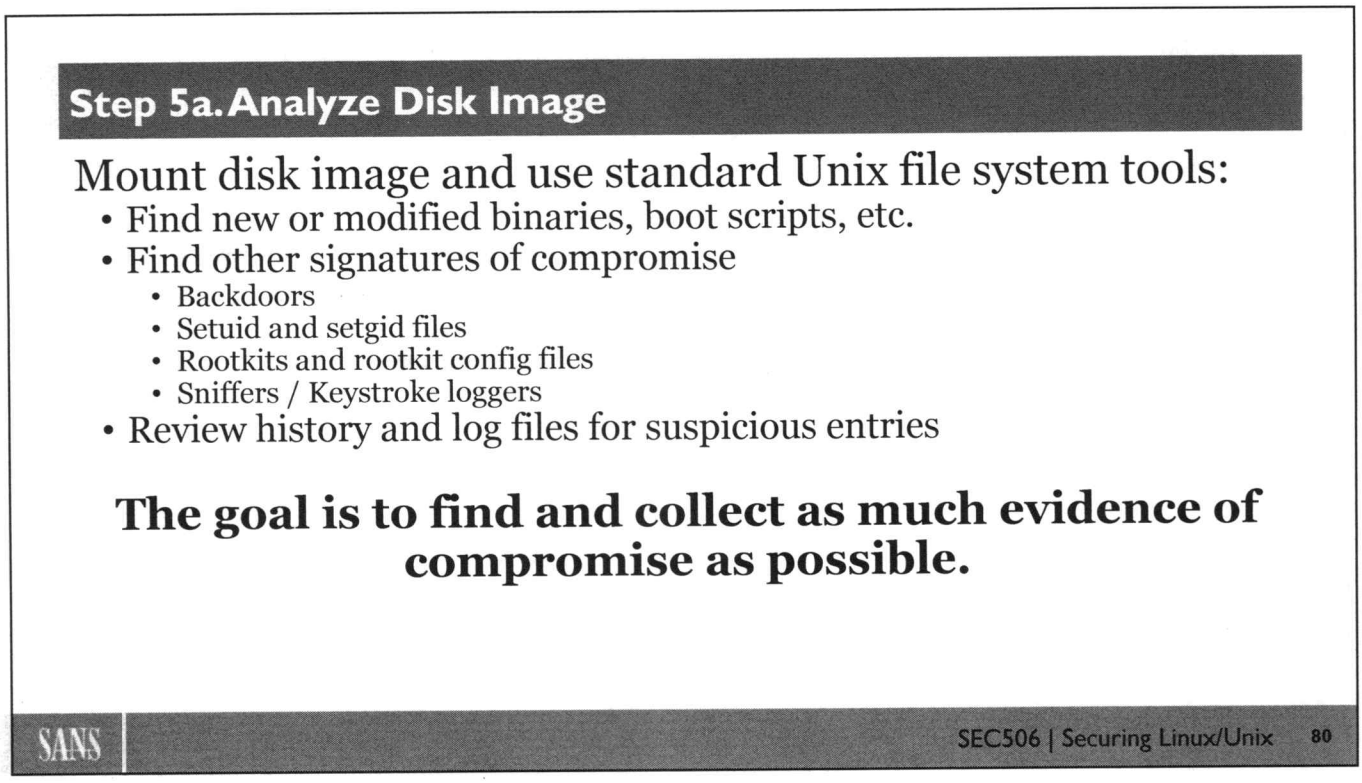
Analyze Media

Media analysis is an art unto itself, and this is where your existing experience with Unix and Linux become really important. Through your investigation, you will have collected a lot of data, and have a lot of clues about where trojans, configuration files, and malware data have been placed. It is time to start running these clues to ground, examining each of them in detail looking for further supporting evidence.

For example, let's say that using the timeline that you created, you determine that the `ps` command has been trojaned. Usually, trojans like these require a configuration file that dictate which process names should be hidden. Yet, at this point, you haven't found a configuration file. This is the point where you examine the `ps` trojan with strings to look for a path/filename that might indicate the configuration file.

There are many other file system modifications that could have been done. The key here is that we can use file system aware tools to start examining existing files on the disk image. Furthermore, this is the point in time that you need to start considering what evidence might exist that resides in unallocated portions of disk space. The recovery of evidence from these portions of the disk is also a critical step. More often than not, some sort of evidence can be found there.

© 2017 Hal Pomeranz and Deer Run Associates



Step 5a. Analyze Disk Image

During media analysis, you will perform many different tasks. While it would be too difficult to list every action here, some of the actions that are recommended are:

1. Examine file system for modification to operating system software or configuration
2. Examine file system for backdoors, check for setuid and setgid files
3. Examine file system for any sign of a sniffer program
4. Shell history files
5. Show start-up files and processes

© 2017 Hal Pomeranz and Deer Run Associates

Loopback File System Mounts

- Can mount image file as if it were a physical disk partition
- Can specify various file system types
- Options to preserve integrity of disks

Loopback File System Mounts

Once you have created a raw disk image, you will need to be able to examine it. Remember you can use `dd` or `dcfldd` to gather a raw image from a variety of different file systems. Linux already has the ability to read and interact with most file system types.

`mount` is the command that will take the raw image and mount it onto a specified directory of choice to be able to examine the contents of that image. The image has to be a recognizable file system. You cannot mount swap, for example, this is just memory storage space and is not in any recognizable file system format.

© 2017 Hal Pomeranz and Deer Run Associates

Using `mount` on Disk Images

Useful **`mount`** Options

`-t`	file system type (**`ext3`**, **`ext4`**, **`ntfs`**, **`vfat`**, etc.)
`loop`	use a loop device (used for image files)
`offset`	offset (***in bytes***) from start of image
`ro`	mount as read only
`noexec`	files from mounted partitions are not executable

Have to do some math to calculate **`offset`** ...

Using `mount` on Disk Images

The `mount` command can be used to mount the images that you have created and protect them from alteration. In many cases, the `mount` command can automatically discover the file system type but otherwise simply provide the file system type with the `-t` flag.

The `loop` option enables to mount a file system image as if it were a regular disk device. It is also a good idea to include the `ro` option to mount read-only, and the `noexec` option so that there is no chance that malware might get executed on the analysis system.

Similar to the "`-o`" option we saw earlier with `fls`, the `mount` command has an `offset` option to specify an offset corresponding to the beginning of our partition inside a full disk image. However, the `mount` command uses *bytes* as the default unit for its offset options, and not 512-byte sectors like the TSK tools do. That means we'll have to do a little math to figure out the right `offset` value. More on this on the next slide.

© 2017 Hal Pomeranz and Deer Run Associates

Mounting Partitions

```
# mmls -t dos dev_sda.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0001884159   0001884128   Linux (0x83)
03:  00:01   0001884160   0002097151   0000212992   Linux Swap (0x82)

# expr 512 \* 32
16384

# mount -o ro,noexec,loop,offset=16384 dev_sda.dd /mnt/hacked
```


Mounting Partitions

At the top of the mmls output, we see that mmls reports in terms of 512-byte sectors. So, when mmls says the Linux partition starts 32 sectors into the image, that's actually 32 * 512 = 16384 bytes into the image. This is the value we need to feed into the offset option of the mount command.

We also need the loop option so we can mount our image file as if it were a normal disk device. And we use ro (read-only) to ensure we don't accidentally modify our image. noexec protects us from accidentally executing malware from the image.

This is the simplest possible case you will encounter when trying to mount images. Modern Linux systems may use full-disk encryption and will likely be using logical volume management (LVM) instead of basic disk partitions. More information on these issues is covered in:

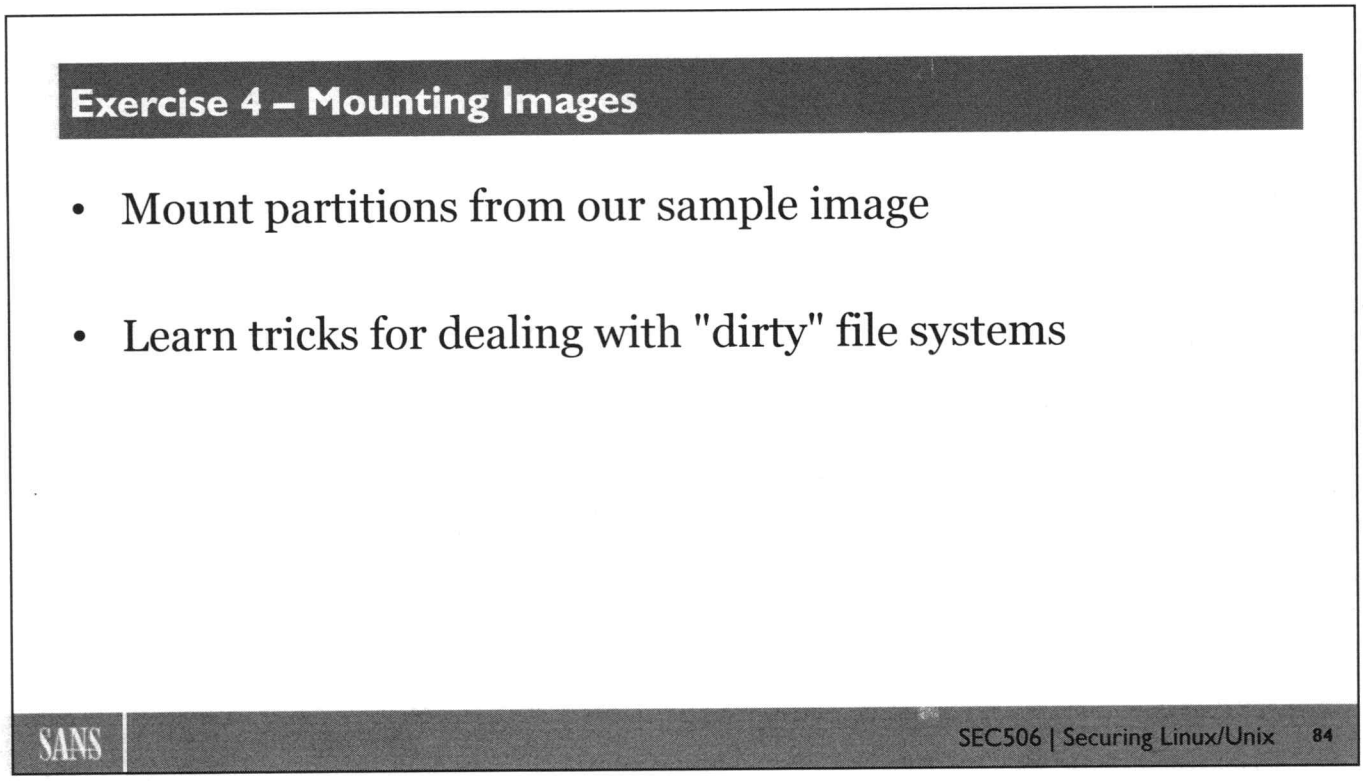
http://computer-forensics.sans.org/blog/2010/10/06/images-dmcrypt-lvm2/

http://deer-run.com/~hal/CEIC-dm-crypt-LVM2.pdf

You may also have to deal with "dirty" file systems—file systems that were not unmounted properly and which try to force journal recovery before mounting. We'll cover that topic in the next exercise, but more information can be found at:

http://computer-forensics.sans.org/blog/2011/06/14/digital-forensics-mounting-dirty-ext4-filesystems

© 2017 Hal Pomeranz and Deer Run Associates



Exercise 4 – Mounting Images

Exercises are in HTML files in two places: on the course USB media, and in the home directory of the SANS user in the lab virtual machine. These two locations contain identical copies of the exercise files—it just depends on whether you prefer to look at them from a web browser inside the lab VM (because you're working in full-screen mode) or from your host operating system.

1. Open your web browser
2. Select "File … Open …" from the browser menu (Ctrl-O is the usual keyboard shortcut)
3. This is Day 6, Exercise 4, so navigate to ... */Exercises/Day_6/index.html* and open this HTML file
4. Click on the hyperlink which is the title of Exercise 4
5. Follow the instructions in the exercise

© 2017 Hal Pomeranz and Deer Run Associates

Basic System Profiling

Linux distro name/version number:

`/etc/*-release`

Installation date:

Look at dates on `/etc/ssh/ssh_host_*_key` files

Computer name:

`/etc/hostname` (also log entries under `/var/log`)

IP address(es):

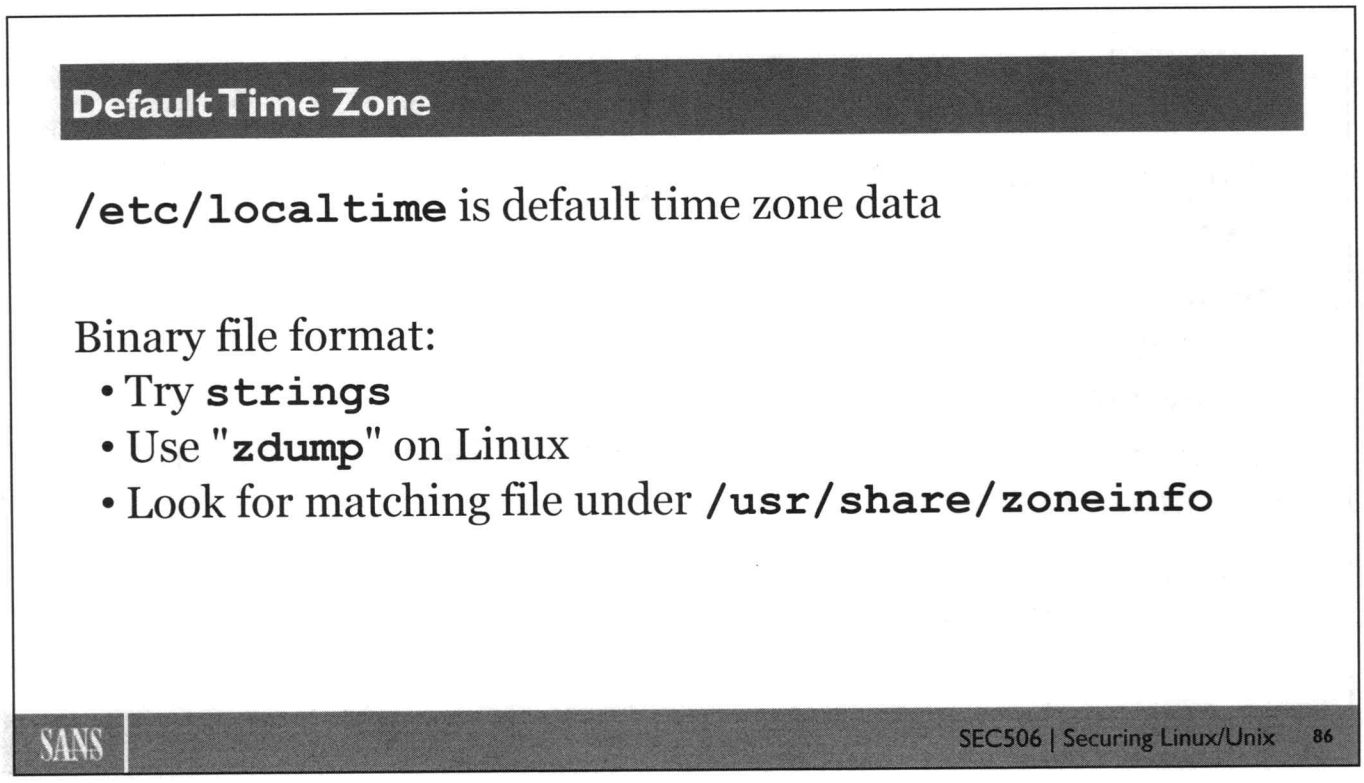
`/etc/hosts` (static assignments)

`/var/lib/dhclient, /var/log/*` (DHCP)

Basic System Profiling

Now that we have our disk mounted, the next step is to gather basic information about the host. This data includes items like the name of the host, the OS version, its IP address, etc. This information will end up in your final report but is also important for analyzing the system. Different artifacts are found in various directories depending on factors like the OS type, etc.

© 2017 Hal Pomeranz and Deer Run Associates



Default Time Zone
The system's time zone is configured in `/etc/localtime`. This is a binary file requiring special software for interpretation.

If you're doing your analysis from a Linux system, you can use the `zdump` utility to obtain the time zone name:

```
# zdump /mnt/hacked/etc/localtime
/mnt/hacked/etc/localtime  Sat Mar 16 07:19:34 2013 PST
```

The "PST" at the end of the line indicates US Pacific time.

Another approach is to `md5sum /etc/localtime` and look for matching files under `/usr/share/zoneinfo`:

```
# md5sum /mnt/hacked/etc/localtime
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/etc/localtime
# find /mnt/hacked/usr/share/zoneinfo -type f | xargs md5sum |
    grep 7c7836c1a47b82d402e88495c6001abf
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/America/Los_Angeles
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/SystemV/PST8PDT
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/US/Pacific
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/posix/SystemV/PST8PDT
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/posix/US/Pacific
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/posix/PST8PDT
7c7836c1a47b82d402e88495c6001abf  /mnt/hacked/usr/share/zoneinfo/PST8PDT
```


© 2017 Hal Pomeranz and Deer Run Associates

User Accounts

- Basic user data in **/etc/passwd**
  Any UID 0 account has admin privs
- Password hashes in **/etc/shadow**
  (brute force with "John the Ripper")
- **/etc/sudoers** shows users with admin privs
- Group memberships in **/etc/group**

User Accounts

Local user accounts are listed in `/etc/passwd`. Any account that has user ID zero (third column) has administrative privileges. Try sorting the password file with "`sort -t: -k3 -n /etc/passwd.`" This will cause UID 0 accounts (that may have been added in the middle of the file) to "bubble up" to the top of the output.

Administrative privileges may also be granted via the `sudo` command, which is configured in `/etc/sudoers`. The `/etc/sudoers` configuration often relies on group memberships that are configured in `/etc/group`.

Unix stores passwords for local user accounts in `/etc/shadow`—Linux uses salted hashes to hide the passwords. These passwords can be brute-forced using a variety of password cracking tools, including "John the Ripper" (*http://openwall.org/john/*).

© 2017 Hal Pomeranz and Deer Run Associates

User Login History

/var/log/wtmp

- Shows source, time, and duration of login
- Need to use Linux "**last**" command to view

Other logs that may be useful:

- **/var/log/auth.log**
- **/var/log/secure**
- **/var/log/audit/audit.log**

User Login History

A chronological history of user logins can be found in `/var/log/wtmp`. Use the `last` command to view the contents of this log:

```
# last -if /var/log/wtmp
hal      pts/0        0.0.0.0          Sat Mar 16 07:37   still logged in
root     tty1         0.0.0.0          Sat Mar 16 07:37   still logged in
reboot   system boot  0.0.0.0          Sat Mar 16 07:36 - 08:32  (00:56)
hal      pts/1        0.0.0.0          Tue Feb 19 10:00 - 10:00  (00:00)
hal      pts/0        0.0.0.0          Tue Feb 19 09:59 - down   (1+00:53)
root     tty1         0.0.0.0          Tue Feb 19 09:59 - down   (1+00:53)
reboot   system boot  0.0.0.0          Tue Feb 19 09:58 - 10:53 (1+00:55)
hal      pts/0        0.0.0.0          Tue Feb 19 09:30 - crash  (00:27)
root     tty1         0.0.0.0          Tue Feb 19 09:29 - crash  (00:28)
reboot   system boot  0.0.0.0          Tue Feb 19 09:26 - 10:53 (1+01:26)
hal      pts/1        0.0.0.0          Tue Feb 19 09:15 - down   (00:11)
laura    pts/0        192.168.108.1    Sat Sep  8 14:47 - down   (00:22)
reboot   system boot  0.0.0.0          Sat Sep  8 14:46 - 15:10  (00:23)
laura    pts/0        192.168.108.1    Sat Sep  8 14:44 - down   (00:02)
reboot   system boot  0.0.0.0          Sat Sep  8 14:43 - 14:46  (00:02)
laura    pts/0        192.168.108.1    Sat Sep  8 14:30 - down   (00:12)
```

If the attacker removes `/var/log/wtmp`, there may be other logs that show user login activity on the system.

© 2017 Hal Pomeranz and Deer Run Associates

Commonly Targeted Files

Authentication-related:

- **/etc/passwd, /etc/shadow**
- **authorized_keys**

Can be used to start processes:

- Unit Files (**/etc/inittab**, Boot scripts)
- **/etc/[x]inetd.conf, /etc/xinetd.d**
- Cron scripts and crontabs
- Web shells

Commonly Targeted Files

So, what should we look for in the file system of the compromised image? Generally, attackers will be looking to set up backdoors to allow them to access the system. This means either setting up standard account access for themselves or running processes that give them a back door.

First, validate the passwd and shadow files for extraneous or mangled accounts, odd defaults shells and the like. Attackers will often create extra accounts with a UID 0, or enable blocked administrative accounts in an attempt to leave behind an easy backdoor.

Another common target these days is the `authorized_keys` file that holds the public half of identity certificates used to access the system remotely via SSH. If the attacker can introduce a new public key into this file, then they will be able to access the system remotely whenever they wish (similar to older attacks against `.rhosts` files). Obviously, the `authorized_keys` file for the `root` account is the highest value target for the attacker, but even unprivileged access to a system can be valuable.

Look for new or modified unit files in the `systemd` configuration directories. Or the `inittab` file and system boot scripts (for older, `init`-based systems), `inetd/xinetd` configuration files, and even crontab files are all places where an attacker can create entries that allow them to create backdoors onto the system. For example, an attacker who can write to `inetd.conf` could add a line like:

```
tftp stream tcp nowait root /bin/sh sh -i
```

The attacker waits for `inetd` to be restarted, `telnet`s to the TFTP port and they have a root shell on the system.

© 2017 Hal Pomeranz and Deer Run Associates

Other Items to Look For

- Log files and directories
- User command history files
- Any directory beginning with " . "
- Regular files in **/dev**
- SUID/SGID files
- Recently modified binaries
- Recently created files (per timeline)

Other Items to Look For

Be sure to review the contents of your logs and the command history of any user accounts that may have been involved in the incident.

Look for any hidden directories that begin with a ".", some results will be valid directories others may not. You need to take a look at each of these and validate them. This type of search can be expanded to include hidden files but will result in many more false positives.

More often than not, attackers will hide trojan rootkits in the `/dev/` directory. This is because there are so many files listed there. They have names like `sdz1` to `sdz9` so it can be hard to spot the files that do not belong. So, make sure to scrutinize regular files in `/dev`. Most of the entries in this directory are devices. Regular files in this directory are worth a good hard look.

Sometimes crackers use setuid/setgid files to elevate privileges to root without accessing that account. SetUID files run in the context of the owner of the file. Imagine an editor, owned by root that has been setUID. Any user on the system can use this editor to modify or read any file.

Have any of the binaries in the file system been modified or created recently? Do any of them appear to have been "back-dated"? A good, hard look at the MAC times of binaries can tell you a lot about what has occurred on the system.

For more ideas, see Ed Skoudis' highly-useful "Intrusion Discovery Cheat Sheet", available from *http://www.sans.org/score/checklists/ID_Linux.pdf*

© 2017 Hal Pomeranz and Deer Run Associates

Me and My Shadow (File)

```
root:$1$gm64oWDG$/W3MX0Pb7/2oCB7Jkyvga1:12270:0:99999:7:::
bin:*:12247:0:99999:7:::
daemon:*:12247:0:99999:7:::
adm:*:12247:0:99999:7:::
..snip..
gopher:*:12247:0:99999:7:::
ftp:*:12247:0:99999:7:::
admin:$1$YAkCbk.7$JoZPsqqGxO.ImKonKAucm.:12248:0:99999:7:::
nobody:*:12247:0:99999:7:::
mailnull:!!:12247:0:99999:7:::
rpm:!!:12247:0:99999:7:::
ident:!!:12247:0:99999:7:::
apache:!!:12247:0:99999:7:::
```


Me and My Shadow (File)

It is very common for attackers to add new accounts or enable deactivated accounts on systems that they have compromised. Although a few of the accounts have been snipped from this screenshot for brevity, one account of interest stands out: the admin account. This account didn't previously exist on the system and has suddenly appeared. When this was recombined with the password file and John the Ripper run against it, the results are amazing! The password for the new admin account is… you guessed it, "`admin.`" Apparently, even attackers don't pick strong passwords.

© 2017 Hal Pomeranz and Deer Run Associates

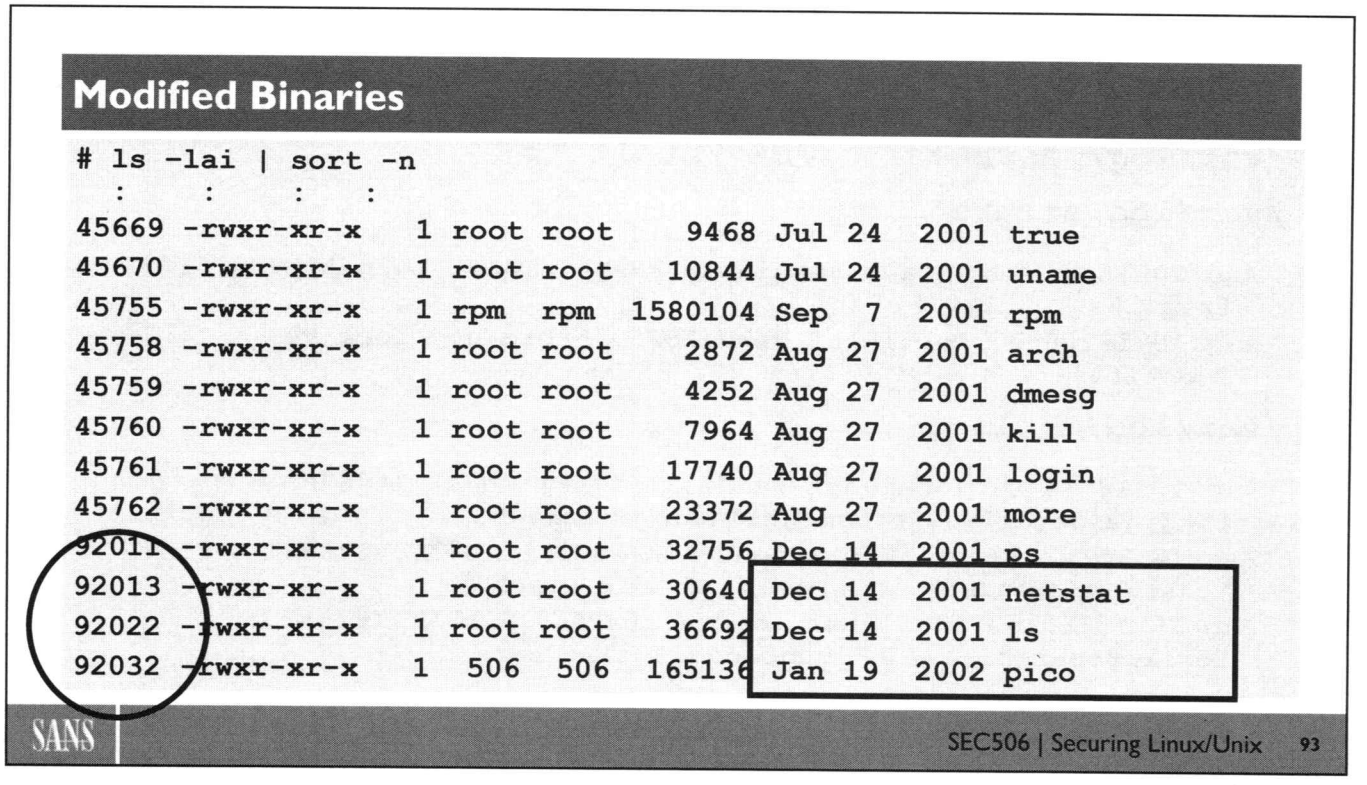
Hidden Files and Directories

```
# find /mnt/hacked -name .\* -type d -print
/mnt/hacked/lib/.x
/mnt/hacked/root/.ssh
/mnt/hacked/root/.links
# ls /mnt/hacked/lib/.x
cl          hide.log    install.loglog   sk
hide  inst        ip           s
# find /mnt/hacked/dev -type f -print
/mnt/hacked/dev/MAKEDEV
/mnt/hacked/dev/shm/k
/mnt/hacked/dev/ttyop
/mnt/hacked/dev/ttyoa
/mnt/hacked/dev/ttyof
/mnt/hacked/dev/hdx1
/mnt/hacked/dev/hdx2
```


Hidden Files and Directories

Two very quick and easy find commands will show the majority of hidden files and directories. First, we look for hidden dotted directories which uncover `/lib/.x` (the others appear benign). Then a quick check for regular files in the `/dev` directory uncovers 5 entries that appear to be configuration files.

© 2017 Hal Pomeranz and Deer Run Associates



Modified Binaries

A thorough examination of a timeline for this file system would NORMALLY reveal ALL of the binaries and files that have been modified. However, in this case, the attacker back-dated the binaries. However, it is often useful to use the `ls` command and include the `-i` flag to show inode numbers. We'll do a numeric sort on the inode numbers in the `ls` output.

Typically, when the installation creates files in the file system, nearly sequential inode numbers are used. At a minimum, they cluster into similar values. If we look for significant outliers that are grouped together, then we might easily be able to identify binaries that were added to the system.

As an example, this screenshot shows that all of the binaries that were created at installation time have inode values of around 45000. However, there is a cluster of binaries including `ps`, `netstat`, `ls`, and `pico` whose inodes cluster around 92000. These appear suspicious, and after additional inspection, it is revealed that `ps`, `netstat`, and `ls` are trojans. The `pico` binary seems to have been added around the same time but not back-dated. Notice that the ownerships of this executable also look a little strange. I'd guess the attacker added this binary separately from the rootkit installation script.

This is a simple example of "outlier analysis", an active field of research in forensics. For more information, see Dave Hull's informative blog posts at:

http://computer-forensics.sans.org/blog/2011/11/07/outlier-analysis-in-digital-forensics

© 2017 Hal Pomeranz and Deer Run Associates

Log Files

/var/log/messages -> /dev/null

/var/log/secure

```
Aug 10 16:04:14 localhost xinetd[732]: START: telnet pid=15169
  from=193.109.122.5
Aug 10 18:58:33 localhost sshd[15287]: Did not receive
  identification string from 202.85.165.46.
```

/var/log/maillog

```
Aug 10 14:14:01 localhost sendmail[4768]: h7ALE1t04763:
  to=jijeljijel@yahoo.com, ctladdr=apache ...
Aug 10 15:42:31 localhost sendmail[23331]: h7AMUVC23321:
  to=newptraceuser@yahoo.com, ctladdr=apache ...
Aug 10 15:43:43 localhost sendmail[25659]: h7AMWXH25629:
  to=skiZophrenia_siCk@yahoo.com, ctladdr=root ...
```

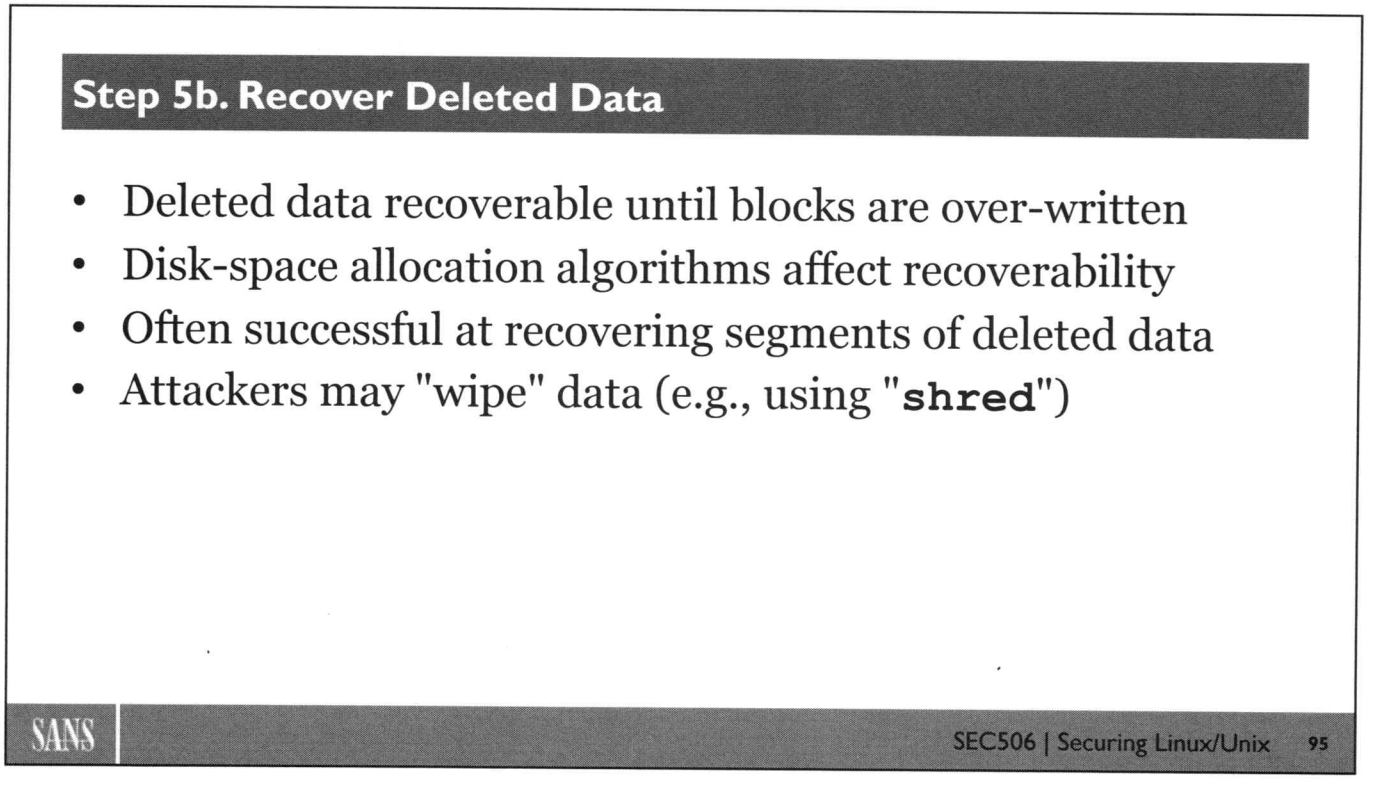
SANS | SEC506 | Securing Linux/Unix 94

Log Files

Inspecting the log files yields some interesting clues. First, notice that the `/var/log/messages` file is linked to `/dev/null` so that system messages are not logged. Also, the `secure` file provides some IP addresses that are worth researching further. Finally, a look at the `maillog` yields a couple of e-mail addresses that might help in researching the suspect on the Internet. It is always worth looking at the `maillog`, as many rootkits are designed to send e-mail to the attacker once the system has been compromised.

The other important thing is that these logs give us more clues about the TIMES that the attacker was on the system: August 10 between 2pm and 6pm.

© 2017 Hal Pomeranz and Deer Run Associates



Step 5b. Recover Deleted Data

If a file is deleted, the contents (data blocks) are not overwritten immediately. The data still exists on disk and can be recovered until the free space is re-allocated by the OS and overwritten.

You might think that the lifespan of deleted data is short, and that storage space that has been freed is used again quickly. This isn't necessarily the case. In fact, because of the inode and disk block allocation algorithms, and the size of modern disk drives, this sort of collision occurs infrequently.

The bottom line is that we can usually recover most (if not all) of a file even though it has been deleted. Studies have been done that indicate that about 80% of the time you will be able to recover files unless they have been deleted using a tool like `shred`. `shred` defeats recovery techniques by wiping (or overwriting) the contents of the file before the file is deleted.

© 2017 Hal Pomeranz and Deer Run Associates

Linux Deleted Files

OS and file system determine how files are deleted:

- EXT2 deallocates inode WITHOUT clearing its contents
  - This makes complete file recovery easy! (`icat`)
- EXT3 clears inode block pointers before deallocation
  - Use specialized tools leveraging indirect blocks (`frib`)
- EXT4 clears extents before deallocation
  - No indirect blocks, so carve (`foremost`)

Linux Deleted Files

Every OS can delete a file on the same file system differently. As we saw previously, there are different structures for different parts of the file system and there are links between the different structures.

The Linux system changed how it deletes files between EXT2 and EXT3. EXT2 keeps inode metadata intact, while EXT3 wipes the inode before putting it back on the free inode list. This makes file recovery MUCH harder.

© 2017 Hal Pomeranz and Deer Run Associates

EXT2 File Undeletion w/ `icat`

- Recover deleted files on older systems
- Writes out a file by its inode number
- Operates on a disk device or disk image

EXT 2 File Undeletion w/ `icat`

Given an inode (particularly of a deleted file) it is often desirable to copy the file by its inode number. The `icat` utility does just this. It accesses the file specified by a given inode and writes it to the stdout file descriptor (which can be redirected to a file). As with the other TSK commands, this operation can be performed on an actual disk device, or on a file that contains a disk image. One common use for `icat` is to recover deleted files (or at least portions of them) in EXT2 file systems.

With flags, you can force `icat` to ignore holes (which are blocks of all zeros). This is useful when dealing with "sparse" files.

The `-s` flag will force `icat` to also display the "slack space" of a file, where attackers will sometimes hide data. For example, consider a file that is 100 bytes in size and it has a 4K block allocated to it. Normally, `icat` will only output the 100-bytes, but `icat -s` will output the full 4K bytes.

© 2017 Hal Pomeranz and Deer Run Associates

icat Example

Get inode numbers of deleted files from "`ils -r`"

```
# ils -r dev_sda1.dd
# icat dev_sda1.dd 45307
```

(appears to be a very informative deleted log file)

icat Example

```
Aug 10 13:33:33 localhost smbd -D[3137]: log: Server listening on port
2003.
Aug 10 13:33:33 localhost smbd -D[3137]: log: Generating 768 bit RSA key.
Aug 10 13:33:34 localhost smbd -D[3137]: log: RSA key generation complete.
Aug 10 14:14:41 localhost smbd -D[5505]: log: Connection from
213.154.118.218 port 2020
Aug 10 14:14:42 localhost smbd -D[3137]: log: Generating new 768 bit RSA
key.
Aug 10 14:14:58 localhost smbd -D[5505]: log: Password authentication
failed for user root from extreme-service-10.is.pcnet.ro.
Aug 10 15:30:30 localhost kernel: eth0: Promiscuous mode enabled.
Aug 10 15:30:30 localhost modprobe: modprobe: Can't locate module ppp0
Aug 10 15:32:16 localhost kernel: eth0: Promiscuous mode enabled.
Aug 10 15:52:12 localhost httpd: fopen: No such file or directory
Aug 10 15:52:12 localhost httpd: httpd: could not open error log file
/etc/httpd/logs/error_log.
Aug 10 15:52:12 localhost httpd: httpd startup failed
Aug 10 15:54:18 localhost httpd: httpd shutdown failed
Aug 10 15:56:11 localhost su(pam_unix)[14689]: session opened for user root
by (uid=0)
Aug 10 16:03:01 localhost su(pam_unix)[14689]: session closed for user root
```


© 2017 Hal Pomeranz and Deer Run Associates

Searching Unallocated Space

All that we are left with on modern operating systems.

How to find "interesting" data?

- Specialized tools: **`foremost`**, **`frib/fib`**
- Use regex string searching

Searching Unallocated Space

Unfortunately, on EXT3 much of the inode's contents are zeroed when a file is deleted. This means that we are left with two rather basic search techniques. One is to use regular expressions to perform string searches. While this is very basic, it can yield very good results if you have the patience to sift through all of the false positives. Searches for IP addresses, e-mail addresses, log file snippets, and similar items can really pay-off!

The other way to look at unallocated space is using specialized tools. `foremost` is a "file carving" tool that looks for file signatures and tries to recover contiguous collections of blocks. `frib` (File Recovery via Indirect Blocks) and `fib` (Find Indirect Blocks) can leverage the indirect block strategy of EXT3 (and earlier) file systems to recover files more precisely.

Understand that these are not mutually exclusive search techniques. You cannot forego one for the other. All methods should be used for complete results.

© 2017 Hal Pomeranz and Deer Run Associates

foremost

- Excellent tool for finding popular (binary) file types (documents, images, ZIP, etc.)
- Extensible and relatively fast
- Be sure to use **-d** option when searching Unix file systems!

foremost

`foremost` was originally developed by the US Air Force Office of Special Investigations (AFOSI) and the Naval Post-Graduate School. It is a simple command-line tool for searching an image for header strings that indicate a certain type of file. The tool is clearly focused on finding common types of binary data—like MS Office documents, images and video files, PDF documents, ZIP files, etc. It tends to do a less good job on textual data like HTML documents and source code. However, you can add your own search patterns to the tool's configuration file if you want to train it to look for other types of data (but most of the common binary file formats are already configured into the tool as noted above). `foremost` is also substantially faster than the `lazarus` tool we'll be discussing next.

By default, `foremost` will create a sub-directory called "`output`" that contains additional sub-directories for each type of file that it finds ("`gif`", "`jpg`", "`pdf`", etc.). The `output` sub-directory will also contain a file called "`audit.txt`" which summarizes the data discovered by the tool.

Typical command-line usage would be something like "`foremost -Tvd` *imagefile.*" The `-T` option causes `foremost` to append a time/date stamp to the "`output`" directory so that you can run `foremost` multiple times without overwriting the output from previous runs. `-v` forces more verbose output so you can monitor the tool as it's operating. You can also use the `-t` option is used to tell `foremost` to search for a particular file type or list of file types (for example, "`-t pdf,ole`" to search for PDF and MS Office documents exclusively). With no `-t` option, `foremost` will search for all known file types (which is what would happen with our sample command-line).

Finally, the `-d` option causes `foremost` to skip the first indirect block in Unix files when you are recovering larger files. It's unfortunate that foremost doesn't try to do more to detect indirect blocks and use the block addresses in them to recover larger chunks of data.

© 2017 Hal Pomeranz and Deer Run Associates

Pertinent Questions

- Why are we just ignoring indirect block data?
- Couldn't we use it do recover file content?
- Can we rebuild the entire original file?

Pertinent Questions

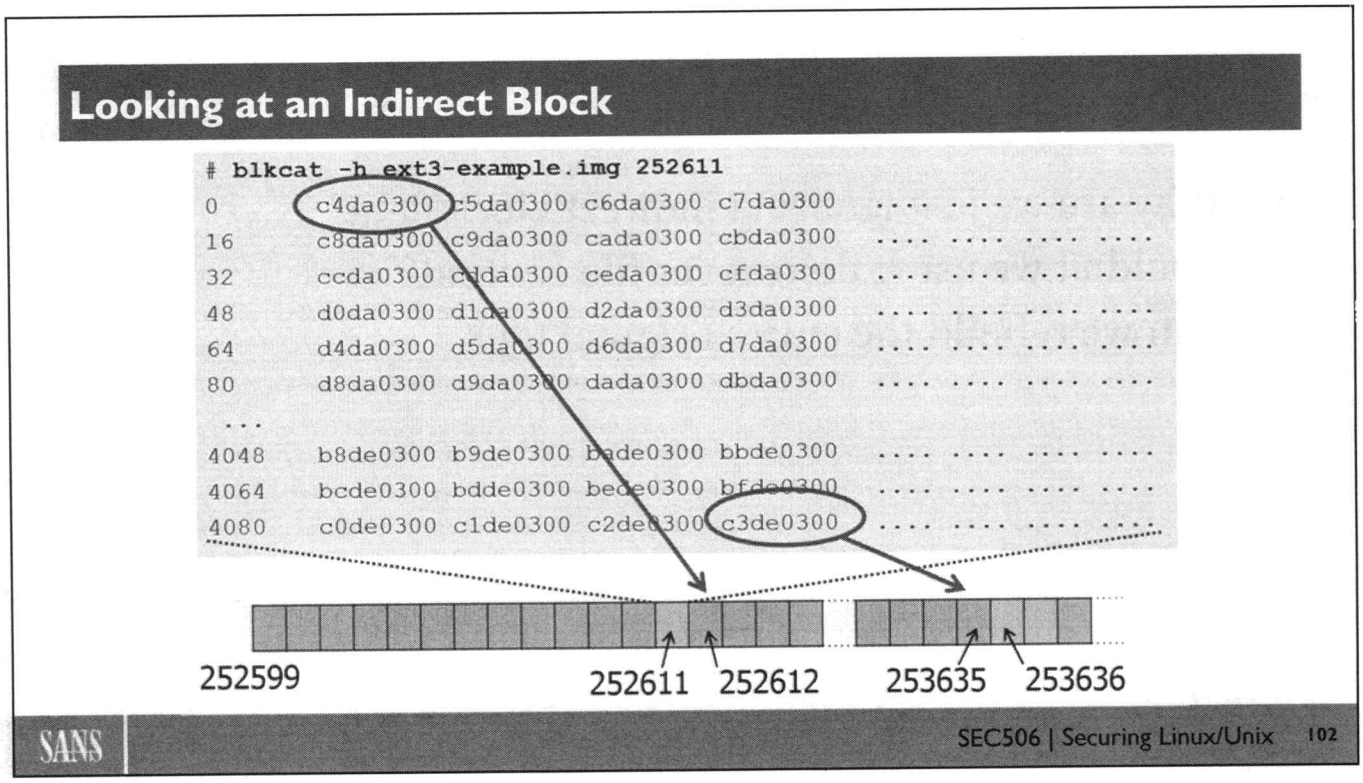
`foremost` is a useful tool, but it can become confused by file fragmentation (when the blocks that make up the file are not contiguous) and doesn't deal gracefully with large files. Rather than just skipping indirect blocks as `foremost` does, Hal Pomeranz created tools which actually use indirect blocks in the file system to recover files more exactly and completely.

We'll take a look at these tools in the upcoming slides, and more details (and the tools themselves) can be found at:

http://blog.mandiant.com/archives/1593

https://github.com/halpomeranz/dfis

© 2017 Hal Pomeranz and Deer Run Associates



Looking at an Indirect Block

If you dump an indirect block with `blkcat`, you will see it's filled with the addresses of the data blocks. Each address is four bytes in *little endian* format. The first four bytes in our example on the slide should be read as 0x0003dac4, which is 252612 in decimal and the address of the first block *after* the indirect block.

Similarly, the last block address in our indirect block is 0x0003dec3 or 253535. And it's likely that 253636 is another indirect block—in this case, the first of the double indirect pointer chain. That means there's a second indirect block at 253637.

By decoding the block addresses in these indirect blocks, we can recover the data in the original file.

© 2017 Hal Pomeranz and Deer Run Associates

Simple File Recovery Strategy

- Find beginning of file via signature
- Does the 13th block look like an indirect block?
- If so, dump associated data blocks
- If last block address is not null, keep going

We can do this manually...

Simple File Recovery Strategy

So, the easiest thing to do is to find files that we want to recover using some file signature to match the beginning of the file. `formost` uses file signatures to find MS Office documents, PDFs, JPEGs, and GIFs, etc. and we can too. In fact, you can simply steal file signatures out of the `foremost.conf` configuration file.

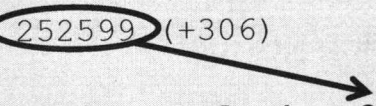
Once we've found the start of file signature, we just "follow our nose." The thirteenth block in the run should be the first indirect block, and we can decode up to 1024 block addresses out of it. If that block is completely full of block addresses (no nulls at the end of the block), then we can assume that the next two blocks start the double indirect block pointer chain. That means more block addresses to decode. We stop when we hit null block addresses in one of our indirect blocks.

Now you could do this manually, but why not write a program to do it instead…

© 2017 Hal Pomeranz and Deer Run Associates

There's an App for That ...

```
# sigfind -b 4096 1F8B0800 ext3-example.img
Block size: 4096  Offset: 0  Signature: 1F8B0800
Block: 251904 (-)
Block: 252096 (+192)
Block: 252293 (+197)
Block: 252599 (+306)
...
# frib ext3-example.img 252599 >recovered.gz
# tar ztf recovered.gz
  ...
perl-5.10.1/patchlevel.h
perl-5.10.1/Configure
```



There's an App for That ...

Here we're using `sigfind` from TSK to search for the standard hexadecimal signature for the beginning of a `gzip`-ed file. Typically, Unix operating systems will contain many such files. We take one of the block numbers reported by `sigfind` and use the `frib` tool to recover the original file. `frib` takes care of locating and decoding all of the indirect blocks and reassembling the file for you.

The advantage to this approach is it allows us to completely recover the original file, assuming the blocks haven't been re-used by newer files. And it deals gracefully with file fragmentation, which cannot be said of `foremost`.

© 2017 Hal Pomeranz and Deer Run Associates

Don't Have a File Signature?

- Indirect blocks have a signature:
  - *Any block N whose first 4-bytes == N+1*
- Use relative location of indirect blocks to put file together
- Beginning of file data will (hopefully) be the 12 blocks before the first indirect block

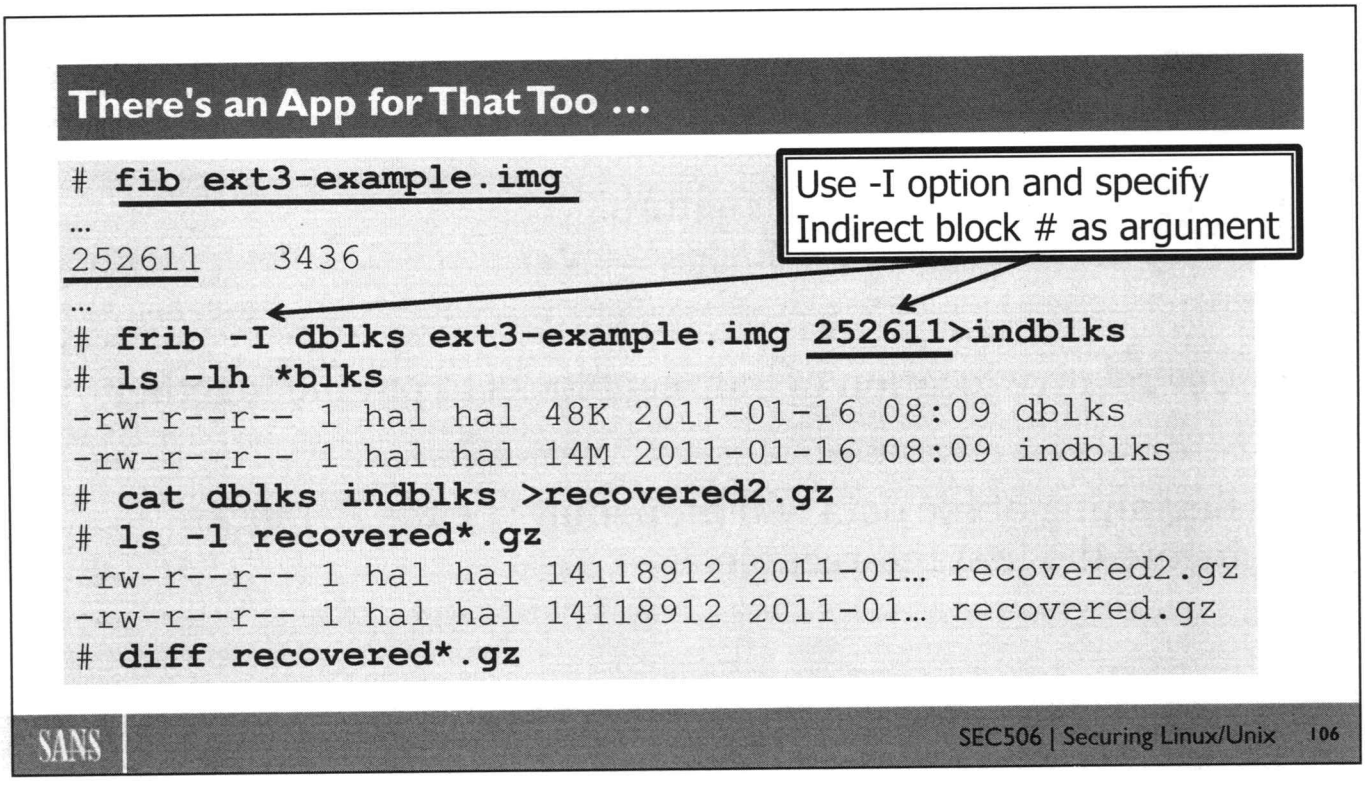
Don't Have a File Signature?

But what if you're looking for a file that doesn't necessarily have a standard beginning of file signature—like a text log file or source code? It turns out that indirect blocks themselves have a signature: simply look for any block whose first four bytes decode to be the address of the next block. This signature works very well and results in few if any false-positives.

Having located an indirect block, you can use the relative placement of other indirect blocks to reassemble larger files. For example, if the last data block referenced in the indirect block is immediately followed by two more blocks that appear to be indirect blocks, then congratulations, you've just discovered the start of the double indirect block chain.

Similarly, once you've located the start of the double indirect chain and the initial indirect block before it, you can probably assume that the 12 blocks prior to the initial indirect block are the direct blocks that make up the beginning of the file. Unless the file is badly fragmented, this is normally the case.

© 2017 Hal Pomeranz and Deer Run Associates



There's an App for That Too ...

The `fib` tool looks for indirect blocks using the signature we discussed on the previous slide. It will even put together the double and triple indirect block pointer chains into a complete list of blocks. `fib` outputs the first indirect block and the number of blocks it finds following that block. In the example on the slide, `fib` must have been able to find the double indirect chain because the number of blocks found (3436) exceeds the maximum of 1024 blocks that can be stored in a single indirect block.

Once we know where the first indirect block sits, we use "`frib -I`" to recover the original file. The argument to the "`-I`" option is the name of the file where the 12 blocks prior to the indirect block should be placed. The blocks from the indirect block onward are sent to the standard output, which here we're redirecting to a file. Concatenating the two files together should get you the original file.

Actually, you don't have to write the 12 direct blocks out as a separate file. Use "`-I -`" to send the first 12 blocks to the standard output followed by the rest of the file recovered from the indirect block chain. So, our command would look like:

```
frib -I - ext3-example.img 252611 >recovered3.gz
```

The resulting file should be the same as our other recovered files.

© 2017 Hal Pomeranz and Deer Run Associates

Step 5c. Search for Strings

String searches useful for:

- Detecting "signatures" of known malware
- Finding "vocabulary" related to crime
- Finding specific intellectual property
- Locating deleted logs, mail messages, etc.

May be all that you're left with on modern Unix file systems

Step 5c. Search for Strings

Sometimes string searching can be the best mechanism to zero in on certain data of interest to your investigation. For example, if you suspect a particular piece of malware, there may be embedded strings that you can look for to detect the presence of the malware on your system. Similarly, particular types of crime will often tend to use a specific "vocabulary"—e.g. bank accounts, victim and "money mule names", etc. for financial fraud—that you can search for. If you suspect theft of trade secrets, you could use string searches to look for stolen intellectual property in your seized images. And while tools like foremost may not be so good at pulling out "textual" kinds of data like log files and e-mail messages, string searches are perfect for this.

© 2017 Hal Pomeranz and Deer Run Associates

Word List Searches (grep -f)

Have a "Dirty Word List" file with known signature strings:

```
# grep -abif patternfile dev_sda.dd
```

Pattern file format is one string per line:

```
rootkit
hack
Irc
gr33tz
etc ...
```


Word List Searches (`grep -f`)

A way to quickly analyze a drive on your forensic station is to perform a quick string search through the entire physical drive. This is easily accomplished by running `grep` on the image files you captured. Once you know that a certain string is on the hard drive, then you could pinpoint where on the drive that information is.

The strings in your pattern file will vary depending on the type of case you're dealing with. For example, you might include different words if you think the attack on your system originated from China, South America, or Eastern Europe. The words you use in a child pornography investigation are different from what you'd be looking for if you suspected an external attack and rootkit installation. Professional forensic investigators will have several different "dirty word lists" for different types of cases that they continuously update and refine.

© 2017 Hal Pomeranz and Deer Run Associates

Creating Strings Databases

- May need to do repeated searches
- More efficient to extract all strings first:

```
# mmls -t dos dev_sda.dd
DOS Partition Table
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000031   0000000031   Unallocated
02:  00:00   0000000032   0001884159   0001884128   Linux (0x83)
03:  00:01   0001884160   0002097151   0000212992   Linux Swap (0x82)

# dd if=dev_sda.dd bs=512 skip=32 count=1884128 |
            strings -a -t d >dev_sda1.asc
```


Creating Strings Databases

The reality, however, is that you will often end up running many different string searches as your case evolves. Having to constantly re-run you search over the entire disk image is wastefully time-consuming. Also, we'd rather have our byte offsets be from the start of each partition, rather than from the start of the entire disk image. That way, we'll have to do less math to figure out the position of each block within the file system.

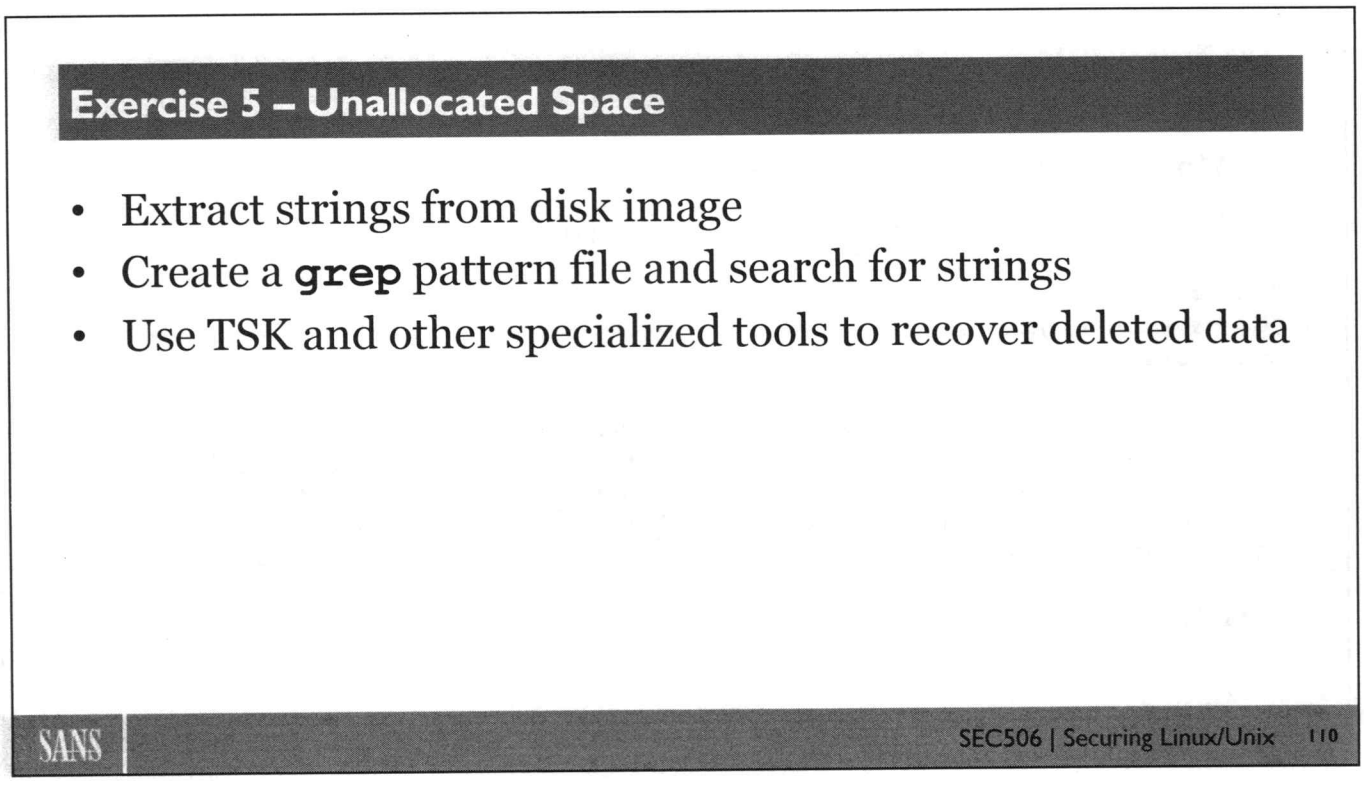
It is usually far more useful to extract all strings from each partition in your image first so that you have a smaller collection of data to search through and so you get more useful byte offsets. Here we're using `dd` combined with the output of `mmls` to extract our Linux partition to the standard output. "`bs=512`" specifies that we're using units of 512-byte sectors to extract the data. Since our partition starts 32 sectors into the image, we "`skip`" the first 32 sectors before we start extracting data. The "`Length`" field in the `mmls` output tells us how many sectors long the partition is, and we feed that into the "`count`" option to `dd`.

The `strings` command will extract all strings of 4 or more ASCII characters terminated by a null (ASCII 0) or newline. "`-a`" ("all") means to search the entire input file, even though it will appear to be binary data, and "`-t d`" means to output the decimal byte offset of the start of each string (similar to the "`-b`" option with `grep`). Once you've extracted the strings, you can just `grep` against that—no need for "`grep -b`" now since `strings` already output the byte offsets for you! Our `grep` command would be something like "`grep -if` *patternfile* `dev_sda1.asc`"

If you're dealing with a Windows image or a Unix image that has lots of Windows files in it (perhaps a file server, etc.) you will also want to use "`strings -a -t d -e l`" to extract the Unicode strings from your image. You could repeat the above `dd` command to do this, re-reading the same data twice is wasteful—and time-consuming if you're dealing with a large image! The following article has instructions on how to use FIFOs to extract ASCII and Unicode strings at the same time:

http://computer-forensics.sans.org/blog/2010/01/27/fun-with-fifos-part-ii-output-splitting

© 2017 Hal Pomeranz and Deer Run Associates



Exercise 5 – Unallocated Space

Exercises are in HTML files in two places: on the course USB media, and in the home directory of the SANS user in the lab virtual machine. These two locations contain identical copies of the exercise files—it just depends on whether you prefer to look at them from a web browser inside the lab VM (because you're working in full-screen mode) or from your host operating system.

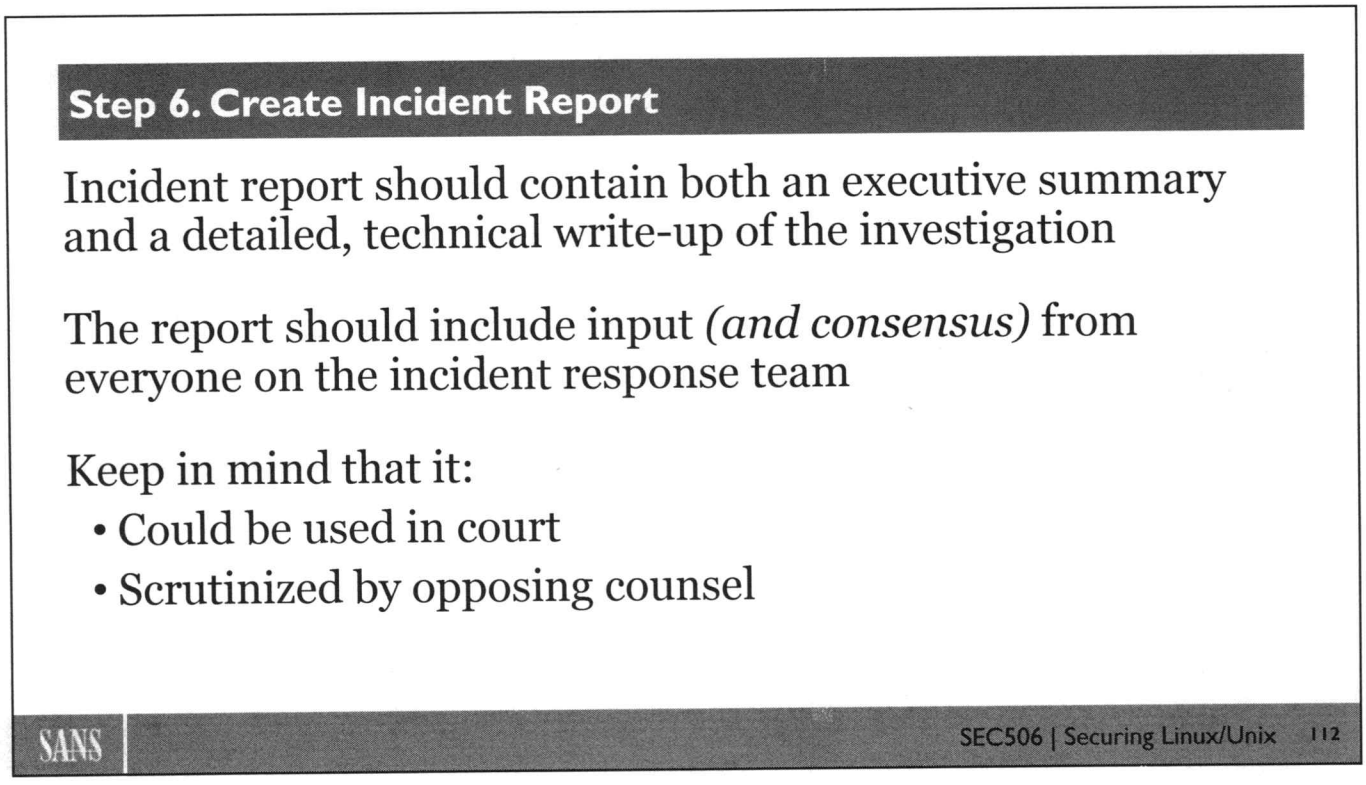
1. Open your web browser
2. Select "File … Open …" from the browser menu (Ctrl-O is the usual keyboard shortcut)
3. This is Day 6, Exercise 5, so navigate to *.../Exercises/Day_6/index.html* and open this HTML file
4. Click on the hyperlink which is the title of Exercise 5
5. Follow the instructions in the exercise

© 2017 Hal Pomeranz and Deer Run Associates

Which Brings Us Full-Circle to the First Exercise in the Course!

This page intentionally left blank.

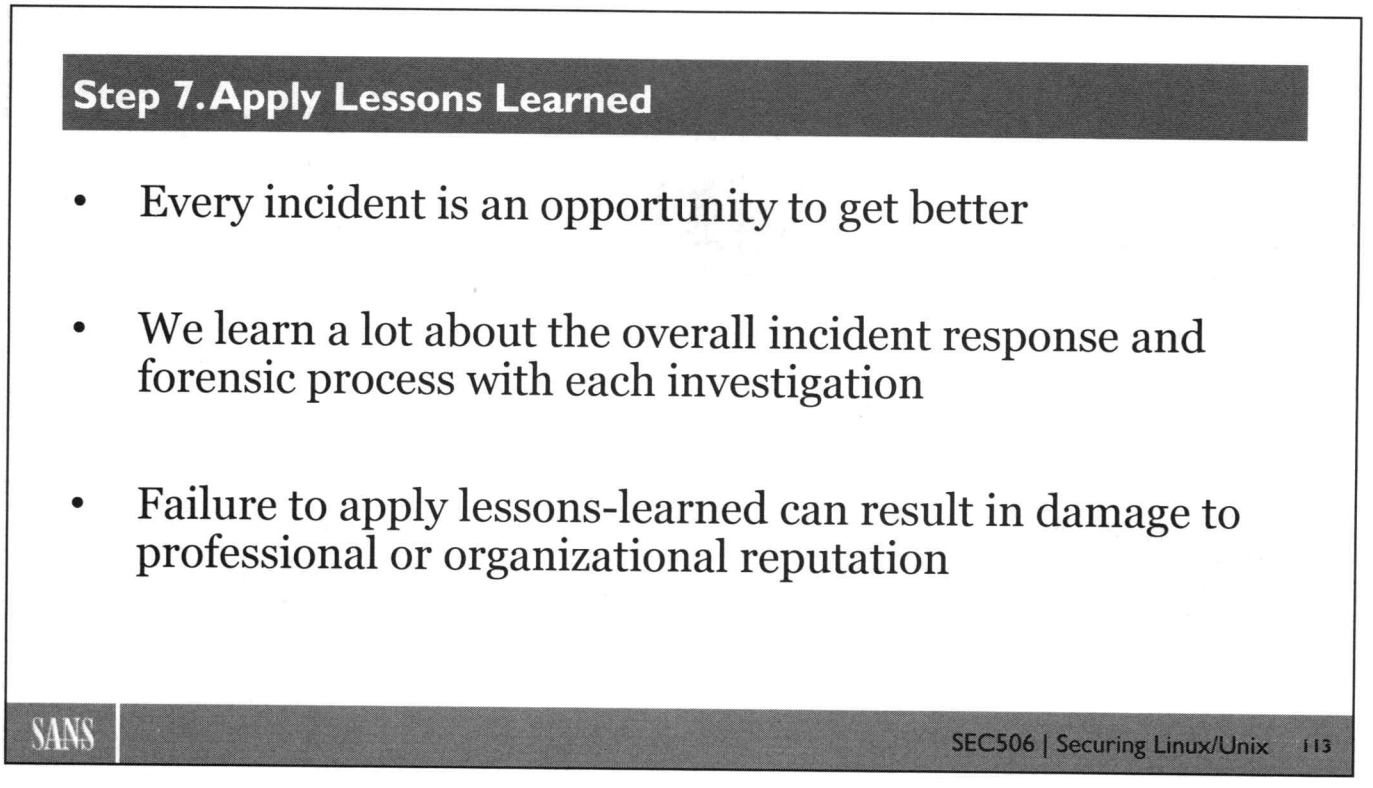
© 2017 Hal Pomeranz and Deer Run Associates



Step 6. Create Incident Report

Reporting is not entirely a technical aspect of computer forensics, but it is probably the most important step of the forensic process. Most reporting is done to individuals who may not be educated in computer hardware, networking topics, or computer crime law. You need to ensure that your reporting clearly explains the evidence you found, the techniques you used and defines everything that is technical. Many investigators, unfortunately, overlook this easy step because they think that most people can understand how the Internet works or even topics as simple as file downloading. However, in a court of law, or even in your own company, for anyone to utilize your results, they need to make sure that it is understandable. In a nutshell, document everything, explain and define basic and advanced topics, and show the results of your investigation.

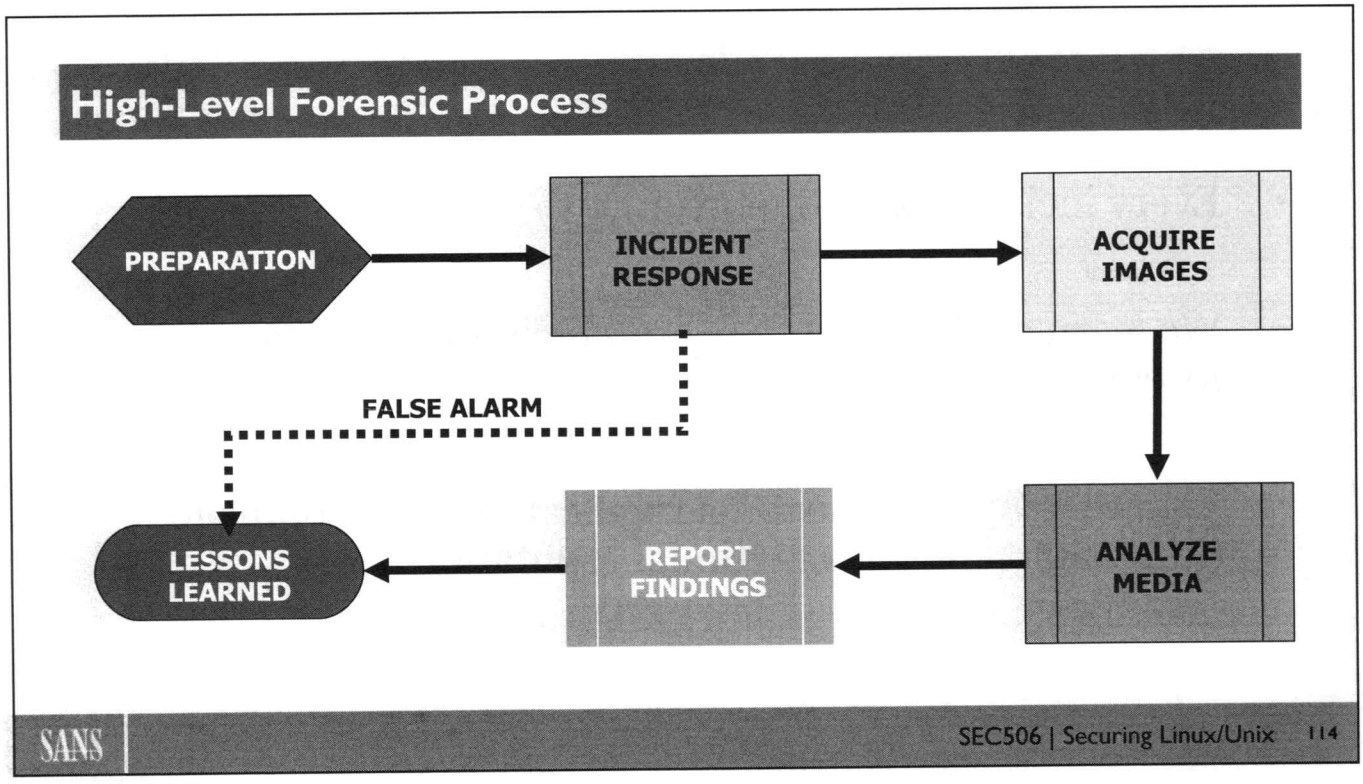
© 2017 Hal Pomeranz and Deer Run Associates



Step 7. Apply Lessons Learned

Every incident is an opportunity to learn about security and its application within our organization. It may be that the lesson learned is that new policies are needed. It may be that a policy was ignored. It could also be the case that there was little the organization could do, in which case the lessons learned would revolve around better incident handling. Make absolutely sure that your organization learns from these lapses. Failure to apply these lessons could result in even worse damage to corporate reputation. After all, you can survive one web defacement, but how about if it happens again one week later? Maybe not.

© 2017 Hal Pomeranz and Deer Run Associates



High-Level Forensic Process

Again, here you see a simplified flowchart of the forensic investigation process. As we stated before, being adequately prepared to respond is absolutely critical, and if you spend enough time in this proactive phase, you can greatly reduce the stress that you encounter and the chance of making costly mistakes during the reactive process of incident response.

It is important to mention that not every incident is an intrusion or compromise, so we have incorporated a dotted-line to indicate that false alarms do occur. In these cases, the incident response team is activated to handle what appears to be an incident, but it later turns out to be something less serious—such as an improperly configured system or over-zealous user/administrator.

Image acquisition is the process of creating true bit images of the disks, tapes, and removable media that may be associated with the system in question. It is an art-form unto itself, albeit an easy one to master. Once these images have been acquired, the original evidence (system disk) can be locked in a container for safe keeping. You will conduct your investigation against these images, performing media analysis. Examining the images for small pieces of evidence that will ultimately help you to reconstruct what happened in the past on a system.

The two final pieces are the ones that are most often overlooked. The creation of an accurate and effective incident report, and incorporating any lessons-learned (especially after "false alarms") back into the security processes and policies of the organization.

© 2017 Hal Pomeranz and Deer Run Associates

Conclusions

- We have covered a tremendous amount of information about Linux Forensics
- There is MUCH more, but armed with this basic knowledge, you will succeed
- Forensics is 30% preparation, 70% perspiration
- Requires diligence, a keen eye, and persistence

This page intentionally left blank.

© 2017 Hal Pomeranz and Deer Run Associates

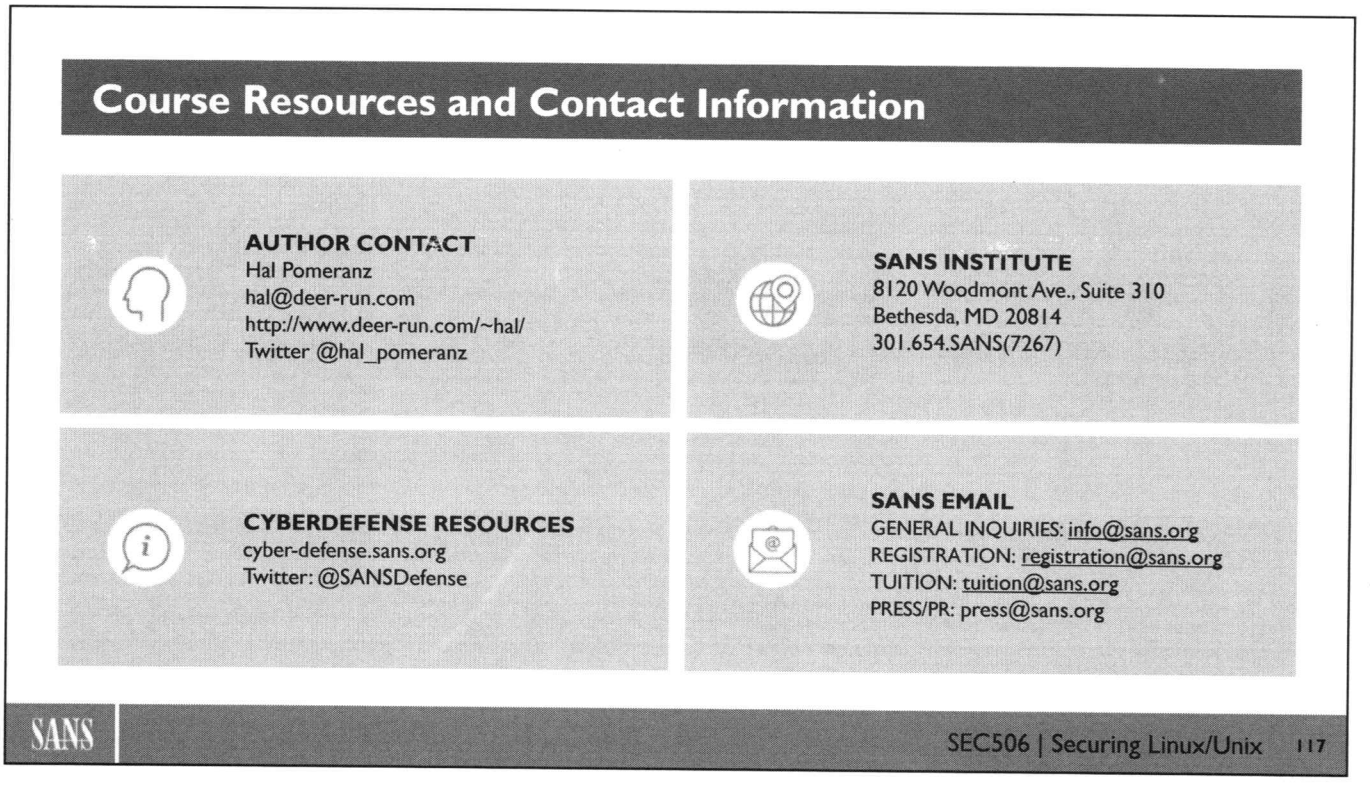
THANK YOU

Please remember to fill out your evals!

SANS SEC506 | Securing Linux/Unix 116

Hal Pomeranz
hal@deer-run.com
http://www.deer-run.com/~hal/
@hal_pomeranz on Twitter

© 2017 Hal Pomeranz and Deer Run Associates



Hal Pomeranz
hal@deer-run.com
http://www.deer-run.com/~hal/
@hal_pomeranz on Twitter

© 2017 Hal Pomeranz and Deer Run Associates

This page intentionally left blank.

Index

© 2017 Hal Pomeranz and Deer Run Associates

C

© 2017 Hal Pomeranz and Deer Run Associates

D

E

© 2017 Hal Pomeranz and Deer Run Associates

F

G

H

© 2017 Hal Pomeranz and Deer Run Associates

I

J

K

© 2017 Hal Pomeranz and Deer Run Associates

L

M

N

© 2017 Hal Pomeranz and Deer Run Associates

O

P

© 2017 Hal Pomeranz and Deer Run Associates

Q

R

© 2017 Hal Pomeranz and Deer Run Associates

S

© 2017 Hal Pomeranz and Deer Run Associates

© 2017 Hal Pomeranz and Deer Run Associates

T

U

© 2017 Hal Pomeranz and Deer Run Associates

V

W

X

Y

Z

© 2017 Hal Pomeranz and Deer Run Associates