

First Responders Guide to Computer Forensics: Advanced Topics

Richard Nolan
Marie Baker
Jake Branson
Josh Hammerstein
Kris Rush
Cal Waits
Elizabeth Schweinsberg

September 2005

HANDBOOK
CMU/SEI-2005-HB-003



CarnegieMellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

First Responders Guide to Computer Forensics: Advanced Topics

CMU/SEI-2005-HB-003

Richard Nolan
Marie Baker
Jake Branson
Josh Hammerstein
Kris Rush
Cal Waits
Elizabeth Schweinsberg

September 2005

CERT Training and Education

Unlimited distribution subject to the copyright.

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Christos Scondras
Chief of Programs, XPK

This work is sponsored by the SEI FFRDC primary sponsor and the Department of Homeland Security. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2005 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (<http://www.sei.cmu.edu/publications/pubweb.html>).

Table of Contents

Executive Summary	xi
Abstract.....	xiii
1 Module 1: Log File Analysis.....	1
1.1 Swatch	3
1.1.1 Swatch Log Monitor.....	4
1.1.2 Swatch Installation	5
1.1.2.1 Installing Perl Modules.....	5
1.1.2.2 Installing Swatch	6
1.1.3 Swatch Configuration.....	8
1.1.3.1 The Configuration File Location.....	9
1.1.3.2 Adding Rules to the Configuration File	9
1.1.4 Swatch Execution	15
1.1.5 Summary.....	17
1.2 Microsoft Log Parser 2.2.....	18
1.2.1 Microsoft Log Parser Features	20
1.2.2 Log Parser Installation	21
1.2.3 Log Parser Input and Output	22
1.2.3.1 Input Formats.....	22
1.2.3.2 Output Formats.....	24
1.2.4 Log Parser Queries.....	26
1.2.4.1 Query Examples	27
1.2.5 Log Parser COM Objects.....	29
1.2.5.1 Creating Custom Input Formats.....	29
1.2.5.2 Using the Log Parser COM API	30
1.2.6 Log Parser Execution.....	31
1.2.7 Summary.....	33
2 Module 2: Process Characterization	35
2.1 Understanding a Running Process	36
2.1.1 Objectives.....	37
2.1.2 Programs, Processes, and Threads.....	38
2.1.3 Threads	39
2.1.3.1 Displaying Threads for a Running Process.....	39

2.1.3.2	Sysinternals <i>Process Explorer</i>	40
2.1.4	Process Tree Structure	43
2.1.4.1	<i>pstree</i> (Linux).....	44
2.1.4.2	Linux <i>ps -A</i>	44
2.1.5	Process Descriptions	46
2.1.6	Process Hashes (National Software Reference Library).....	47
2.1.7	Process Analysis Checklist	49
2.1.8	Common Process Characteristics	51
2.1.8.1	Process Filenames	51
2.1.8.2	Open Ports	53
2.1.8.3	Open Files	55
2.1.8.4	Base Priority	56
2.1.8.5	Process Times and Terminated Processes.....	58
2.1.8.6	Location of Process Image	60
2.1.8.7	Survivable Processes	61
2.1.8.8	Process Forensic Tasks.....	66
2.2	Automated Process Collection	76
2.2.1	Objectives.....	77
2.2.2	<i>First Responder Utility (FRU)</i>	78
2.2.2.1	<i>First Responder Utility (FRUC)</i> Setup	79
2.2.3	<i>Forensic Server Project (FSP)</i>	82
2.2.3.1	<i>FSP</i> Setup	82
2.2.3.2	Testing <i>FRUC</i>	83
2.2.3.3	Output of <i>FRUC</i>	84
3	Module 3: Image Management.....	87
3.1	Slice and Dice with <i>dd</i>	88
4	Module 4: Capturing a Running Process	101
4.1.1	Hedons and Dolors.....	103
4.1.2	Capturing a Process on a Windows System.....	104
5	Module 5: Understanding Spoofed Email.....	113
5.1	Objectives	114
5.2	Identifying Spoofed Email	115
5.2.1	Definition of the Problem.....	116
5.2.2	Understanding the Process of Sending and Receiving Email	117
5.2.2.1	The Life Cycle of an Email.....	117
5.2.2.2	Overview of the Simple Mail Transfer Protocol.....	119
5.2.3	Understanding Email Headers.....	123
5.2.3.1	Interpreting Email Headers.....	123
5.2.4	How Spoofed Email Is Sent	127

5.2.4.1	Open Mail Relay	127
5.2.4.2	Compromised Machines	129
5.2.4.3	Self-Owned Mail Servers	129
5.2.4.4	Temporary Accounts	129
5.2.4.5	Hijacked Accounts	129
5.2.5	How to Identify Spoofed Email	130
5.2.5.1	Carefully Examine the “Received” Headers.....	130
5.2.5.2	Look Out for Spoofed Headers	132
5.2.5.3	Comparing Timestamps	133
5.3	Tracing the Origins of a Spoofed Email.....	135
5.3.1	<i>nslookup</i>	136
5.3.2	<i>whois</i>.....	139
5.3.2.1	IP Block Identification.....	139
5.3.2.2	WHOIS Information for a Domain Name	142
5.3.3	<i>Traceroute</i>	144
5.3.4	<i>Sam Spade</i>.....	145
5.4	Summary	146
References.....		147

List of Figures

Figure 1:	Example Run of the Swatch Configuration File	14
Figure 2:	Example Run of PsList	39
Figure 3:	Sysinternals Process Explorer Utility.....	40
Figure 4:	Verifying a Process Image in Process Explorer	41
Figure 5:	The Strings Tab in Process Explorer	41
Figure 6:	Displaying a Process Tree Using PsList.....	43
Figure 7:	Displaying a Process Tree Using pstree	44
Figure 8:	Displaying PID Assignments Using ps	45
Figure 9:	WinTasks Process Description	47
Figure 10:	Listing Process Filenames Using pulist.....	52
Figure 11:	Displaying Open Ports Using fport	53
Figure 12:	Displaying Open Ports Using netstat.....	54
Figure 13:	Viewing Handles Using handle.....	55
Figure 14:	Displaying Which Process Has Port 6002 Open	56
Figure 15:	Displaying Who Has the Bash Shell Open	56
Figure 16:	Displaying All the Currently Open Files by the User Root	56
Figure 17:	Listing Priority Levels Using pslist	57
Figure 18:	Listing Priority Levels Using top	57
Figure 19:	Displaying the Priority Level for a Specific Process	57

Figure 20: Checking Uptime Using psuptime	58
Figure 21: Checking Elapsed Time for a Process Using pslist.....	58
Figure 22: Windows Event Log	59
Figure 23: psloglist Command.....	59
Figure 24: Locating a Process Image Using ListDLLs	60
Figure 25: Locating a Process Image Using ps.....	60
Figure 26: Locating a Process Image by PID.....	61
Figure 27: autorunsc.exe Command.....	62
Figure 28: The chkconfig -list Command.....	63
Figure 29: A Cron Log	64
Figure 30: The Crontab Command.....	65
Figure 31: The svchost.exe 780 Process.....	67
Figure 32: listdlls.exe Output for svchost.exe.....	68
Figure 33: MD5deep Utility.....	69
Figure 34: Performing a String Search Using grep.....	69
Figure 35: The mshearts.exe 2840 Process	70
Figure 36: listdlls.exe Output for the mshearts Process.....	71
Figure 37: MD5deep.exe Command Line Arguments	71
Figure 38: strings Command.....	73
Figure 39: strings Command Output	73
Figure 40: Hash of John the Ripper	74
Figure 41: First Part of the fruc.ini File	80
Figure 42: Second Part of the fruc.ini File	80

Figure 43: Final Part of fruc.ini File.....	81
Figure 44: FSP Setup	83
Figure 45: FRUC Utility Command	83
Figure 46: FSP Command Output	84
Figure 47: FRUC Output File.....	85
Figure 48: FRUC Audit File.....	85
Figure 49: Result of Using md5 to Calculate a Hash Value.....	92
Figure 50: Confirming the Result of Splitting Images	92
Figure 51: Result of Using cat and md5sum to Check the Integrity of Split Images.....	93
Figure 52: Result of Using md5sum to Check the Integrity of a New Image	94
Figure 53: Finding a .jpg Tag in a Captured Image	96
Figure 54: Decimal Form of the Beginning of the .jpg File	96
Figure 55: Searching for the End of the .jpg File	97
Figure 56: Tag Delineating the End of a .jpg File.....	97
Figure 57: Decimal Address for the End of the .jpg File	97
Figure 58: Calculating the Size of the .jpg File	97
Figure 59: File Carved Out Using dd	98
Figure 60: Viewing Carved .jpg File.....	98
Figure 61: Running a Trusted Command	106
Figure 62: Command Shell Spawned from a Trusted CD.....	106
Figure 63: netcat Command to Listen on Port 3333.....	106
Figure 64: Using Trusted pslist and netcat to Specify IP Address and Listening Port.....	107

Figure 65: Looking for Suspicious Processes Using cat	107
Figure 66: Suspicious Process Found.....	107
Figure 67: netcat Command to Listen on Port 4444.....	108
Figure 68: Specifying netcat Listener Machine and Port.....	108
Figure 69: Viewing Path to a Suspicious Process.....	108
Figure 70: Setting Up a Listening Session on a Suspicious Process.....	109
Figure 71: Collecting the Executable of a Suspicious Process	109
Figure 72: Calculating a Hash of a Captured Process	109
Figure 73: The Life Cycle of an Email	118
Figure 74: Mail Delivery for Valid Users	128
Figure 75: Spoofed Email via an Open Relay	128
Figure 76: nslookup of Valid Fully Qualified Domain Name	137
Figure 77: nslookup of Falsified Host Information	138
Figure 78: WHOIS Query of ARIN.....	140
Figure 79: WHOIS Query of APNIC	141
Figure 80: WHOIS Query of IANA.....	142
Figure 81: Query of .com WHOIS Database	143
Figure 82: Query of the Registrar's WHOIS Database.....	143
Figure 83: Traceroute Example.....	145

List of Tables

Table 1:	Actions in Swatch.....	11
Table 2:	time_regex for Popular Services	13
Table 3:	Common Input Formats.....	22
Table 4:	Output Formats	24
Table 5:	Misc Log Parser Commands	31
Table 6:	A Subset of ps Options.....	52
Table 7:	Output Headings for ps and top	52
Table 8:	dd Syntax	88
Table 9:	Tools for Capturing Running Processes	104
Table 10:	The Life Cycle of an Email	118
Table 11:	Email Headers.....	124

Executive Summary

First Responders Guide to Computer Forensics: Advanced Topics expands on the technical material presented in SEI handbook CMU/SEI-2005-HB-001, *First Responders Guide to Computer Forensics* [Nolan 05]. While the latter presented techniques for forensically sound collection of data and reviewed the fundamentals of admissibility pertaining to electronic files, this handbook focuses exclusively on more advanced technical operations like process characterization and spoofed email. It is designed for experienced security and network professionals who already have a fundamental understanding of forensic methodology. Therefore, emphasis is placed on technical procedures and not forensic methodology.

The first module focuses on log file analysis as well as exploring techniques for using common analysis tools such as *Swatch* and *Log Parser*. The second module focuses on advanced techniques for process characterization, analysis, and volatile data recovery. The third module demonstrates advanced usage of the *dd* command-line utility. Topics include how to slice an image and reassemble it with *dd*, carving out a section of data with *dd*, and imaging a running process with *dd*. The fourth and final module examines spoofed email messages. This module looks at the RFCs for email, describes how email messages are spoofed, and presents some techniques for identifying and tracing spoofed email.

Our focus is to provide system and network administrators with advanced methodologies, tools, and procedures for applying sound computer forensics best practices when performing routine log file reviews, network alert verifications, and other routine interactions with systems and networks. The final goal is to create trained system and network professionals who are able to understand the fundamentals of computer forensics so that in the normal course of their duties they can safely preserve technical information related to network alerts and other security issues. This handbook is not intended to be a training guide for computer forensics practitioners, but rather an advanced resource for system and network security professionals who are charged with performing first responder functions. The target audience includes system and network administrators, law enforcement, and any information security practitioners who find themselves in the role of first responders. The handbook should help the target audience to

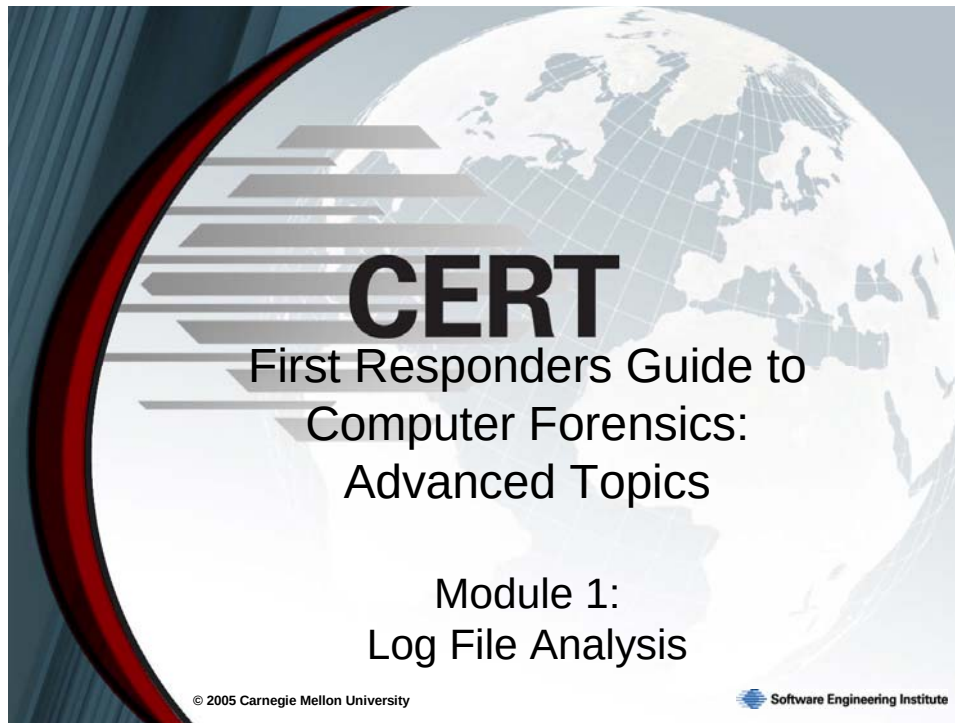
- install, configure, and use *Swatch* to analyze log files
- install, configure, and use *Log Parser* to analyze log files
- understand advanced elements of a running process
- perform an automated collection of volatile data
- carve out data using the *dd* command-line utility
- use the *dd* command-line utility to slice and reassemble images and files

- understand spoofed email
- identify reliable information in an email header

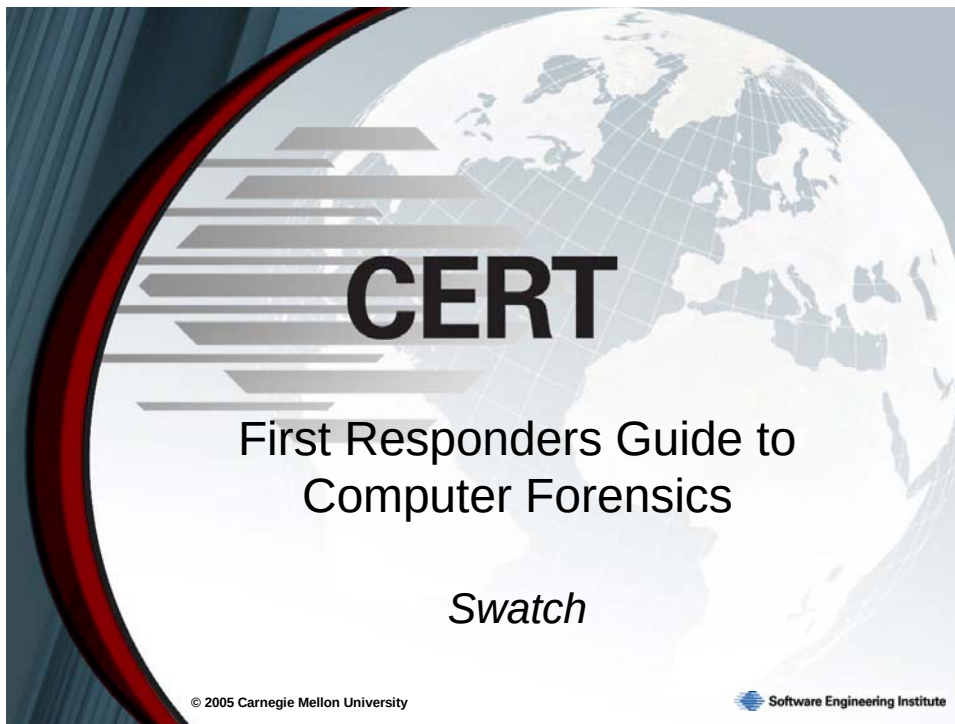
Abstract

This handbook expands on the technical material presented in SEI handbook CMU/SEI-2005-HB-001, *First Responders Guide to Computer Forensics*. While the latter presented techniques for forensically sound collection of data and explained the fundamentals of admissibility pertaining to electronic files, this handbook covers more advanced technical operations such as process characterization and spoofed email. It describes advanced methodologies, tools, and procedures for applying computer forensics when performing routine log file reviews, network alert verifications, and other routine interactions with systems and networks. The material will help system and network professionals to safely preserve technical information related to network alerts and other security issues.

1 Module 1: Log File Analysis



This module focuses on log file analysis, specifically post-event analysis using *Swatch* and *Log Parser*. We explain how to install, configure, and execute *Swatch* and *Log Parser* and provide several sample configurations for each.



Overview

Why Swatch?

Installation

Configuration

Execution

Forensics, in this case, is the act of looking for events after they have happened, not performing an investigation for law enforcement.

2005 Carnegie Mellon University

3



1.1 Swatch

The focus is on using *Swatch* and *Log Parser* as forensic analysis tools, meaning that they will be used on logs after an incident has occurred. If you are planning to use these techniques on files involved in a law enforcement investigation, please make sure you prepare the files according to established best practices before use.

Swatch Log Monitor

Swatch, the Simple Watcher, is an open source log monitoring tool written in Perl for use primarily on UNIX/Linux systems.

Swatch can be used to monitor current logs on running servers, or to examine older logs.

The configuration file contains a list of regular expressions to look for and actions to take, if any are found, called rules.

While originally designed for use with syslog files, Swatch can be used on any file.

2005 Carnegie Mellon University

4



1.1.1 Swatch Log Monitor

Log files are useful only if they are read. After an incident, log files often have clues as to what happened. However, many servers produce large volumes of log information, often spread out over more than one file, so sifting through this data can be tedious and time consuming. As an added problem, different servers have different log formats. If it is necessary to compare files, it can be challenging to match up fields.

Swatch, the Simple Watcher log monitoring tool, is capable of searching a file for a list of strings and then performing specific actions when one is found. It was designed to do real-time monitoring of server log files but can also be set to process a stand-alone file. *Swatch* was designed to work with syslog files, but it can be used on any file.

Swatch was written in Perl, and because of the way it is installed it is best used on a Linux system. It is an open source tool, and the project is maintained on SourceForge.

Throughout this module we will consider more heavily the case where *Swatch* is used to examine older log files as opposed to active log files.

Swatch Installation

Requirements: Perl 5, make utility, tar utility

Download Swatch from the SourceForge project:

<http://sourceforge.net/projects/swatch/>

Download the throttle patch from <http://www.cert.org>

Obtain and install the additional Perl modules:

- Time::HiRes
- Date::Calc
- Date::Format
- File::Tail

Make Swatch – On the command line, type in progression:

```
tar zxvf swatch-3.1.1.tar.gz
patch -p0 < throttle.patch
cd swatch-3.1.1
perl Makefile.PL
make
make test
make install
```

2005 Carnegie Mellon University

5



1.1.2 Swatch Installation

Swatch has the same installation process as a Perl module. You download a tarball, uncompress it, expand it, and build it. The tool installs itself in */usr/bin*, and you can use it from any directory. It also installs a manual page.

To begin, make sure that Perl 5 is installed on the machine. Later versions of Perl may come with some of the necessary modules installed already. You also need the ability to use the GNU utility *make* to fully install *Swatch*.

1.1.2.1 Installing Perl Modules

If your Linux distribution offers versions of the Perl modules needed to support *Swatch*, it is best to get the operating-system-specific ones. Otherwise, you will need to obtain them from either the module's developer's web site or from a centralized repository such as the Comprehensive Perl Archive Network (CPAN).¹ CPAN indexes most of the Perl packages available, makes the list searchable, and has them available for download. They also have links to the developer's web site if you would prefer to get the modules straight from the source.

To install *Swatch* you need these modules:²

- File::Tail – in *File-Tail-0.99.1.tar.gz*
- Date::Calc – in *Date-Calc-5.4.tar.gz*

¹ <http://www.cpan.org>

² All module version numbers are current at time of printing.

- Date::Parse – in *TimeDate-1.16.tar.gz*
- Time::HiRes – in *Time-HiRes-1.66.tar.gz*

To support these you might also need

- Bit::Vector – in *Bit-Vector-6.4.tar.gz*
- Carp::Clan – in *Carp-Clan-5.3.tar.gz*

Once the tar file is on the machine, you must decompress and expand it before it can be installed. Once expanded, read the *INSTALL* file to make sure that the module has the standard installation commands. For these modules, there is a Perl script called *Makefile.PL* that creates a makefile specific to the machine. Next, run the make file three times: once to initialize, once to test, and then once to install. After that, the package is ready to use. In order for other users to be able to use these modules, they must be installed by root. The commands follow in shaded boxes (the normal text is what is sent to the console):

```
tar zxvf perlmod.tar.gz
```

Lists all the files in perlmod.tar

```
cd perlmod
perl Makefile.PL
```

Writing Makefile for Perl::Mod

```
make
```

Check for errors

```
make test
```

Look for "All tests successful"

```
make install
```

Check for errors

Repeat for the other packages and you are ready to install *Swatch* itself.

1.1.2.2 Installing *Swatch*

Installing *Swatch* involves the same procedure as the Perl modules. First, download the tarball to the local machine from <http://sourceforge.net/projects/swatch>. There is a patch needed to enable the throttle action to fully work. Download that from <http://www.cert.org>. These instructions are for *Swatch* 3.1.1 (the normal text is what is sent to the console):

```
tar zxvf swatch-X.X.X.tar.gz
```

Lists all the files in swatch-X.X.X.tar

```
patch -p0 < throttle.patch
```

The character after the -p is a zero

```
cd swatch-X.X.X  
perl Makefile.PL
```

Writing Makefile for swatch

If Time::HiRes, Date::Calc, or Date::Parse are missing it will say

```
make
```

Check for errors

```
make test
```

Look for "All tests successful"

```
make install
```

Check for errors

Swatch is now ready to be executed.

Swatch Configuration–Rules ¹

The configuration file contains a list of rules

Default file location is \$HOME/.swatchrc, but it can be any name and any location

The three parts of a rule:

- Event – “watchfor” or “ignore”
- Pattern – regular expression pattern to look for
- Action – what the script does when the pattern is found

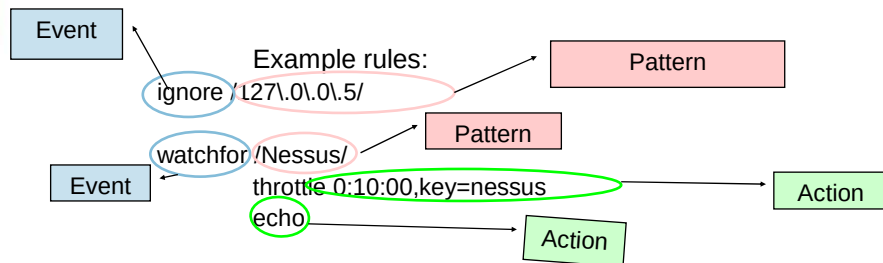
1.1.3 Swatch Configuration

The configuration file is the source of *Swatch*'s power. It is a text file of rules that are used to create the script that will be run against the log file. This topic is about how to develop your own configuration file.

Before you begin adding rules, determine what you are trying to find. Perhaps you want to look for Nessus attacks in your Apache log files, or find when people try to use an SMTP server as an open relay. Make a list of strings or regular expression patterns that you might need. Keep in mind that creating a good configuration file is an iterative process, and it may take a few rounds to extract the desired information from the file.

Swatch Configuration–Rules 2

Rules are looked for in the order they appear in the configuration file.



- The first rule looks for the string “127.0.0.5” and ignores any log entries that contain it. The second rule looks for log entries that contain “Nessus” and echoes them to the console, but only at the rate of one entry every 10 minutes.

2005 Carnegie Mellon University

7



1.1.3.1 The Configuration File Location

By default, *Swatch* looks for the configuration file *.swatchrc* in the home directory. If this file is not found, it uses a default configuration of

```
watchfor /.*/  
echo
```

This merely echoes every message in the log file to the console. This is not any more useful than inspecting the log file by hand. To harness the power, a customized configuration file should be created (this is addressed in the next section, 1.1.3.2).

The default name is *.swatchrc*, but it can be whatever you want. Configuration files for different types of log files may be identified by a distinct name. One might call their Apache configuration file *apache.swatchrc* and their sendmail *sendmail.swatchrc*. The file can be stored anywhere that is accessible from the command line, not just in the home directory. If a different name or location is used for the configuration file, it is added as an argument to the command line when *Swatch* is executed (this is discussed in Section 1.1.4).

1.1.3.2 Adding Rules to the Configuration File

Rules are a list of keywords and values that are used to make conditional statements to check against, and actions to take if one is true. They have three parts: the event, the pattern, and the action(s).

1.1.3.2.1 Types of Events

There are two types of events in *Swatch*: “watchfor” and “ignore.” The keyword “watchfor” looks for the specified pattern in messages in the log file. The “ignore” keyword will skip the rest of the rules when a message matches the pattern.

By default, the first rule that matches a message will be the only rule that acts on that message. This property can be harnessed by using the “ignore” event to filter out messages. For example, since you know that the system administrator always uses the same machine to do penetration testing, you create an “ignore” rule for messages that come from a specific internal IP address, 127.0.0.5, and list the rule for Nessus scans after it so that internal scans will not cause alerts. In this case, you will want to put these two events in this order:

```
ignore /127\.0\.0\.5/
watchfor /Nessus/
        throttle 0:10:00, key=nessus
        echo
```

1.1.3.2.2 Types of Patterns

The value for the event keyword is the regular expression pattern that follows on the same line. The simplest regular expression is a string to match character by character enclosed in “/”; for example, “/Nessus/” matches only the substring “Nessus” if it appears anywhere in the line. If there are characters in the search string, the capitalization must be the same for the string to match. If you want “nEsSuS” to also match, then you need to put an “i” after the second “/” to indicate a case insensitive search.

Regular Expressions

A regular expression is a pattern that describes or matches a set of strings [Wikipedia 05d]. It is a syntax for describing more general criteria for matching strings than simply matching a word in a string. For example, with a regular search you could find the substring “cat” in “catapult.” But a regular expression would let you look for a string that starts with “ca” and ends with “t,” and you would find both “cat” and “catapult.”

For more examples of regular expressions in general, see the Wikipedia entry:

http://en.wikipedia.org/wiki/Regular_expression

For more examples of regular expressions in Perl, see the Perl manual page:

<http://www.perl.com/doc/manual/html/pod/perlre.html>

1.1.3.2.3 Types of Actions

Actions are what the script does when it matches a pattern. They range from printing the log message to the console to executing a separate script to call a pager. There can be multiple

actions associated with each rule. The “ignore” event has its action built in, namely, to stop looking at the log entry. The complete list of actions is found in Table 1 [SourceForge 04]:

Table 1: Actions in Swatch

Action	Description
echo [modes]	Prints the log message to the console. Takes an optional value of a text color, a background color, or a font weight. Possible values are bold, underscore, inverse, blink, black, red, green, yellow, blue, magenta, cyan, white, black_h, red_h, green_h, yellow_h, blue_h, magenta_h, cyan_h, and white_h.
bell [n]	Prints the log message to the console and then rings the bell (007) n times.
throttle H:M:S [,key=log <identifier>] [,time_from=timestamp] [,time_regex=<regex>] [,threshold=N]	throttle reduces the number of times an action is performed on messages matching the same pattern in the specified duration. Hours, minutes, and seconds must all be specified. However, the time does not need to be specified if threshold is being used. The key is the identifier that is stored to compare to new messages. log means use the exact log file, excluding a syslog timestamp, if present, and is the default. Any other string will be used exactly as requested/indicated. <i>Setting the time_from option to timestamp indicates that the time in the log message should be used instead of the system time for comparisons. This is best for examining a log file.</i> The time_regex lets you specify a regular expression to match the timestamp of the message. The default is a regular expression for the syslog timestamp. The threshold=N acts on the first instance and on every Nth instance after that. It repeats counting once N messages have been found. Each instance is appended with “(threshold N exceeded).”
exec command	Executes the command listed. If arguments are needed, they may be substituted with fields in the log message. \$N is replaced with the Nth field in the line. A \$0 or a \$* uses the entire message. The --awk-field-separator switch must be used during execution (see Section 1.1.4).

Action	Description
mail [addresses:bob:joe:...] [, subject=Subject]	Sends an email to the address(es) listed with the subject listed containing the matched log messages. Must have a sendmail compatible server installed. Default recipient is the user who is running the program.
pipe command[, keep_open]	Pipes the log messages into the command. keep_open keeps the pipe open until a different pipe is activated or <i>Swatch</i> exits.
write [user:user:...]	Uses <code>write(1)</code> to send the message to the users listed. The user running <i>Swatch</i> is the default.
continue	Tells <i>Swatch</i> to evaluate this message against the rule set.
quit	Causes <i>Swatch</i> to exit.
when=day:hour	This action is a modifier to all the other actions indicating that the action should occur only during the times specified. For example, when=1-5-8-17 indicates that the action should occur only Monday-Friday between 8 a.m. and 5 p.m.

The `exec` command can be used to write log entries to a file. Redirection from the console does not work because there are unprintable characters on the command line that are printed in the resulting file and are meaningless and in the way. Use this action:

```
exec "echo $* >> output.txt"
```

When examining a file, the most useful actions will be *echo*, *exec*, and *throttle*. The others are more oriented for when you need to be alerted to a new development in real time.

It is also possible to include Perl in the configuration file if you want to do something such as define a regular expression for repeated use. Start each line with “perlcode” and end it with a semicolon.

The *throttle* command can be very powerful. It will take some practice to get the right balance of regular expressions to search for timing and to determine whether *throttle* or *threshold* is better.

For the regular expressions in the `time_regex`, all backslashes must be escaped, (e.g., put in two instead of one). Regular expressions on the action line with commas get cut off, so you need to put the expression in a perlcode. This includes both IIS formats. For example:

```
perlcode my $iis_time = "(\\d{4}-\\d{2}-\\d{2}\\s+\\d{1,2}:\\d{2}:\\d{2}\\s)";
watchfor /WEBROOT DIRECTORY TRANSVERSAL/
    throttle 0:04:00,key=web,time_from=timestamp,time_regex=$iis_time
    echo blue
```

Some log formats may not have `time_regex`. Table 2 lists `time_regexes` for the log files on popular services:

Table 2: `time_regex` for Popular Services

Log File	Timestamp	Regular Expression
Apache access_log	04/Mar/2005:11:38:45	(\d{2}\W\w{3}\W\d{4}:\d{2}:\d{2}:\d{2})
Apache error_log; ssh logs	Fri Mar 04 11:38:51 2005	(\w{3}\s+\w{3}\s+\d{2}\s+\d{2}:\d{2}:\d{2}\s+\d{4})
IIS 6.0 and later	03/04/05, 11:38:51	(\d{2}\d{2}\d{2},\s+\d{1,2}:\d{2}:\d{2}\s)
IIS 5.*, W3C Ex- tended	2005-03-14 11:38:51	(\d{4}-\d{2}-\d{2}\s+\d{1,2}:\d{2}:\d{2}\s)

The *when* command can be useful for identifying events that occur at abnormal times. Use it to look for login attempts in the middle of the night.

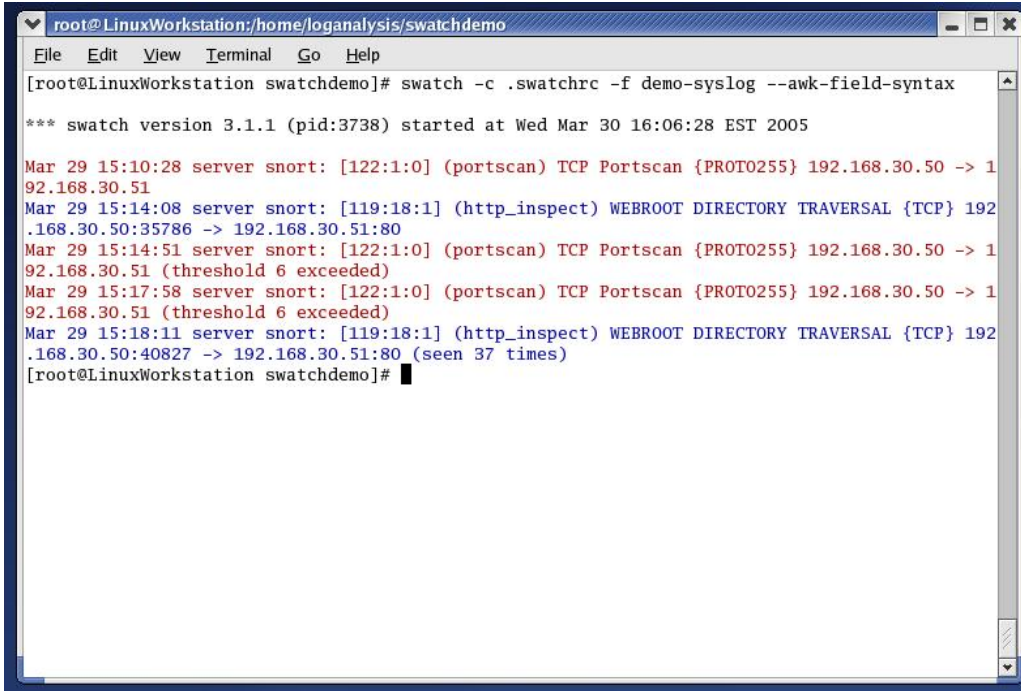
The following is an example configuration file for a syslog setup that has two machines, named “server” and “client,” logging to one file. The central syslog server resides on the server. All facilities on the client log to this file. The client is running the default applications. The server logs most facilities to the file and is running Apache (httpd) and Snort, in addition to the basics.

```
# Copies all of the entries from the client to a separate file
watchfor / client /
    exec "echo $* >> client.log"      #echoes the log entry to a file

# Prints one entry every 4 minutes (based on the time in the logfile #for
any entries containing a Nessus attack keyword.
watchfor /WEBROOT DIRECTORY TRANSVERSAL/
    throttle 0:04:00,key=webroot,time_from=timestamp
    echo blue

# Searches for snort followed by portscan somewhere in the log entry
# Prints the first entry and then every 6th one both to the console and
# a separate file
watchfor /snort.*portscan/
    throttle threshold=6,key=ps
    echo red
    exec "echo $* >> portscan.log"
```

Figure 1 is an example run of the configuration file on a syslog file (“(portscan)” and “(http_inspect)”) distinguish the different types of messages).



```
root@LinuxWorkstation:/home/loganalysis/swatchdemo
File Edit View Terminal Go Help
[root@LinuxWorkstation swatchdemo]# swatch -c .swatchrc -f demo-syslog --awk-field-syntax

*** swatch version 3.1.1 (pid:3738) started at Wed Mar 30 16:06:28 EST 2005

Mar 29 15:10:28 server snort: [122:1:0] (portscan) TCP Portscan {PROTO255} 192.168.30.50 -> 192.168.30.51
Mar 29 15:14:08 server snort: [119:18:1] (http_inspect) WEBROOT DIRECTORY TRAVERSAL {TCP} 192.168.30.50:35786 -> 192.168.30.51:80
Mar 29 15:14:51 server snort: [122:1:0] (portscan) TCP Portscan {PROTO255} 192.168.30.50 -> 192.168.30.51 (threshold 6 exceeded)
Mar 29 15:17:58 server snort: [122:1:0] (portscan) TCP Portscan {PROTO255} 192.168.30.50 -> 192.168.30.51 (threshold 6 exceeded)
Mar 29 15:18:11 server snort: [119:18:1] (http_inspect) WEBROOT DIRECTORY TRAVERSAL {TCP} 192.168.30.50:40827 -> 192.168.30.51:80 (seen 37 times)
[root@LinuxWorkstation swatchdemo]#
```

Figure 1: Example Run of the Swatch Configuration File

Swatch Execution

Swatch is run from the command line.

- `swatch -c .swatchrc -f file.log`
- This runs *Swatch* using the file *.swatchrc* as the configuration file on *file.log*.
- *Swatch* can be run on only one file at a time, but multiple instances of *Swatch* can be running at once.

Switch	Use
-c config.file	Name a specific configuration file
-f file.log	Examine the specific file
-t file.log	Tail the specific file (/var/log/messages is the default)
-p command	Accepts its input from this command

2005 Carnegie Mellon University

8



1.1.4 Swatch Execution

Swatch is run on the command line and has many options to specialize the execution. You can identify the configuration file, the log file, how to monitor the log file, and even what character(s) indicate a new log message. While most options will be listed, the focus is on the options that relate to running *Swatch* on a log file in a single pass. More information about all the options can be found in the manual page.

`-c filename` or `--config-file=filename`

Indicates where the configuration file is. The default location is `$HOME/.swatchrc`.

`-f filename` or `--examine=filename`

Indicates that *Swatch* should perform a single pass on the log file.

`-t filename` or `--tail-file=filename`

This option enables *Swatch* to monitor a file as a service continues to log to it. This action is the default, so if none of `-f`, `-p`, or `-t` is given, *Swatch* will tail either `"/var/log/messages"` or `"/var/log/syslog."`

`-p command` or `--read-pipe=command`

Monitors the data that is being piped in from the given command.

`--awk-field-syntax`

Tells *Swatch* to use the syntax for awk expressions. It is needed when the *exec* action is used.

`--input-record-separator=regular_expression`

This option indicates that the default record separator of carriage return should be replaced with the regular expression listed.

Other options include `--help` and `--version`, which respectively give usage information and the current version; `--script-dir=path`, which indicates where the temporary script should be stored if not in the user's home directory; and `--restart-time=hh:mm[am|pm]`, which tells *Swatch* to restart at a particular time.

The most common usage will be

```
swatch -c .swatchrc -f log_file
```

Remember to specify the complete path of the configuration file or the log file if either one is not located in the local directory.

Summary

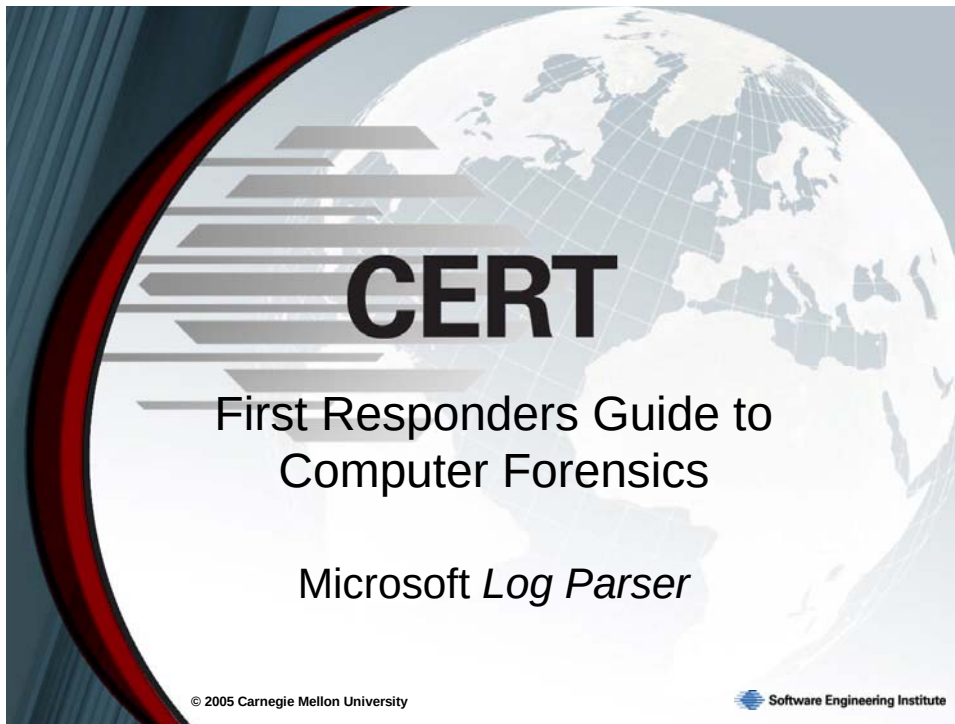
Swatch is a simple log monitor designed to run regular expressions against a text log file.

- an effective tool for finding interesting or anomalous events
- configured easily to watch for any type of entry in any type of file

Many actions can be taken on matching message logs, but for analysis “echo” will be used most often.

1.1.5 Summary

Swatch can be an effective tool for sifting through log files to find interesting or anomalous events. The results serve as a jumping off point for further inspection of the files by leading you directly to areas of interest. Or it can be used to filter out entries that you know can be excluded so as to reduce the amount of material to examine by hand. *Swatch* can also be used on other files, such as VMWare virtual disk files and disk images, to efficiently find strings in data.



1.2 Microsoft *Log Parser* 2.2

This topic is an overview of the installation, configuration, and usage of Microsoft's *Log Parser* 2.2. The focus is on using *Log Parser* as a forensic analysis tool, meaning that it will be used on logs after an incident has occurred. If you are planning to use these techniques on files involved in a law enforcement investigation, please make sure you prepare the files according to established best practices.

Overview

Why Log Parser?

Installation

Writing Queries

Execution

Forensics in this case is the act of looking for events after they have happened, not performing an investigation for law enforcement



Microsoft *Log Parser* 2.2

Command line tool from Microsoft to process log files using SQL-like queries

Can read in many formats and many types of log files

Output available in many formats—from text to XML files to database storage

Easy generation of HTML reports and MS Office objects

Log Parser functions usable in other scripts

2005 Carnegie Mellon University

3



1.2.1 Microsoft *Log Parser* Features

Microsoft *Log Parser* 2.2 is the most recent incarnation of Microsoft's log analysis tool. Released in January 2005, it contains many improvements and additions to make it useful to anyone with a log file to process. *Log Parser* is free to download and use. It is a command line tool; there is no GUI to make creating commands easier.

Log Parser uses SQL-like queries to sort through log files. It is very flexible and can be used to read any text based file, file system objects, registries, and database formats.

Log Parser can format the text output into a variety of formats. It can also send output directly to a syslog server, a database, and MS Office charts, and can be used to create HTML reports from a template.

Log Parser can also be used in other programs and scripts to bring its processing power to other applications.

Log Parser Installation

Requirements: Windows 2000, 2003, or XP Professional

Download *Log Parser* from Microsoft's website

Double-click on the setup file and follow the instructions

The Unofficial Log Parser Support Site is an excellent resource: <http://www.logparser.com/>

1.2.2 Log Parser Installation

Log Parser is a Microsoft Windows application that runs on Windows 2000, 2003, and Windows XP Professional. Installation is quick and easy.

Download the file from the Microsoft website. In addition to the Microsoft website, The Unofficial Log Parser Support Site maintains a current link to the setup file on its home page: <http://www.logparser.com>. That site is also an excellent resource for *Log Parser* information.

Once you've downloaded the file, double-click on the setup file, *LogParser.msi*. Follow the instructions in the Setup Wizard and *Log Parser* is installed.

Log Parser Input and Output

Log Parser can read many text-based log formats

- Use the switch `-i:TYPE` to indicate file type
- Default is determined from the input file type
- Type `LogParser -h -i:TYPE` for more information on a specific type

Output can be formatted into text files or MS Office objects or sent to other programs

- Use the switch `-o:TYPE` to indicate type of report
- Default is determined from the name of the output file
- Type `LogParser -h -o:TYPE` for more information on a specific type

1.2.3 Log Parser Input and Output

Log Parser has a variety of built-in text-based formats that it can use to easily parse files and several output formats it can create. Many of them correspond to the log formats of popular applications, though it is Windows-centric.

1.2.3.1 Input Formats

To specify an import format, use the switch `-i:TYPE`, where *TYPE* is one of the built-in types. The default input type is determined by *Log Parser* based on the file extension in the *FROM* clause. Table 3 lists many of the types and application logs for which each can be used. More information and usage examples for each one can be found by using the command line help: `LogParser -h -i:TYPE`. Other types can be found under `LogParser -h`.

Table 3: Common Input Formats

Type	Uses	Selected Parameters
IISW3C	IIS W3C Extended Log Format, primarily IIS 5.X logs and older	n/a
IIS	Microsoft IIS log format, mostly used with version 6.0 and newer	n/a
NCSA	NCSA Common, Combined, and Extended Log formats, for Apache logs	n/a

Type	Uses	Selected Parameters
CSV	Text files with comma-separated values	-headerRow [ON OFF] – for specifying if there is a header row -iTsFormat <timestamp format> – for specifying timestamps other than “yyyy-MM-dd hh:mm:ss”
TSV	Tab or space separated values	-headerRow [ON OFF] – for specifying if there is a header row -iSeparator <character> – Character that indicates a new field; can be any character, “spaces,” “space,” or “tab”
W3C	Generic W3C log format	-iSeparator <character> – Character that indicates a new field; can be any character, “spaces,” “space,” or “tab”
XML	XML formatted logs	-rootXPath <XPath> – XPath query of nodes to be considered roots
EVT	Windows Event Log	-fullText [ON OFF] – Use the full text message
NETMON	NetMon captures files	-fMode [TCPIP TCPConn] – Field mode, each record is a single packet or a single connection
REG	Registry keys and values	n/a
ADS	Active Directory objects	-objClass <class name> – Specific class for object mode -username <uname> – user name for AD access -password <pwd> – password for the specified user
TEXTLINE	Parses entire lines of text files	n/a
TEXTWORD	Parses single words out of generic text files	n/a
FS	File system properties	n/a
COM	Custom COM input format	-iProgID <progid> – version independent Prog ID of the COM plug-in

1.2.3.2 Output Formats

To specify an output format to create, use the switch `-o:TYPE`. Output can be in several text formats as well as some non-text ones. The default, for when no `INTO` clause is stated, is to print the list using NAT to `STDOUT`. When the `INTO` clause is stated, if there is no `-o:TYPE`, then *Log Parser* will attempt to guess from the file extension. Table 4 lists many of the types. More information and usage examples can be found by using the command line help: `LogParser -h -o:TYPE`. Other types can be found under `LogParser -h`.

Table 4: *Output Formats*

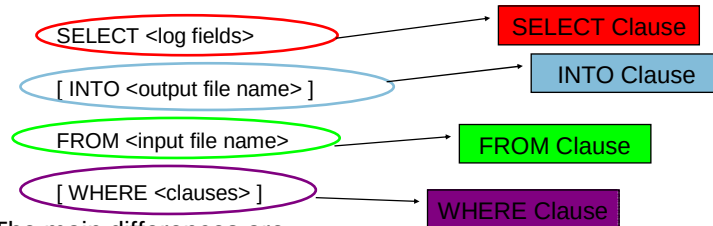
Type	Format	Special Parameters
CSV	Comma-separated values	-headers [ON OFF AUTO] – Write field names as the first line; AUTO won't write headers when appending to an existing file
TSV	Tab-separated values	-headers [ON OFF AUTO] – Write field names as the first line; AUTO won't write headers when appending to an existing file -oSeparator <any string> – Separator between fields: <string>, "space," or "tab"
XML	XML output format	-rootName <element name> – Name of the Root element
DATAGRID	ASP.NET data type for displaying tabular data	-rpt <number of rows> – Rows to print before pausing
CHART	Microsoft Office Chart web component; MS Office must be installed for use	-chartType <chart type> – One of the designated chart types
SYSLOG	Sends output to a syslog server	-hostName <hostname> – syslog server name
NAT	Native format – tabular view of the records	-rtp <number of rows> – Rows to print before pausing
W3C	W3C Extended Log format	-rtp <number of rows> – Rows to print before pausing
IIS	IIS Log format	-rtp <number of rows> – Rows to print before pausing

Type	Format	Special Parameters
SQL	Sends the output to a SQL table	-server <server name> – Server the database is on -database <database name> – Name of the database
TPL	Output a report in a user-specified template	-tpl <template file path> – Path of the template file -tplHeader <header path> – Location of a header file to use

Log Parser Queries

Log Parser can be run with standard SQL queries on the log files

Format:



The main differences are

- inclusion of the INTO clause with the output filename
- FROM clause specifies a filename, not a table

Help with SQL can be gotten from *Log Parser*: `LogParser -h GRAMMAR`

2005 Carnegie Mellon University

6



1.2.4 Log Parser Queries

Log Parser queries are based on standard SQL queries. This enables users to be able to select the fields that they need displayed, the log entries to ignore, and which to act on. This also allows for aggregation of log information, such as counting the number of “Error 500” responses from a web server.

The basic parts of a SQL query are the **SELECT** clause, the **FROM** clause, and the optional **WHERE** clause. **SELECT** indicates the fields to return. **FROM** indicates the data source to use. And **WHERE** indicates any conditions for rows to be included in the results.

There are a few differences between standard SQL and the *Log Parser* SQL. First, the output location needs to be specified with the **INTO** clause in *Log Parser* SQL. This is generally a filename, though it might also be **STDOUT** if displaying the information on the console is desired. Next, the input is from a file, rather than from a table, so a filename follows the **FROM** clause. Lastly, the input is limited to one file. Joins are not allowed, though subqueries are allowed.

There is query documentation in the help command using `LogParser -h Grammar`. There are also a number of examples, both under the help sections for specific input and output formats as well as under a special examples topic: `LogParser -h Examples`. For more references on how to write SQL queries, see <http://en.wikipedia.org/wiki/SQL>.

Log Parser Query Examples

Filtering out irrelevant entries

```
SELECT *
INTO output.log
FROM input.log
WHERE sc-status >= 400
```

Adjusting timestamps

```
SELECT LogFilename, LogRow, RemoteHostName,
RemoteLogName, UserName,
To_Localtime(Add(DateTime, Timestamp('00:00:10',
'hh:mm:ss'))) AS DateTime,
Request, StatusCode, BytesSent, Referer, UserAgent, Cookie
FROM access_log
```

2005 Carnegie Mellon University

7



1.2.4.1 Query Examples

1.2.4.1.1 Filtering out Irrelevant Entries

When focusing on a web server's logs, you will probably want to separate out entries where page requests failed to reduce the logs to the important information. The query

```
SELECT * INTO output.log FROM input.log
WHERE sc-status >= 400
```

would return all of the log entries where the status code was 400 or above, indicating any server errors, "Page Not Found" errors, and access denials. By filtering out what is known to not be a problem, (e.g., successful page requests), it reduces the log files to a more manageable size. Please note that this example is for IISW3C input; different input formats will have different field names.

1.2.4.1.2 Adjusting Timestamps

Another common problem is that sometimes the time on one machine may drift and not be the same as on another machine. When you try to compare logs generated on these two machines, events may be out of order. *Log Parser* can be used to adjust a timestamp in a file. Since we care about the entire log entry and not just the adjusted timestamp, we will need to specify the timestamp and all the other fields. This example uses the NCSA input, is formatted for W3C output, and adds 10 seconds to all the entries.

```
SELECT LogFilename, LogRow, RemoteHostName, RemoteLogName, UserName,
To_Localtime(Add(DateTime, Timestamp('00:00:10', 'hh:mm:ss')))
```

```
        AS DateTime, Request, StatusCode, BytesSent, Referer,  
        UserAgent, Cookie  
FROM access_log
```

This can also be done on IISW3C formatted logs, though the timestamp is different so a different command is needed:

```
To_LocalTime(Add(To_Timestamp(date, time), Timestamp('00:00:10',  
        'hh:mm:ss')))
```

This query will not change the timestamps in the original files but will create a new file with the adjusted timestamps, making log comparison easier.

Log Parser COM Objects

COM allows for cross-platform development. It can be used in C++, C#, Visual Basic, Jscript, and VBScript.

COM Objects can be used in two ways:

1. Create custom input formats (the syslog format is not currently supported).
2. Use *Log Parser* functions in other programs—add queries and input formats to your own scripts.

2005 Carnegie Mellon University

8



1.2.5 Log Parser COM Objects

Component Object Model (COM) objects allow for cross-platform development of programs and scripts. *Log Parser* comes with a COM Application Programming Interface (API), allowing programmers to use the underlying constructions of *Log Parser* either to extend its capabilities or for use in their own programs.

Log Parser COM API is available for use in C++, C#, Visual Basic, JScript, and VBScript. The only adjustment needed to use the API is that the *LogParser.dll* binary needs to be registered with the computer's COM infrastructure so the API will be found. Use this command:

```
C:\LogParser>regsvr32 LogParser.dll
```

1.2.5.1 Creating Custom Input Formats

If the built-in input formats are too restricting, you can create your own. There is one interface to extend in C++ or Visual Basic and another for JScript and VBScript. The resulting script must also be registered with the computer, as in the example above. After that, it can be used when running *Log Parser*, as in this example:

```
C:\LogParser>LogParser "SELECT * INTO out.file FROM in.file" -i:COM -iProgID:MySample.MyInputFormat
```

For more examples of this, please refer to the documentation that comes with *Log Parser 2.2*.

1.2.5.2 Using the *Log Parser* COM API

The *Log Parser* COM API allows for all the same actions as the command line binary, plus more. In addition to query execution and the various input and output formats, it is also possible to have direct access to the log entries in record format.

The MSUtil.LogQuery object is the base object for *Log Parser*. All others are subclasses. There are subclasses for each of the input and output types, as well as the LogRecordSet and LogRecord types.

There are two modes of query execution: batch and interactive. Batch execution is used when the output will be formatted with one of the output formats, as is done in the command line program. Interactive execution is when no output format is specified and a LogRecordSet is returned. The programmer can then move through the set and process individual records as desired.

The *Log Parser* documentation provides further illustration of these concepts and specific commands that are available.

Log Parser Execution

The Log Parser program opens a command window in the Log Parser directory.

Log Parser is run in two ways:

1. entering the query at the command line
LogParser "SELECT * INTO out.file FROM in.file" -i:TYPE -o:TYPE <other switches>
2. using a saved query
LogParser file:query.sql -i:TYPE -o:TYPE



1.2.6 Log Parser Execution

Log Parser will normally be run on the command line in the special *Log Parser* window. To start *Log Parser* go to Start → Programs → Log Parser 2.2 → Log Parser 2.2. This opens a command window, displays the help information, and then gives a command prompt with which to work.

On the command line, you list the query, input format, and output format. Certain formats also have other parameters that need to be entered using some extra commands. These commands are listed in Table 5. Switches are separated from their parameters by a colon. The format of the line is

```
>LogParser "SELECT * INTO out.file FROM in.file" -i:TYPE -o:TYPE  
<switches>
```

Table 5: Misc Log Parser Commands

Command	Function	Parameters
-q	Quiet mode	ON or OFF (default)
-e	Maximum number of errors allowed before aborting	integer, -1 is default (ignore all)
-iw	Ignore warnings	ON or OFF (default)
-stats	Display statistics after executing query	ON (default) or OFF

Command	Function	Parameters
-c	Use built-in conversion query	-i:TYPE <filename> -o:TYPE <filename>
-multiSite	Send any BIN conversion output to multiple files depending on the SiteID value	ON or OFF (default)
-saveDefaults	Save options as default values	none
-restoreDefaults	Restore factory defaults	none
-queryInfo	Display query processing information, but do not execute query	none

Queries can get long and it can be cumbersome to type the same one over and over. You can specify a file that contains a query on the command line instead of the query itself. Type `file:query.sql` instead of the full query. This will be particularly useful for queries that convert from one file type to another (aside from the predefined conversions) and for queries that you run repeatedly.

Summary

Log Parser 2.2 is an extremely flexible tool for parsing and searching through logs, *Event Viewer* files, XML, and other text-based files.

Use it to

- convert logs from one format to another
- filter out specific types of log entries into a new file
- create a new log with an adjusted timestamp
- write custom programs and scripts to process log files

It assumes prior knowledge of SQL.



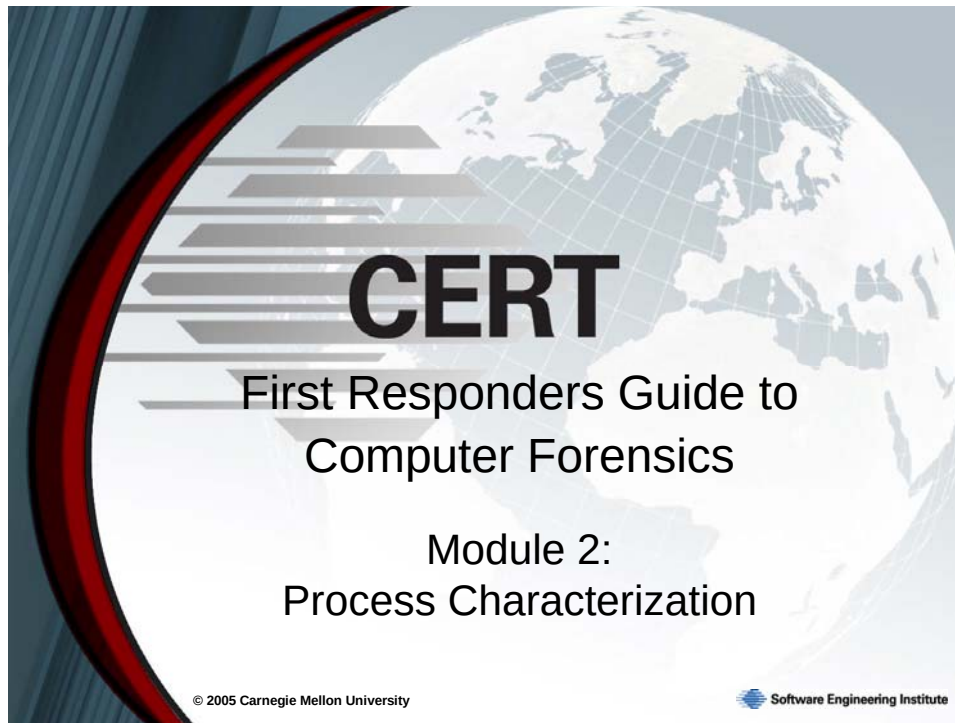
1.2.7 Summary

Log Parser 2.2 is such an improvement on previous versions that it is like a new program. *Log Parser* is as extensible and flexible as you can make it. It can be used for

- converting log files from one format to another for ease of analysis
- filtering out specific types of log entries into a new log
- creating a new log with an adjusted timestamp after skew has been determined
- writing custom programs and scripts to process log files

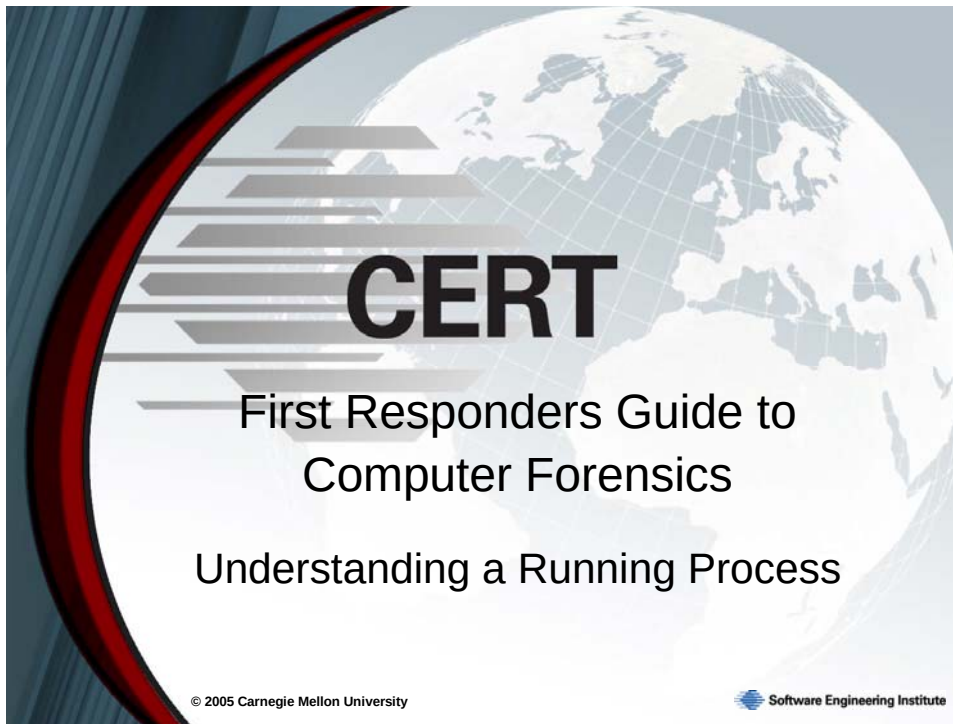
There is a steep learning curve with *Log Parser*. It is necessary to know the fundamentals of SQL queries to be able to process logs effectively. Once this limitation is overcome, many standard logs can be processed and reduced to create meaningful output.

2 Module 2: Process Characterization



The identification, characterization, and forensic collection of currently running processes on a PC should be a frequently practiced information security procedure. Baselining a running system's processes frequently and enumerating the list of currently running processes will allow you to monitor system activity and see whether serious changes have been made to the system. Running processes on a PC are at the crux of either normal or abnormal system behavior. However, even after process collection, it can be a difficult task to make the determination whether a particular process or set of processes may be the result of an intrusion or malicious user activity.

For most system users and security practitioners, the first alert to abnormal system behavior may be the trivial questions one has about the system during routine day to day interactions with the PC. Why is my PC responding so sluggishly? Why does my PC show extremely high processor activity? What is that process and why is it running? These are commonly asked questions. In most cases, the default reaction to abnormal system behavior is the widely practiced reboot. This second-nature reaction may temporarily solve a problem, but if the machine was truly infected or compromised you may never find the source of the problem because of the volatility of running processes.



2.1 Understanding a Running Process

This module is intended to enable system users and first responders (system and network administrators, law enforcement, etc.) to

- better understand running processes
- forensically collect and enumerate the set of current running processes on a system
- potentially differentiate between normal running processes and abnormal running processes (i.e., malware) by looking at key process characteristics

Objectives

Background info on processes

- Programs, processes, threads
- Process tree structure
- Process identifiers (PIDs)
- Process descriptions
- Process analysis checklist

8 key process characteristics

- Tools and native commands for collecting process characteristics

3 process forensic tasks

2005 Carnegie Mellon University

3



2.1.1 Objectives

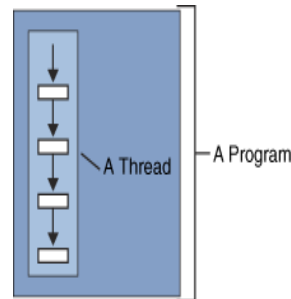
The focus of this module is to demonstrate how to determine the existence of running malware by performing basic process characterization and the forensic examination of running processes on a system. The module is limited to running processes because the majority of malware (viruses, worms, Trojans, backdoors, etc.) have an associated process that may be aliased, newly created, or masked that allows the malware to perform its malicious actions on a machine and sometimes even remain or replicate after a reboot.

Therefore, in an attempt to identify potential rogue processes, we will first cover some background information on processes, identify eight key process characteristics, demonstrate forensic collection procedures, and, finally, introduce native commands and tools that will allow a first responder to forensically collect the key process characteristics for running processes.

Programs, Processes, Threads

A computer process can be best defined as a program in execution. Generally, a process consists of the following:¹

- executable code
- data
- the execution context (e.g., the contents of certain relevant CPU registers)



Threads are execution contexts for a process.

¹ Gollman, Dieter. *Computer Security*. England: John Wiley & Sons Ltd, 1999.

2.1.2 Programs, Processes, and Threads

A computer process can best be defined as a program in execution. Generally, a process consists of the following:³

- executable code
- data
- the execution context (e.g., the contents of certain relevant CPU registers)

While the word “program” refers to the executable code (the exe file, for example), a process is a program that is being executed. When you start a program in Windows, the executable will be loaded into RAM. Windows will then add the new process to its internal process list and make sure the process receives some CPU time as well as memory and other resources. A process can then request any amount of resources from Windows as long as there are resources left. Windows keeps track of which processes are using which resources. As soon as a process is closed or terminated, all resources used by that process will be returned to Windows and will then be handed out to other processes. Unlike memory and similar resources, CPU time cannot simply be requested but is instead shared equally between processes. A process can also return the CPU to Win-

³ Gollman, Dieter. *Computer Security*. England: John Wiley & Sons Ltd, 1999.

dows before the assigned time slice ends. This is actually what happens most of the time and is the reason why your CPU usage is not always at 100 %.⁴

The Linux Tutorial⁵ is a great tutorial for understanding in depth how processes work and also provides interactive demonstrations describing the parent-child process relationship.

2.1.3 Threads

Threads are execution contexts. Initially each process has a single execution context. This execution context is called a thread. If a process requires another execution context, it can simply create another process. Threads were invented to provide a lightweight mechanism for creating multiple execution contexts. Windows and Linux schedule threads from the operating system with the goal of providing a fair execution environment.

The most obvious distinction between processes and threads is that all threads of a process share the same memory space and system-defined “facilities.” Facilities include open file handles (file descriptors), shared memory, process synchronization primitives, and current directory. Because global memory is shared and almost no new memory must be allocated, creating a thread is simpler and faster than creating a process.⁶

2.1.3.1 Displaying Threads for a Running Process

Using the Sysinternals⁷ *PsList* command line utility with the `-d` command line argument will display currently running processes, threads for each process, each process’s thread state, and memory statistics for each process. This utility comes in very handy when you need a quick way of enumerating all currently running processes, each process’s associated threads, and their thread state.

Idle	Tid	Pri	Cswtch	State	User Time	Kernel Time	Elapsed Time
0:	0	0	13925748	Running	0:00:00.000	12:44:40.062	0:00:00.000
System 4:							
	Tid	Pri	Cswtch	State	User Time	Kernel Time	Elapsed Time
	8	0	132098	Ready	0:00:00.000	0:00:19.668	0:00:00.000
	16	13	26322	Wait:Queue	0:00:00.000	0:00:00.931	43:22:13.277
	20	15	24921	Wait:Queue	0:00:00.000	0:00:00.801	43:22:13.277
	24	15	19026	Wait:Queue	0:00:00.000	0:00:00.360	43:22:13.277
	28	13	66433	Wait:Queue	0:00:00.000	0:00:03.935	43:22:13.277
	32	13	48187	Wait:Queue	0:00:00.000	0:00:01.922	43:22:13.277
	36	12	318579	Wait:Queue	0:00:00.000	0:00:04.866	43:22:13.277
	40	12	353811	Wait:Queue	0:00:00.000	0:00:06.148	43:22:13.277
	44	12	147279	Wait:Executive	0:00:00.000	0:00:03.414	43:22:13.277
	48	12	253695	Wait:Queue	0:00:00.000	0:00:04.556	43:22:13.277
	52	12	19299	Wait:Queue	0:00:00.000	0:00:02.463	43:22:13.277
	56	12	54114	Wait:Queue	0:00:00.000	0:00:04.816	43:22:13.277
	60	12	185057	Wait:Queue	0:00:00.000	0:00:04.877	43:22:13.277
	64	15	1473774	Wait:Queue	0:00:00.000	0:00:02.052	43:22:13.277
	68	15	51677	Wait:Executive	0:00:00.000	0:00:00.000	43:22:13.277
	72	18	11427	Wait:VirtualMem	0:00:00.000	0:00:00.650	43:22:13.257

Figure 2: Example Run of PsList

⁴ <http://www.liutilities.com/products/wintasksp/whitepapers/paper8/>

⁵ <http://www.linux-tutorial.info/modules.php?name=Tutorial&pageid=3>

⁶ Bradford, Edward “High-Performance Programming Techniques on Linux and Windows 2000,” <http://www.developertutorials.com/tutorials/linux/run-time-linux-windows-050428/page1.html>.

⁷ <http://www.sysinternals.com/index.html>

2.1.3.2 Sysinternals *Process Explorer*

Another Sysinternals utility called *Process Explorer* is an excellent administrative tool for showing a dynamic display of real-time system process activity. *Process Explorer*'s graphical user interface (GUI) is a more robust Windows Task Manager. It displays in an easy to read format what handles each process has open and what DLLs and memory-mapped files each process has loaded, and has a quick search capability to locate a particular DLL or handle for the currently running processes.⁸

Figure 3 is an action screenshot of the *Process Explorer* GUI. The GUI displays a process list (e.g., tree format) of currently running processes in the top half of the pane, while the bottom half of the pane displays all currently loaded DLLs for the highlighted *mozilla.exe* process.

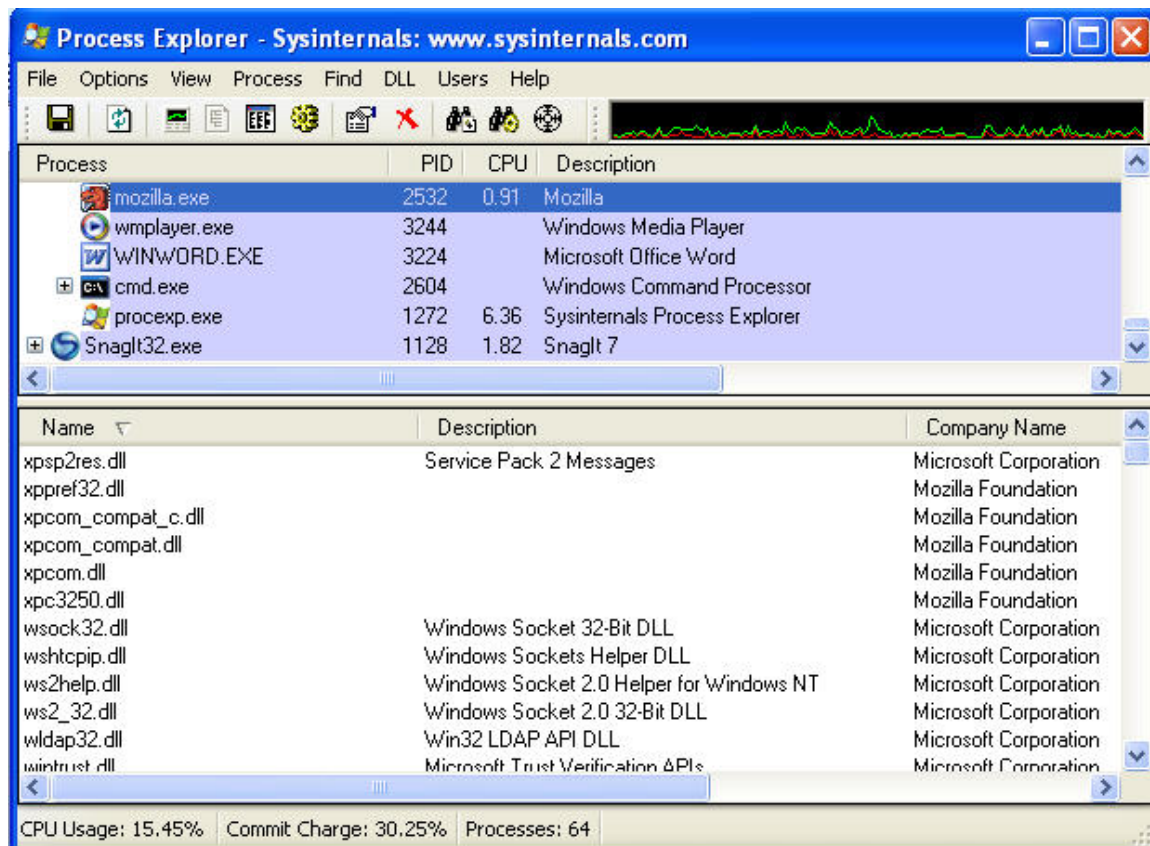
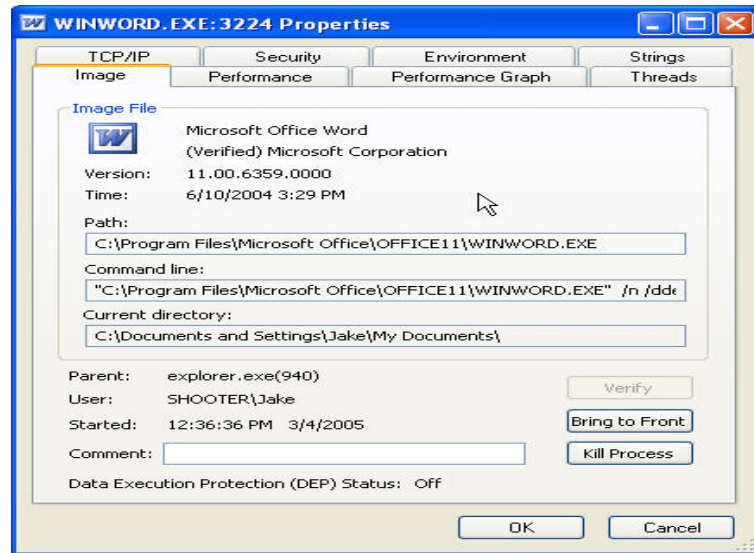


Figure 3: Sysinternals *Process Explorer* Utility

⁸ <http://www.sysinternals.com/ntw2k/freeware/procexp.shtml>

If you are curious about whether a particular running process is a legitimate Microsoft process, you could verify the process image by using the added functionality in *Process Explorer*.

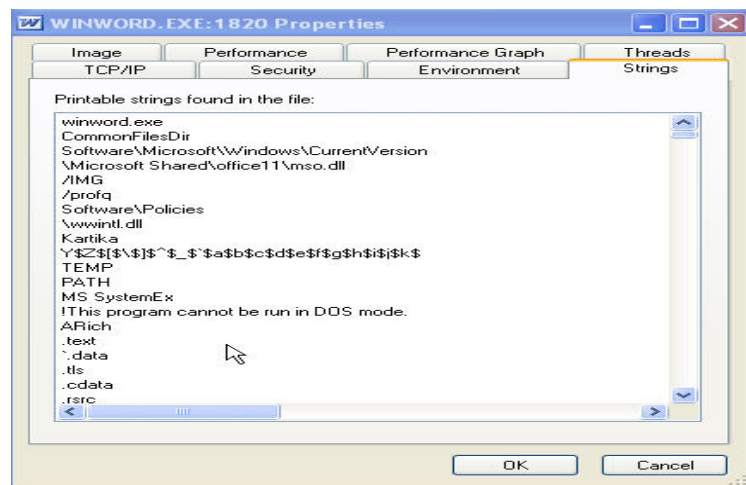


To do this, you would

1. Right-click on any of the displayed processes in the *Process Explorer* GUI.
2. Click Properties.
3. Click Verify. An example result is shown in Figure 4.

In addition to the Verify option, another great feature is the Strings tab.

Clicking on the Strings tab will display all alpha and numeric strings found in the process image, as shown in the example in Figure 5. This may come in handy for looking further into an unknown process image.

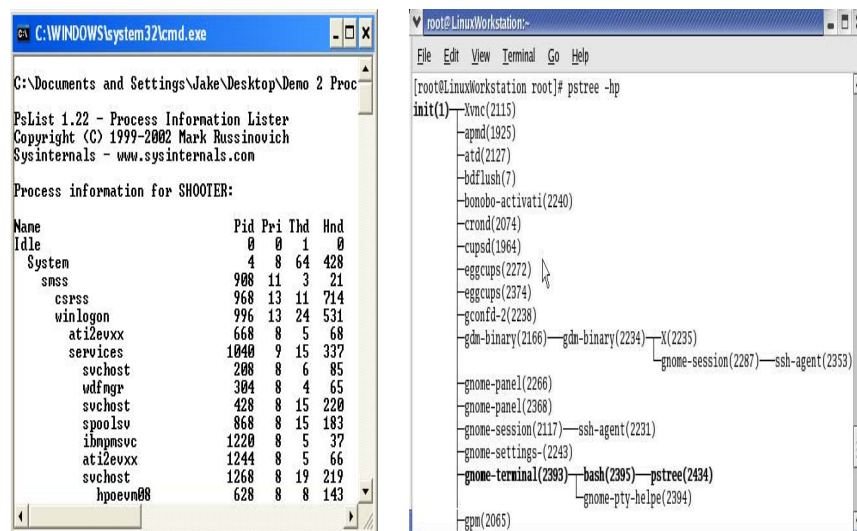


It is important to point out that this utility uses a GUI

and, therefore, is not an ideal tool that a first responder would use in responding to a computer security incident to forensically analyze running processes on a machine. This utility should have already been incorporated into the daily operations for everyday system and network troubleshooting. If you were to use the *Process Explorer* utility in an incident response situation you might actually contaminate potential uncollected evidence from a possibly compromised machine (e.g., changing Mac Times on critical files or folders on the system). Sysinternals' command line utility *pslist.exe* is a better choice for responding to

computer security incidents and forensically collecting running processes, simply because it is much lighter and leaves a significantly smaller footprint.

Process Tree Structure and PIDs



2005 Carnegie Mellon University

5

CERT

2.1.4 Process Tree Structure

Windows and Linux OS environments currently running processes exhibit a hierarchical tree structure. By looking at this hierarchical tree structure of currently running processes we can gain insight about what processes have started other processes and so forth.

This kind of relationship is known as the parent and child process relationship. The creator is called the parent process, the created is called the child process, and the parent-child relationship is expressed by a process tree.

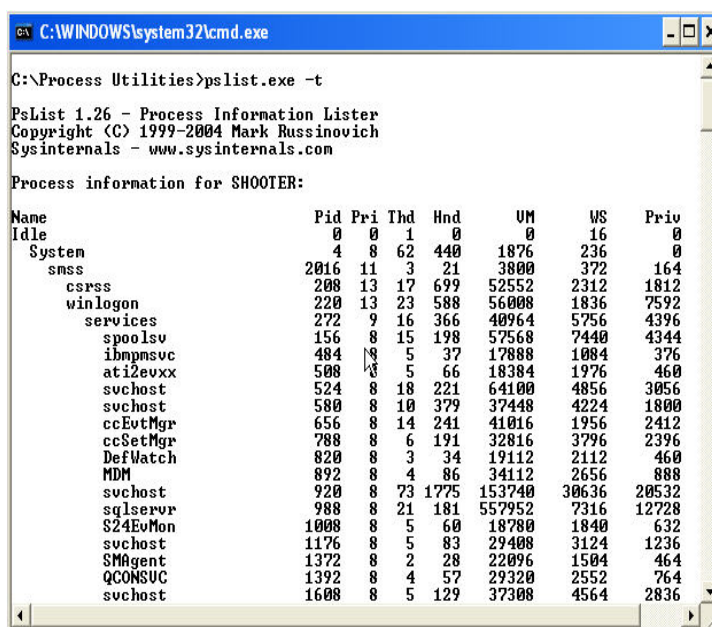


Figure 6: Displaying a Process Tree Using PsList

Using Sysinternals' *PsList* utility with the `-t` command line argument, we can visually display the parent-child relationship (i.e., the process tree) for currently running processes.

In the example shown in Figure 6, we can easily see that the *System.exe* process is the parent process for *smss.exe* and so forth as you work your way down the process tree.

2.1.4.1 *pstree* (Linux)

Using the native Linux *pstree* command, we can easily display the parent-child process relationship for currently running processes. Using *pstree* with the *-hp* command line argument will display the process tree, highlight the current process, and display the PIDs for each process (Figure 7).

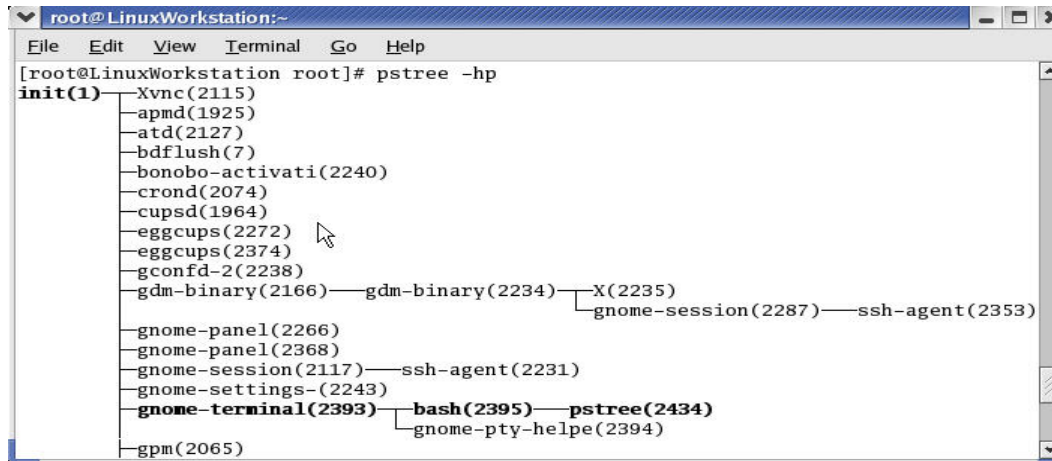


Figure 7: Displaying a Process Tree Using *pstree*

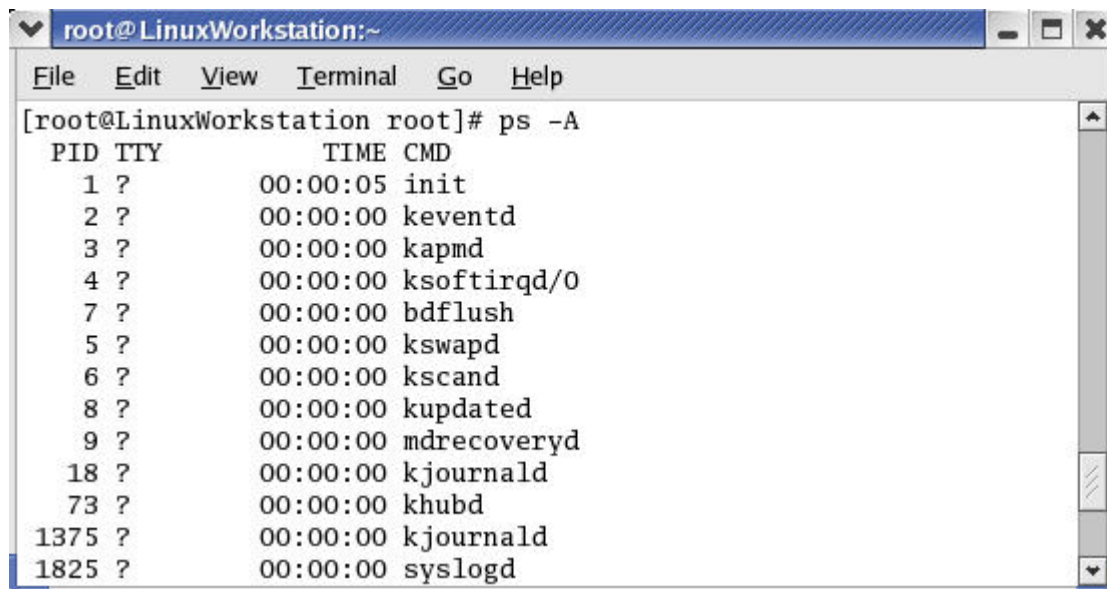
2.1.4.1.1 PIDS

Process identifiers, commonly known as PIDs, are unique integer values assigned to each currently running process. Generally, PID assignments will be multiples of 4 for Windows operating systems, thus guaranteeing an even integer PID assignment ranging from 0–XXXX.

From a forensic or first responder perspective, PIDs offer a quick and easy way of uniquely identifying running processes. However, there is nothing of forensic value in distinguishing legitimate processes from non-legitimate processes by just looking at the PID assignments. What is useful is the mapping or correlation from the PID assignments to generated system event log tickets. We can search the system event logs for a PID that was responsible for generating a certain ticket's event log and map them back to the PID or current running process.

2.1.4.2 Linux *ps -A*

Using Linux's native *ps* command, we can quickly display each currently running process's PID assignment and the command that was used to start the process. When displaying processes in Linux, PIDs will always be displayed in sequential order ranging from the infamous *init* process, or PID 1, to XXXX (Figure 8).



```
[root@LinuxWorkstation root]# ps -A
```

PID	TTY	TIME	CMD
1	?	00:00:05	init
2	?	00:00:00	keventd
3	?	00:00:00	kapmd
4	?	00:00:00	ksoftirqd/0
7	?	00:00:00	bdflood
5	?	00:00:00	kswapd
6	?	00:00:00	kscand
8	?	00:00:00	kupdated
9	?	00:00:00	mdrecoveryd
18	?	00:00:00	kjournald
73	?	00:00:00	khubd
1375	?	00:00:00	kjournald
1825	?	00:00:00	syslogd

Figure 8: Displaying PID Assignments Using ps

Process Descriptions

WinTasks Process Library

- Great resource for knowing the exact purpose and description of every single Windows process

Categories of Windows processes

- Top system processes
- Top application processes
- Top security threat processes
- Other unfamiliar processes

Example: [svchost.exe](#)

2005 Carnegie Mellon University

6



2.1.5 Process Descriptions

There is no easy way of quickly knowing whether the current set of running processes are normal, especially if you do not have in-depth knowledge of the system you are analyzing or if proper process baselining was not implemented. Understanding what each process is and why it is currently running can be a difficult task. There are a few online resources outlined below that will attempt to alleviate some of the ambiguity of unfamiliar Windows processes, particularly in determining whether a process is legitimate.

Uniblue has an online hyperlinked table for each type of Windows process. The online resource is a great tool for quickly checking and gathering information about a known rogue process or gaining information such as a description about any legitimate Windows process. A Uniblue hyperlinked table for each of the following process categories is available online.⁹

Categories of Windows Processes

- top system processes
- top application processes
- top security threat processes
- other, unfamiliar processes

Figure 9 is a snapshot of the type of information you can obtain from the Uniblue website for each of the categories of Windows processes. We chose a process description pertaining to the *svchost.exe* process that is often found running on Windows systems.

⁹ <http://www.liutilities.com/products/wintasksp/processlibrary/allprocesses/>

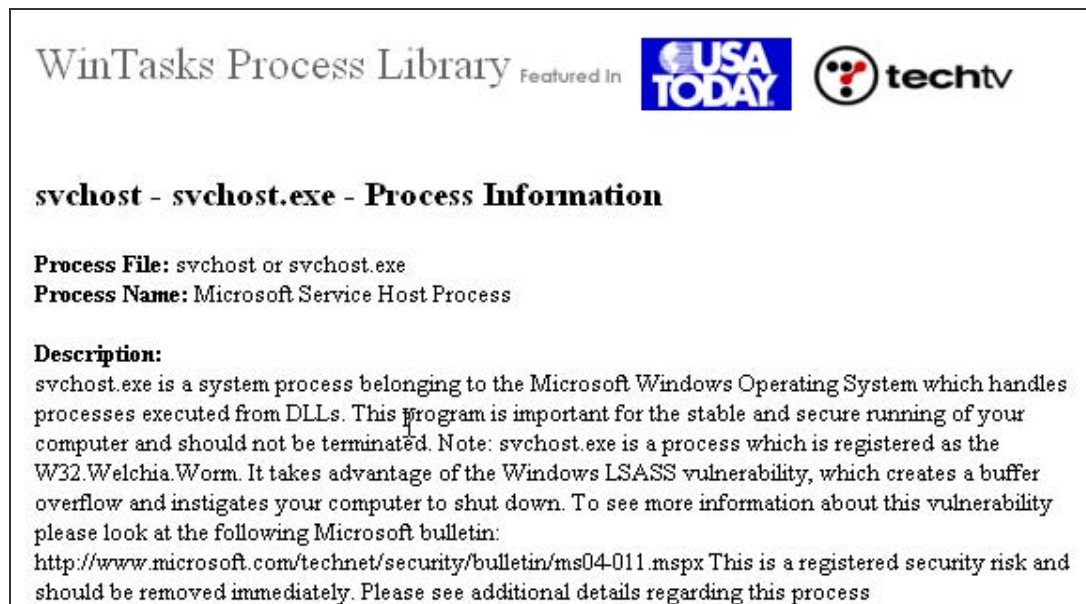


Figure 9: WinTasks Process Description

2.1.6 Process Hashes (National Software Reference Library)

Another valuable online resource is NIST's National Software Reference Library's repository of SHA1 and MD5 hashes for critical system and application files. The NSRL repository includes hashes of non-English software files, operating system files, application software files, images, and graphics found on a typical Windows installation. NSRL's stored repositories of hashes are cryptographic hashes of safe or uncorrupted files. This is very important when wanting to compare your own system's critical file hashes against a known safe set. This online resource provides the cryptographic hashes free of charge; they are downloadable as ISO images.

Once you've burned the images to a CD-ROM, you can unzip the zipped files and get started. You'll find a list and description of the five text files that come with the NIST Operating System ISO. The Operating System ISO is important because it contains safe hashes of executables and DLL files. These are common types of critical files on Windows machines that become corrupted or replaced with compromised ones. NSRL maintains and updates the ISO images periodically (non-English software, operating systems, application software, and images and graphics) as new updates and patches are released.

Operating System ISO:

- *Hashes.txt* – contains hashes for the files so you can check the integrity of the downloaded files
- *Version.txt* – displays the version and date of the downloaded ISO. Keep in mind these ISOs will be updated periodically, so it is important to check the version file for each ISO file because hashes may change with different versions.

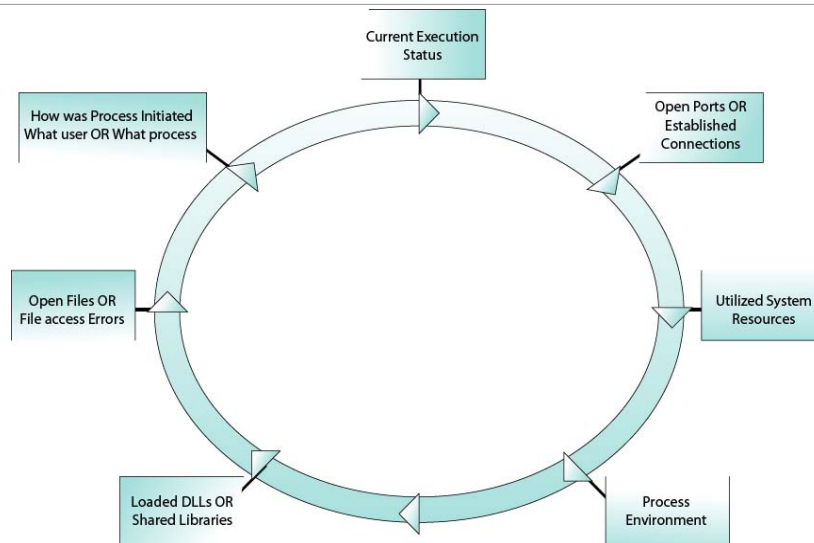
- *NSRLMfg.txt* – displays the Manufacture Code (MfgCode) and Name (MfgName) of the supported manufacturers
- *NSRLProd.txt* – displays the supported “ProductCode,” “ProductName,” “Product Version,” “OpSystemCode,” “MfgCode,” “Language,” and “ApplicationType”
- *NSRLFile.txt* – This is the file that actually has the list of the SHA1 and MD5 hashes that can be used for checking critical associated OS files. The format of each entry in the text file goes as follows:

SHA-1, MD5, CRC32, FileName, FileSize, ProductCode, OpSystemCode, SpecialCode

Here is an example of one entry in the *NSRLFile.txt* text file that contains a list of SHA-1 and MD5 hashes. As you can see, the entry corresponds to the *mshearts.exe* program for a Windows XP machine and has a SHA-1 and MD5 hash for the *mshearts* executable.

```
"001A6C9B8D9471B0A3B4F46302DB951F4D877227","BE1B85306352E0AC901EC08506792B6B","CB76D275","mshearts.exe",126976,1567,"WINXP",""
```

Process Analysis Checklist



2.1.7 Process Analysis Checklist

The procedure of inspecting processes for unexpected behavior or characteristics involves many detailed actions. The abbreviated checklist below contains questions about processes that you may wish to ask yourself and characteristics you may wish to enumerate. In the following paragraphs we are going to demonstrate how to forensically collect some identified items and some extra process characteristics using native commands and third-party tools.

- How was this process initiated?
 - By what user?
 - From what program or other process?
- What is the current execution status of each process?
 - Is it running, stopped, suspended, swapped out, exiting, or in some other unexpected state?
 - Does the process continue to appear among active processes after it should have exited?
 - Is it missing from among the processes you expected to be active?
- In what environment is this process executing?
 - What system settings are in effect for this process?
 - Did the process inherit any environment settings from other processes?
 - How might the current environment settings affect how the process operates and what it can access?
- With what options or input arguments is the process executing? Are these appropriate settings?

- Are the system resources (CPU time, memory usage, etc.) being utilized by each process within expected consumption amounts?
 - Are there any processes that seem to be tying up an unusually large amount of system resources?
 - Are any processes not performing as expected because they don't seem to be getting enough resources?
- What is the relationship between this process and other processes executing on the system? What are the characteristics of the related processes?
- What files have been opened by the processes executing on the system?
 - Are they authorized to have these files open?
 - Have the files been opened with excessive privileges (e.g., opened with read-write capability when there is no reason for the process to write to the file)?
- Have there been any unexpected accesses to sensitive system files or other private data, such as password files?
 - From what process were the accesses made?
 - With which user is that process associated?
- Have there been any unauthorized attempts to access a file? Has the system reported any file access errors?

8 Process Characteristics

1. Process filename
2. Open ports
3. Open files
4. Base priority
5. Start, stop, elapsed times
6. Location (i.e., full path) of process image
7. Survivable processes
8. Loaded DLLs or libraries



1.	ps.exe	ps -aux
2.	fport.exe	netstat -tap
3.	handle.exe	lsuf
4.	pslist.exe	top -n 1
5.	pslist.exe psloglist.exe	ps -auxw
6.	listdlls.exe	lsuf -p PID
7.	autorunsc.exe	checkconfig crontab
8.	listdlls.exe	ldd

2005 Carnegie Mellon University

8



2.1.8 Common Process Characteristics

We have outlined the eight key process characteristics that follow along with the Process Analysis Checklist, as well as a set of tools and native commands a first responder can use to collect those characteristics. Note that this is not an exhaustive list of running process characteristics. But collecting these eight process characteristics for a potential rogue can significantly aid in determining whether that running process is legitimate or not.

2.1.8.1 Process Filenames

The process filename is the filename of the process image that was executed to initiate the currently running process. In most cases you can look at the process filename to determine whether the current running process is a legitimate Windows or Linux process.

2.1.8.1.1 pulist (Windows)

Using the *pulist.exe* command line utility, a Windows Resource kit utility, we can generate a list of currently running processes and the associated filenames to be examined for unexpected process filenames and unusual user identifications.

In the screenshot in Figure 10, the four processes that we identified to be rogue processes judging by the process filenames are *tini.exe*, *klogger.exe*, *svchost1.exe*, and *qfskrtwj.exe*.

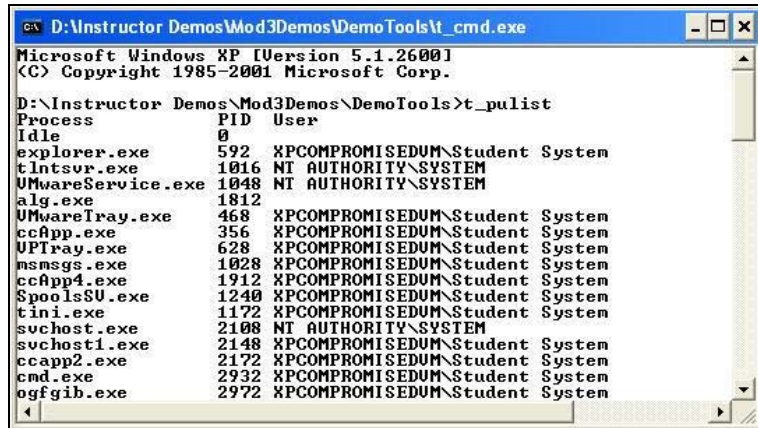


Figure 10: Listing Process Filenames Using pulist

2.1.8.1.2 ps (Linux)

The Linux *ps* command can be used to display the filename of the process image as well as other things outlined in Table 6 and Table 7.

Table 6: A Subset of ps Options

Option	Description
\$ ps -ux	View current processes
\$ ps -U user	View other system users running processes
\$ ps -C program_name	View all occurrences of a program
\$ ps -p4, 8, 2203	View selected processes 4, 8, 2203
\$ ps -efww	View all processes with full command lines

Table 7 describes some output headings for *ps* and *top* output.

Table 7: Output Headings for ps and top

Field	Description
USER	Username of the process owner
PID	Process ID
%CPU	Percentage of the CPU this process is using
%MEM	Percentage of real memory this process is using
VSZ	Virtual size of the process, in kilobytes

Field	Description
RSS	
TT	
STAT	Current Process Status R= Runnable D = In disk wait I = Sleeping (< 20 sec) S = Sleeping (> 20 sec) T = Stopped Z = Zombie Additional Flags L = Some pages are locked in core (for rawio) S = Process is a session leader (head of control terminal) W = Process is swapped out + = Process is in the foreground of its control terminal
START	Time the process was started
TIME	CPU time the process has consumed
COMMAND	Command name and arguments

2.1.8.2 Open Ports

Another critical process characteristic is the number of ports a particular process has open. Processes that have unfamiliar or unnecessary TCP or UDP ports open could indicate that the process is a backdoor or Trojan allowing remote access to the machine.

2.1.8.2.1 *fport* (Windows)

The *fport.exe* command line utility, like the native windows *netstat -anb* command, displays all open TCP/IP and UDP ports and maps them to the owning application as shown in Figure 11.

Fport also maps those ports to running processes with the PID, process name, and path to the process image.

```

D:\Instructor Demos\Mod3Demos\DemoTools>t_cmd.exe
D:\Instructor Demos\Mod3Demos\DemoTools>t_fport
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid  Process          ->  Port  Proto  Path
688   tlnsvr             ->  23    TCP    C:\WINDOWS\System32\tlnsvr.exe
2424  svchost1           ->  80    TCP    c:\windows\system32\svchost1.exe
1448  svchost1           ->  135   TCP
4     System            ->  139   TCP
4     System            ->  445   TCP
1496  System            ->  1028  TCP
2204  tini              ->  7777  TCP    c:\windows\system32\tini.exe
2408  frdpjscep         ->  27374 TCP    C:\WINDOWS\frdpjscep.exe

0     System            ->  123   UDP
2424  svchost1           ->  123   UDP    c:\windows\system32\svchost1.exe
0     System            ->  137   UDP
0     System            ->  138   UDP
688   tlnsvr             ->  445   UDP    C:\WINDOWS\System32\tlnsvr.exe
1448  tlnsvr             ->  500   UDP
4     System            ->  1025  UDP
2204  tini              ->  1026  UDP    c:\windows\system32\tini.exe
4     System            ->  1027  UDP
0     System            ->  1900  UDP
2204  tini              ->  1900  UDP    c:\windows\system32\tini.exe
2408  frdpjscep         ->  2967  UDP    C:\WINDOWS\frdpjscep.exe
1496  frdpjscep         ->  4500  UDP

D:\Instructor Demos\Mod3Demos\DemoTools>_

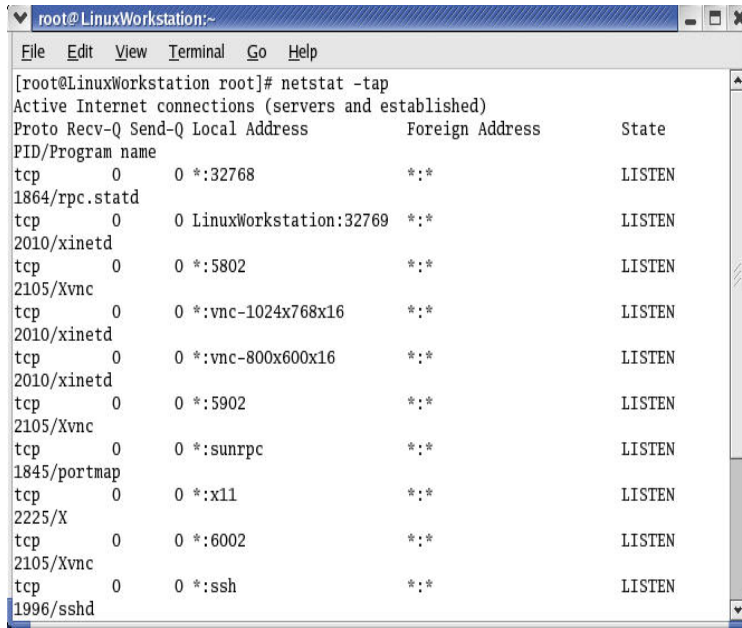
```

Figure 11: Displaying Open Ports Using *fport*

2.1.8.2.2 netstat (Linux)

The Linux *netstat* command can be used to display all TCP/IP and UDP ports that are open in relation to a running process.

Using *netstat* with the *-tap* command line arguments will display all running processes that have a TCP/IP port open, the PID of the process, the port number assignment, the foreign address if connected, and the state of the port (Figure 12).



Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
Active Internet connections (servers and established)					
PID/Program name					
tcp	0	0	*:32768	*:*	LISTEN
1864/rpc.statd					
tcp	0	0	LinuxWorkstation:32769	*:*	LISTEN
2010/xinetd					
tcp	0	0	*:5802	*:*	LISTEN
2105/Xvnc					
tcp	0	0	*:vnc-1024x768x16	*:*	LISTEN
2010/xinetd					
tcp	0	0	*:vnc-800x600x16	*:*	LISTEN
2010/xinetd					
tcp	0	0	*:5902	*:*	LISTEN
2105/Xvnc					
tcp	0	0	*:sunrpc	*:*	LISTEN
1845/portmap					
tcp	0	0	*:x11	*:*	LISTEN
2225/X					
tcp	0	0	*:6002	*:*	LISTEN
2105/Xvnc					
tcp	0	0	*:ssh	*:*	LISTEN
1996/sshd					

Figure 12: Displaying Open Ports Using netstat

Potential Rogue Processes

Filename	Port	PID	Location
cuoikqkxvs.exe	27374	2288	C:\WINDOWS
svchost1.exe	80	312	C:\..\System32
notepad.exe:alds.exe	7777	988	C:\..\System32
spoolsv.exe	NA	284	C:\..\System32\1024

2005 Carnegie Mellon University

9



2.1.8.3 Open Files

Open files associated with an executing process should not be overlooked. Often, rogue processes such as a key logger or network sniffer will have an associated open file to capture their collected information. One quick way to determine whether a potential rogue process has any current files open is to use the *handle.exe* utility for Windows and the native *lsof* command for Linux.

2.1.8.3.1 *handle* (Windows)

The *handle.exe* utility displays information about open handles for processes running on the system. You can use it to view the programs that have a file open or to view the object types and names of all the handles of a program.

The screenshot in Figure 13 demonstrates how we can use the *handle.exe* utility to look at all open handles for the potential rogue process *svchost1*.

```

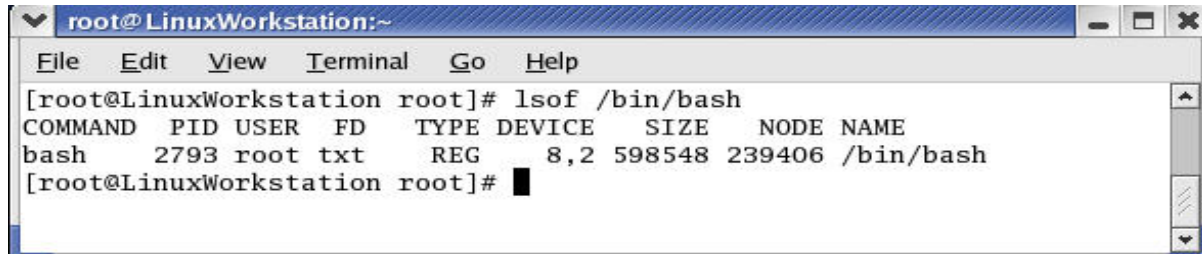
D:\Instructor Demos\Mod3Demos\DemoTools\cmd.exe
D:\Instructor Demos\Mod3Demos\DemoTools>t_handle -a -p svchost1
Handle v2.2
Copyright (C) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

-----
svchost1.exe pid: 2148 XPROMPTEDUM\Student System
c: File C:\Documents and Settings\Student System
784: Port
78c: IoCompletion
790: IoCompletion
794: Event
798: IoCompletion
79c: File \Device\Tcp
7a0: File \Device\Ndis\Endpoint
7a4: Desktop \Default
7a8: WindowStation \Windows\WindowStations\WinSta0
7ac: Section
7b0: Event
7b4: Event
7b8: File \Device\KsecDD
7bc: Thread svchost1.exe(2148): 2152
7c0: Event
  
```

Figure 13: Viewing Handles Using *handle*

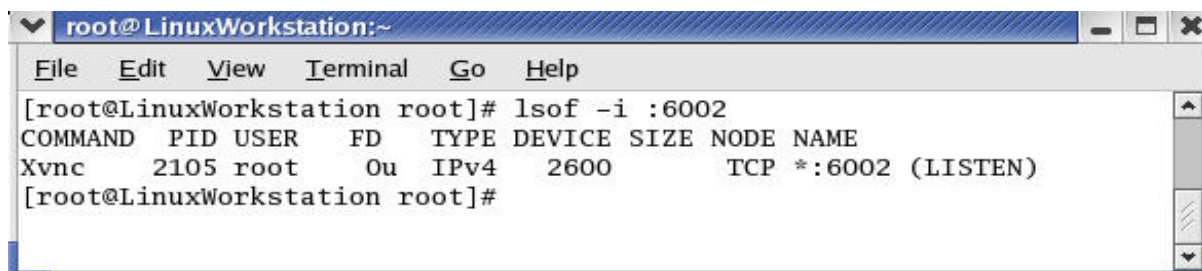
2.1.8.3.2 lsof (Linux)

The native *lsof* command without any command line arguments will display all open files belonging to all currently running processes. The three screenshots (Figure 14, Figure 15, and Figure 16) demonstrate *lsof*'s command versatility.



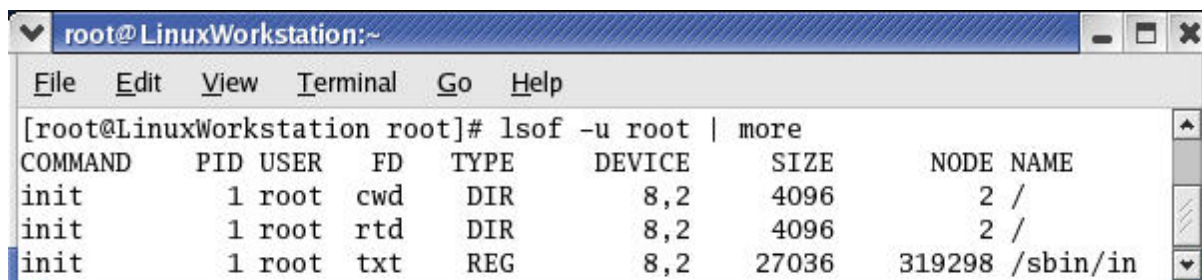
```
root@LinuxWorkstation:~  
File Edit View Terminal Go Help  
[root@LinuxWorkstation root]# lsof /bin/bash  
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME  
bash 2793 root txt REG 8,2 598548 239406 /bin/bash  
[root@LinuxWorkstation root]#
```

Figure 14: Displaying Which Process Has Port 6002 Open



```
root@LinuxWorkstation:~  
File Edit View Terminal Go Help  
[root@LinuxWorkstation root]# lsof -i :6002  
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME  
Xvnc 2105 root Ou IPv4 2600 TCP *:6002 (LISTEN)  
[root@LinuxWorkstation root]#
```

Figure 15: Displaying Who Has the Bash Shell Open



```
root@LinuxWorkstation:~  
File Edit View Terminal Go Help  
[root@LinuxWorkstation root]# lsof -u root | more  
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME  
init 1 root cwd DIR 8,2 4096 2 /  
init 1 root rtd DIR 8,2 4096 2 /  
init 1 root txt REG 8,2 27036 319298 /sbin/in
```

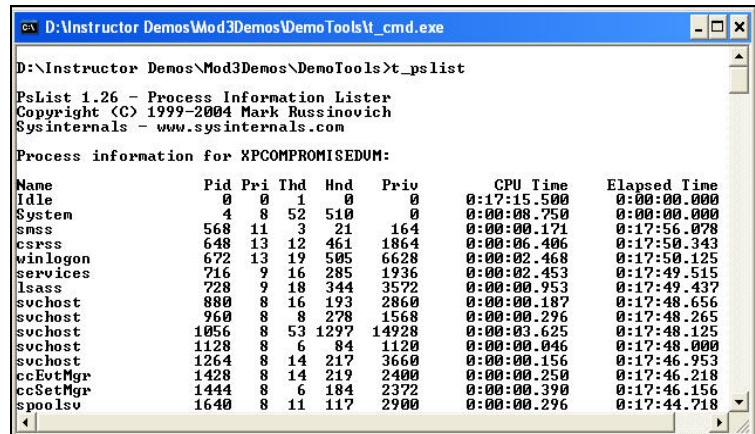
Figure 16: Displaying All the Currently Open Files by the User Root

2.1.8.4 Base Priority

When a process is initially executed for both Windows and Linux it is assigned a base priority value. That value determines what priority it has over other processes in regard to the computer resources it is assigned and consumes, such as memory and CPU time. When looking at a potential rogue or runaway process you may want to check the assigned priority value.

2.1.8.4.1 pslist (Windows)

Using Sysinternals's *pslist.exe* utility we can enumerate the priority levels for each current running process by looking at the "Pri" column. The screenshot in Figure 17 demonstrates how to display priority values for running processes.



```
D:\Instructor Demos\Mod3Demos\DemoTools>t_pslist

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

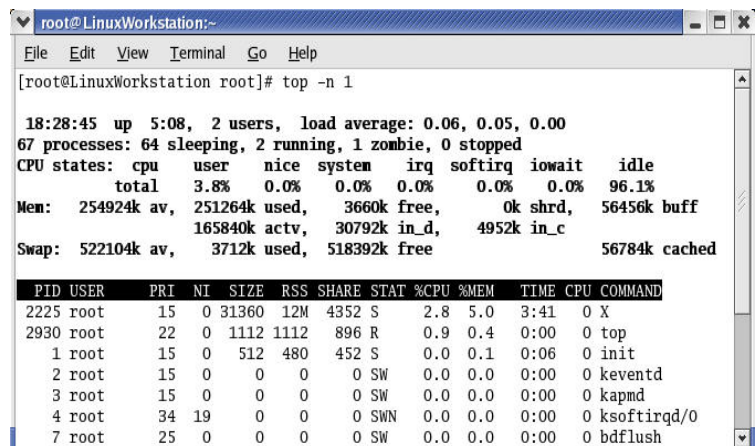
Process information for XPCompromisedum:

Name           Pid Pri Thd  Hnd  Priv    CPU Time    Elapsed Time
-----
Idle            0  0   1    0    0      0:17:15.500    0:00:00.000
System          4  8  52   510  0      0:00:08.750    0:00:00.000
smss            568 11  3    21   164    0:00:00.171    0:17:56.078
csrss           648 13  12   461  1864    0:00:06.406    0:17:50.343
winlogon        672 13  19   505  6628    0:00:02.468    0:17:50.125
services        716  9  16   285  1936    0:00:02.453    0:17:49.515
lsass           728  9  18   344  3572    0:00:00.953    0:17:49.437
svchost         880  8  16   193  2860    0:00:00.187    0:17:48.656
svchost         960  8  8    278  1568    0:00:00.296    0:17:48.265
svchost        1056  8  53  1297  14928    0:00:03.625    0:17:48.125
svchost        1128  8  6    84   1120    0:00:00.046    0:17:48.000
svchost        1264  8  14   217  3660    0:00:00.156    0:17:46.953
ccEvtMgr        1428  8  14   219  2400    0:00:00.250    0:17:46.218
ccSetMgr        1444  8  6    184  2372    0:00:00.390    0:17:46.156
spoolsv        1640  8  11   117  2900    0:00:00.296    0:17:44.718
```

Figure 17: Listing Priority Levels Using pslist

2.1.8.4.2 top (Linux)

Using the native Linux *top* command we can enumerate the priority levels for each current running process. Linux processes will generally have a priority value between -20 and 19, where the value of -20 is the highest and 19 is the lowest priority value (Figure 18).



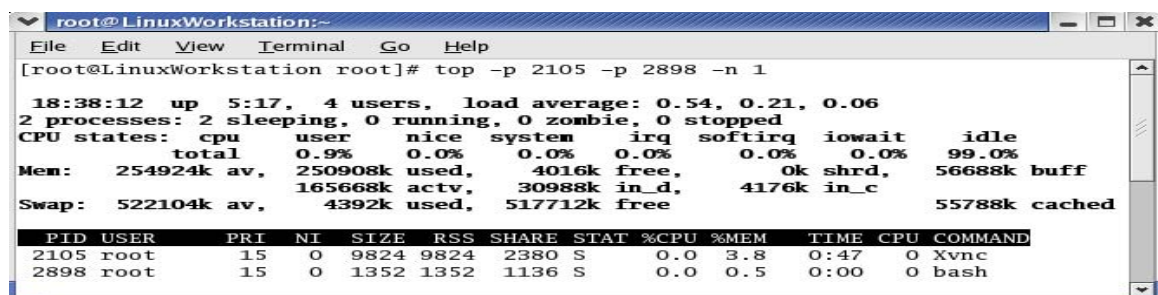
```
root@LinuxWorkstation:~
File Edit View Terminal Go Help
[root@LinuxWorkstation root]# top -n 1

18:28:45 up 5:08, 2 users, load average: 0.06, 0.05, 0.00
67 processes: 64 sleeping, 2 running, 1 zombie, 0 stopped
CPU states:  cpu  user  nice  system  irq  softirq  iowait  idle
              total 3.8%  0.0%  0.0%  0.0%  0.0%  0.0%  96.1%
Mem: 254924k av, 251264k used, 3660k free, 0k shrd, 56456k buff
Swap: 522104k av, 3712k used, 518392k free, 56784k cached

  PID USER   PRI  NI  SIZE  RSS SHARE STAT  %CPU %MEM    TIME CPU  COMMAND
  2225 root    15   0 31360 12M 4352 S    2.8  5.0   3:41  0 X
  2930 root    22   0 1112 1112  896 R    0.9  0.4   0:00  0 top
    1 root    15   0  512  480  452 S    0.0  0.1   0:06  0 init
    2 root    15   0   0   0   0 SW    0.0  0.0   0:00  0 keventd
    3 root    15   0   0   0   0 SW    0.0  0.0   0:00  0 kpmnd
    4 root    34  19   0   0   0 SWN   0.0  0.0   0:00  0 ksoftirqd/0
    7 root    25   0   0   0   0 SW    0.0  0.0   0:00  0 bdflush
```

Figure 18: Listing Priority Levels Using top

Since the *top* command only displays the top current processes, if we need to enumerate the process priority value for a particular process we can use the *top* command with the command line arguments *-p* PID *-n* 1, as shown in Figure 19.



```
root@LinuxWorkstation:~
File Edit View Terminal Go Help
[root@LinuxWorkstation root]# top -p 2105 -p 2898 -n 1

18:38:12 up 5:17, 4 users, load average: 0.54, 0.21, 0.06
2 processes: 2 sleeping, 0 running, 0 zombie, 0 stopped
CPU states:  cpu  user  nice  system  irq  softirq  iowait  idle
              total 0.9%  0.0%  0.0%  0.0%  0.0%  0.0%  99.0%
Mem: 254924k av, 250908k used, 4016k free, 0k shrd, 56688k buff
Swap: 522104k av, 4392k used, 517712k free, 55788k cached

  PID USER   PRI  NI  SIZE  RSS SHARE STAT  %CPU %MEM    TIME CPU  COMMAND
  2105 root    15   0 9824 9824 2380 S    0.0  3.8   0:47  0 Xvnc
  2898 root    15   0 1352 1352 1136 S    0.0  0.5   0:00  0 bash
```

Figure 19: Displaying the Priority Level for a Specific Process

2.1.8.5 Process Times and Terminated Processes

2.1.8.5.1 Process Start Time

The process start time is the point in time when the process started executing. An interesting characteristic of rogue processes is that they generally will have a start time that is a few seconds or few minutes later than all other legitimate running processes. To discover processes that may have started later in time after the boot cycle you can use the *psuptime.exe* and *pslist.exe* command line tools. These tools will tell you when a process first started.

psuptime (Windows)

To calculate the initial start time of a process, we first have to collect the uptime or how long the system has been running. We can do this using the *psuptime.exe* utility.

Once we have the uptime of the system, the next step is to enumerate all of the elapsed times for the current set of running processes using *pslist.exe*.

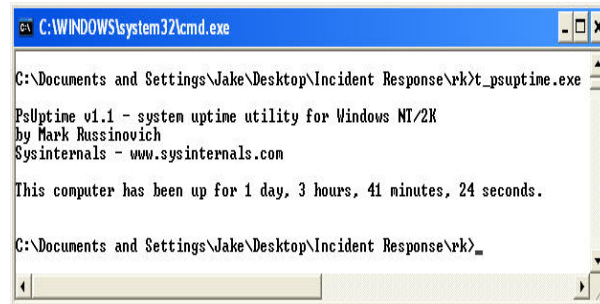


Figure 20: Checking Uptime Using *psuptime*

2.1.8.5.2 Process Elapsed Time

pslist (Windows)

Once we have the uptime of the system and the elapsed time for a particular process we can simply subtract the (Uptime – Elapsed time) to calculate Start Time for any given process.

For example, the Uptime of the system was = 27h:41m:24s and the Elapsed Time of the *Svchost.exe* process was = 3h:34s:41s; therefore, the Start Time for the *Svchost.exe* process was 24h:6m:43s (12:06:43 a.m.).

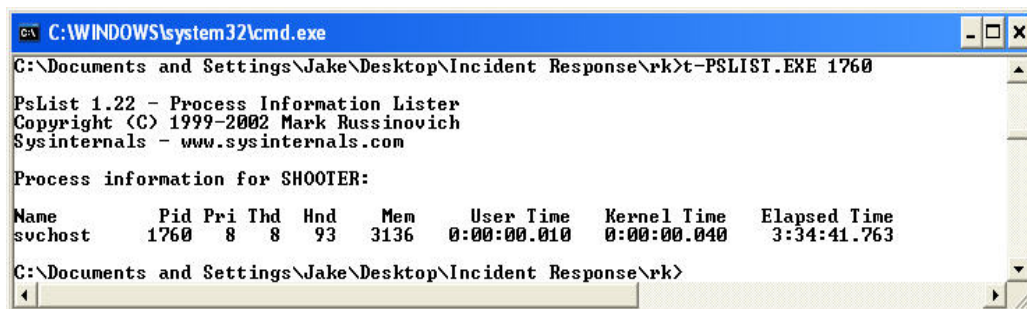


Figure 21: Checking Elapsed Time for a Process Using *pslist*

Note that, by just looking at the *pslist.exe* output, more specifically the elapsed times, you can determine which processes started after the boot process.

2.1.8.5.3 Terminated Processes

Suspended or prematurely terminated processes can be indicators of abnormal system behavior. Often in computer security situations, a critical system process will be suspended or terminated. For example, an antivirus or other critical process may be terminated in an attempt to prevent the host system's security mechanisms from checking for running malware on the system. Therefore, terminated processes should be collected and identified.

One method to check for terminated processes in Windows, assuming proper auditing is enabled, is reviewing the event logs.

The screenshot in Figure 22 displays a system event log using the Windows *Event Viewer*. A terminated process in Windows exhibits an Event ID of 7034.

Knowing what Event ID to search for, we can use a command line utility called *psloglist.exe* developed by Sysinternals to collect and parse through the entire system event logs looking for only event logs that exhibit the Event ID of 7034.

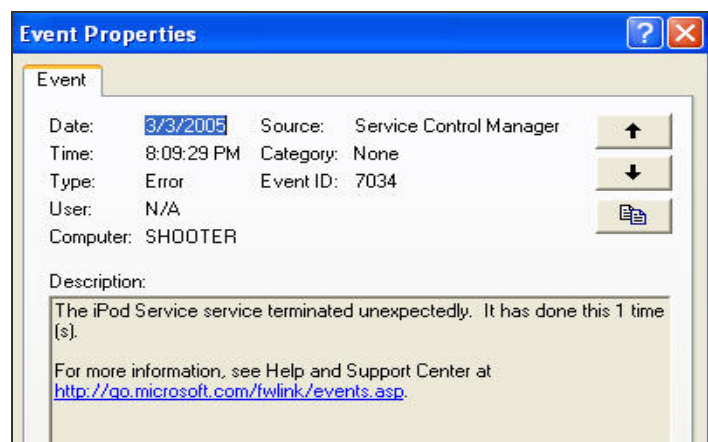


Figure 22: Windows Event Log

Figure 23 demonstrates how to use the *psloglist.exe* utility with the *-i* command line argument to search for all event logs that have the Event ID of 7034.

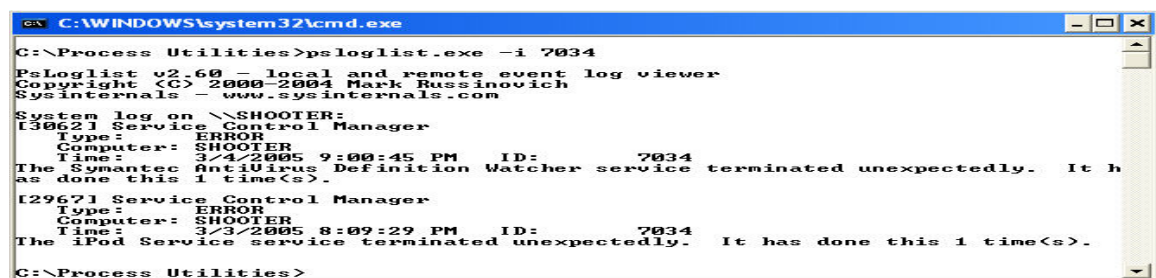


Figure 23: psloglist Command

2.1.8.5.4 Process Terminated Time

As stated, we can determine which processes have been terminated by using Sysinternals' *psloglist.exe*. Using the same approach, we can see that the output of the collected event log clearly states at what time the process was terminated.

2.1.8.6 Location of Process Image

The location of the process image can give you further insight into whether the process is a legitimate or rogue process. For example, if a running process's image is located in the *Startup* folder or another anomalous file location, there is a good chance that the currently running process is not legitimate. We can quickly identify the location of a process image by using the following utilities for Windows and Linux.

2.1.8.6.1 ListDLLs (Windows)

Using Sysinternals' *ListDLLs* utility we can determine the command line used to execute the process and the location of the process image.



```
C:\ D:\WINDOWS\system32\cmd.exe

D:\WTools>listdlls.exe firefox

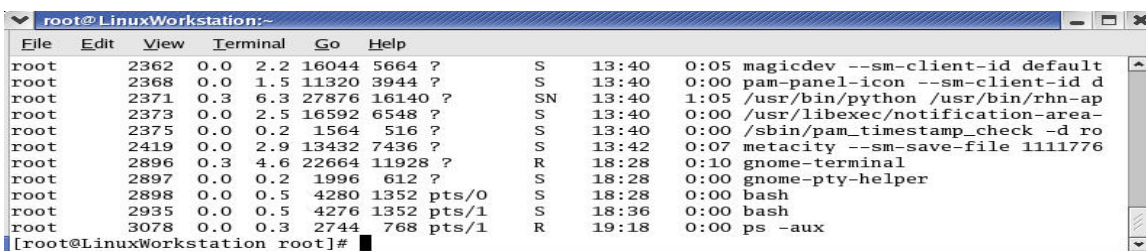
ListDLLs U2.23 - DLL lister for Win9x/NT
Copyright (C) 1997-2000 Mark Russinovich
http://www.sysinternals.com

-----
firefox.exe pid: 2928
Command line: "D:\Program Files\Mozilla Firefox\firefox.exe"
```

Figure 24: Locating a Process Image Using ListDLLs

2.1.8.6.2 ps and lsof (Linux)

Using the native Linux *ps* command with the *-aux* command line arguments, we can determine the command line used to execute the process and the location of the process image.



```
root@LinuxWorkstation:~# ps -aux
```

File	Edit	View	Terminal	Go	Help
root	2362	0.0	2.2	16044	5664 ? S 13:40 0:05 magicdev --sm-client-id default
root	2368	0.0	1.5	11320	3944 ? S 13:40 0:00 pam-panel-icon --sm-client-id d
root	2371	0.3	6.3	27876	16140 ? SN 13:40 1:05 /usr/bin/python /usr/bin/rhn-ap
root	2373	0.0	2.5	16592	6548 ? S 13:40 0:00 /usr/libexec/notification-area-
root	2375	0.0	0.2	1564	516 ? S 13:40 0:00 /sbin/pam_timestamp_check -d ro
root	2419	0.0	2.9	13432	7436 ? S 13:42 0:07 metacity --sm-save-file 1111776
root	2896	0.3	4.6	22664	11928 ? R 18:28 0:10 gnome-terminal
root	2897	0.0	0.2	1996	612 ? S 18:28 0:00 gnome-pty-helper
root	2898	0.0	0.5	4280	1352 pts/0 S 18:28 0:00 bash
root	2935	0.0	0.5	4276	1352 pts/1 S 18:36 0:00 bash
root	3078	0.0	0.3	2744	768 pts/1 R 19:18 0:00 ps -aux

```
[root@LinuxWorkstation root]#
```

Figure 25: Locating a Process Image Using ps

Using the *lsof* command with the *-p* PID command line argument, we can enumerate the location of the process image defined by process ID or PID and also any other files open by the defined process.

```

[root@LinuxWorkstation root]# lsof -p 2896
COMMAND  PID USER  FD   TYPE    DEVICE  SIZE  NODE  NAME
gnome-ter 2896 root   cwd   DIR      8,2    4096 286849 /root
gnome-ter 2896 root   rtd   DIR      8,2    4096 2      /
gnome-ter 2896 root   txt   REG     8,2   307400 417706 /usr/bin/gnome-terminal
gnome-ter 2896 root   mem   REG     8,2   618112 304947 /usr/share/fonts/bitmap-fonts/9x18.pcf
gnome-ter 2896 root   mem   REG     8,2    96263 272412 /usr/share/fonts/default/Type1/n0

```

Figure 26: Locating a Process Image by PID

2.1.8.7 Survivable Processes

Survivable processes can be defined as processes that will re-execute after the machine has been shut down and then restarted. Often when malware infects a machine or if an attacker compromises the system, the stored malicious code will be located in startup locations on the system. Also, registry keys and values may have been modified so that the malware processes will be started up again upon system reboot. Question certain file locations in the system.

- startup folders, added registry key values and scripts
Check to see if unauthorized applications are starting upon reboot. There are a number of different methods an intruder can use to start a backdoor program, so be sure to check the startup folders, startup scripts, and even registry key values.
- invalid services
Check for invalid services. Some backdoor programs will install themselves as services so they are started when the system boots up.
- scheduled tasks
Check for scheduled tasks and Crontab files.

2.1.8.7.1 Startup Folders, Registry Keys and Values (Windows)

Here are some common Windows startup locations/folders and registry keys to consider.

- Check all items in the *C:\Documents and Settings\All Users\Start Menu\Programs\Startup* folder. Note that there are two startup folders, one for the local user and one for all other users on the system. When a user logs on, all of the applications in both the local user's and in the All Users startup folders are started. Because of this, it is important to check both of the startup folders.
- Check the registry for added keys and key values. The most common locations for applications to start through the registry are the following:
 - *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\KnownDLLs*
 - *HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\KnownDLLs*
 - *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\Run*
 - *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\RunOnce*

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\RunOnceEx
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows ("run=" line)
- HKEY_CURRENT_USER\Software\Microsoft\Windows\Current Version\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\Current Version\RunOnce
- HKEY_CURRENT_USER\Software\Microsoft\Windows\Current Version\RunOnceEx
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
- HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows ("run=" value)

autorunsc (Windows)

The *autorunsc.exe* utility developed by Sysinternals allows you to collect all of the following information regarding survivable processes and services:

- startup applications and their location
- registry key values
- startup services and their location

Figure 27 is a demonstration of using the *autorunsc.exe* utility to collect information about processes that will be started upon reboot. For example, look at the executables located in the *Startup* folder.

```
D:\Instructor Demos\Mod3Demos\DemoTools>t_autorunsc.exe /?

Autorunsc v5.01 - Autostart program viewer
Copyright (C) 2002-2004 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com

Autorunsc shows programs configured to autostart during boot.

Usage: autorunsc [-a] [-c] [-d] [-e] [-m] [-s] [-w]
    -a      Include empty locations.
    -c      Print output as CSV.
    -d      Show Appinit DLLs.
    -e      Show Explorer addons.
    -m      Hide signed Microsoft entries.
    -s      Show autostart services.
    -w      Winlogon entries.

D:\Instructor Demos\Mod3Demos\DemoTools>t_autorunsc.exe -c !more_
C:\Documents and Settings\All Users\Start Menu\Programs\Startup
BackdoorTelnet.bat
c:\documents and settings\all users\start menu\programs\startup\backdoortelnet.bat
Bginfo.exe
BGInfo - Wallpaper text configurator
(Not verified) Sysinternals
c:\documents and settings\all users\start menu\programs\startup\bginfo.exe
ccApp4.exe
c:\documents and settings\all users\start menu\programs\startup\ccapp4.exe
logger.bat
c:\documents and settings\all users\start menu\programs\startup\logger.bat
Microsoft Office.lnk
Microsoft Office XP component
Microsoft Corporation
c:\program files\microsoft office\office10\osa.exe
nccconnect.vbs
c:\documents and settings\all users\start menu\programs\startup\nccconnect.vbs
rogueprocess.bat
c:\documents and settings\all users\start menu\programs\startup\rogueprocess.bat
```

Figure 27: autorunsc.exe Command

Startup Locations and Scripts (Linux)

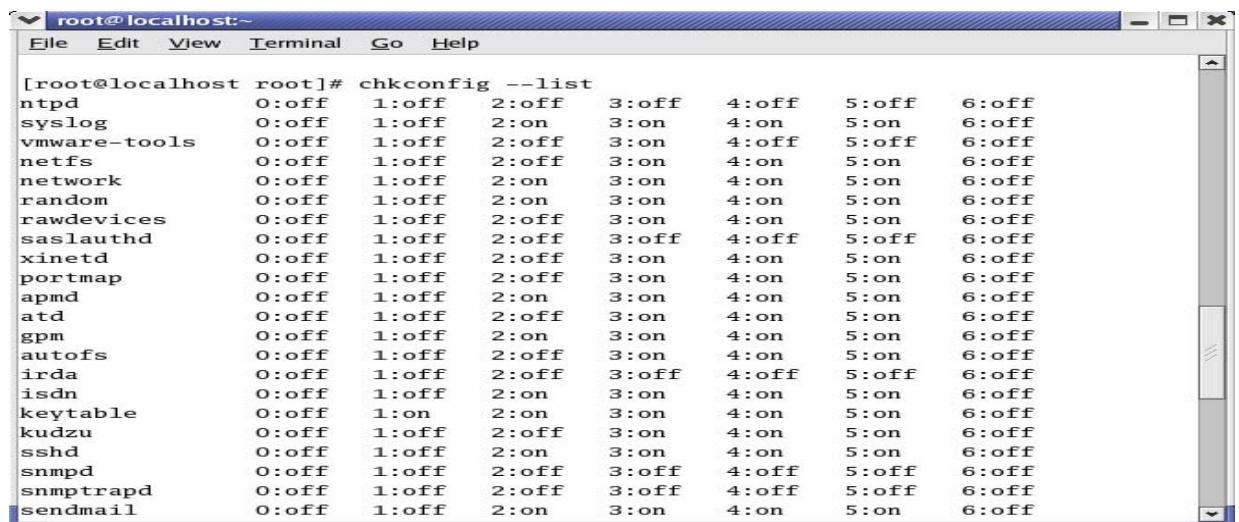
For a Linux system, certain files should be examined to determine whether there are malicious scripts within these files. Often an attacker will place a shell script in one of the following files so that it gets executed every time the machine is rebooted. This is not an exhaustive list by any means, but includes common file locations to consider and check.

- `$ etc/rc.local`
- `$ etc/initab`
- `$ etc/rc.sysinit`

2.1.8.7.2 Invalid Services

Checkconfig (Linux)

The `chkconfig --list` command displays a list of services that will be run at the five different runlevels. This information may help you identify a rogue or malware application that is set to run as a service and at one of the five runlevels.



```
[root@localhost root]# chkconfig --list
ntpd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
syslog        0:off  1:off  2:on   3:on   4:on   5:on   6:off
vmware-tools  0:off  1:off  2:off  3:on   4:off  5:off  6:off
netfs         0:off  1:off  2:off  3:on   4:on   5:on   6:off
network       0:off  1:off  2:on   3:on   4:on   5:on   6:off
random        0:off  1:off  2:on   3:on   4:on   5:on   6:off
rawdevices    0:off  1:off  2:off  3:on   4:on   5:on   6:off
saslauthd     0:off  1:off  2:off  3:off  4:off  5:off  6:off
xinetd        0:off  1:off  2:off  3:on   4:on   5:on   6:off
portmap       0:off  1:off  2:off  3:on   4:on   5:on   6:off
apmd          0:off  1:off  2:on   3:on   4:on   5:on   6:off
atd           0:off  1:off  2:off  3:on   4:on   5:on   6:off
gpm           0:off  1:off  2:on   3:on   4:on   5:on   6:off
autofs        0:off  1:off  2:off  3:on   4:on   5:on   6:off
irda          0:off  1:off  2:off  3:off  4:off  5:off  6:off
isdn          0:off  1:off  2:on   3:on   4:on   5:on   6:off
keytable      0:off  1:on   2:on   3:on   4:on   5:on   6:off
kudzu         0:off  1:off  2:off  3:on   4:on   5:on   6:off
sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off
snmpd         0:off  1:off  2:off  3:off  4:off  5:off  6:off
snmptrapd     0:off  1:off  2:off  3:off  4:off  5:off  6:off
sendmail      0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

Figure 28: The `chkconfig -list` Command

Corresponding to the five levels,

chkconfig has five distinct functions: adding new services for management, removing services from management, listing the current startup information for services, changing the startup information for services, and checking the startup state of a particular service.

When *chkconfig* is run without any options, it displays usage information. If only a service name is given, it checks to see if the service is configured to be started in the current runlevel. If it is, *chkconfig* returns true; otherwise it returns false.

The --level option may be used to have chkconfig query an alternative runlevel rather than the current one [Haas 04].

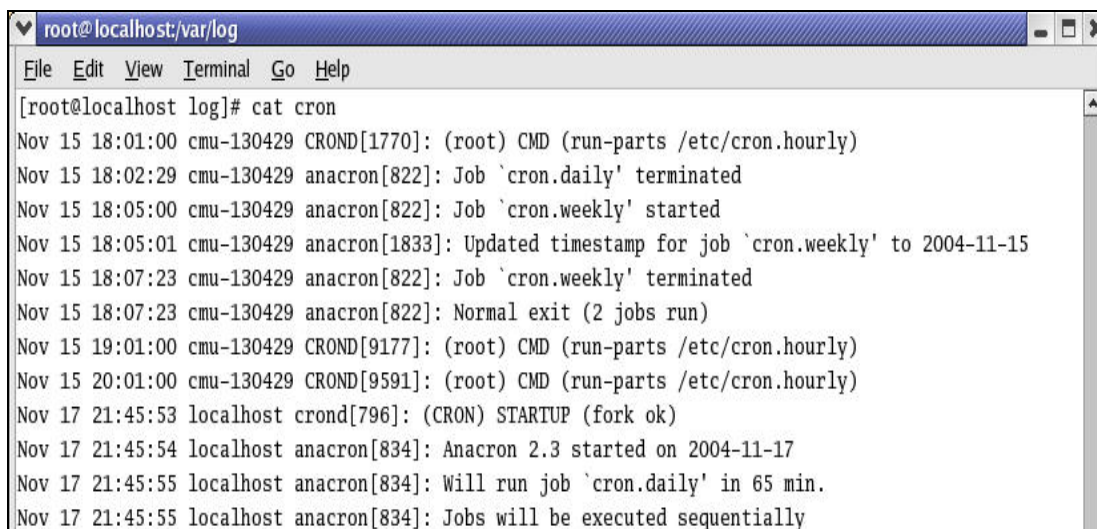
2.1.8.7.3 Scheduled Tasks

at (Windows)

For collecting information about scheduled tasks on a Windows machine, use the native *at.exe* command. The *at.exe* command will display currently scheduled tasks. Scheduled tasks should not be overlooked because an attacker could essentially schedule a certain file, executable, or script to be run on a certain day or time of day that could cause malicious system behavior.

Cron Logs (Linux)

In addition to startup services, you should collect the currently scheduled tasks. Attackers often schedule a malicious file to execute periodically so that the malware remains existent. The cron feature allows system administrators to schedule programs for future execution. All executed cron jobs are logged, usually in the */var/cron/log* or in the default logging directory in a file called *cron*.

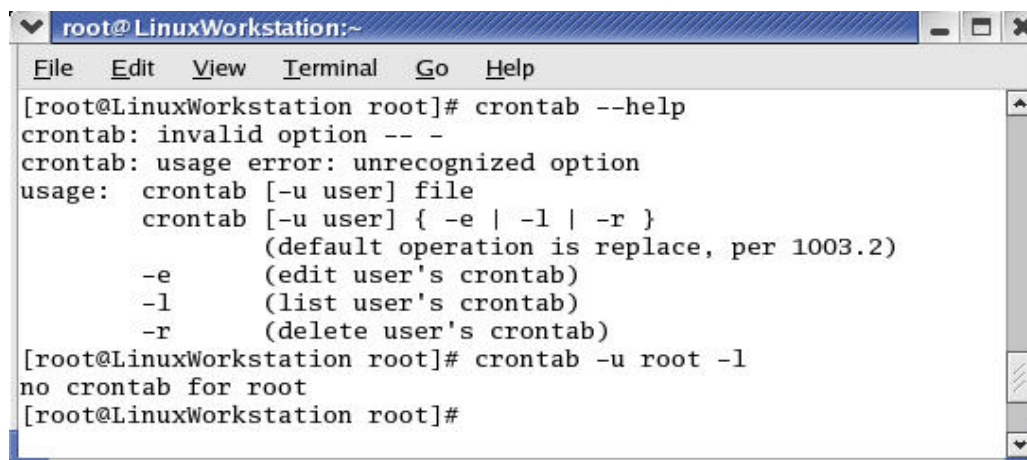


```
root@localhost:/var/log
File Edit View Terminal Go Help
[root@localhost log]# cat cron
Nov 15 18:01:00 cmu-130429 CROND[1770]: (root) CMD (run-parts /etc/cron.hourly)
Nov 15 18:02:29 cmu-130429 anacron[822]: Job `cron.daily' terminated
Nov 15 18:05:00 cmu-130429 anacron[822]: Job `cron.weekly' started
Nov 15 18:05:01 cmu-130429 anacron[1833]: Updated timestamp for job `cron.weekly' to 2004-11-15
Nov 15 18:07:23 cmu-130429 anacron[822]: Job `cron.weekly' terminated
Nov 15 18:07:23 cmu-130429 anacron[822]: Normal exit (2 jobs run)
Nov 15 19:01:00 cmu-130429 CROND[9177]: (root) CMD (run-parts /etc/cron.hourly)
Nov 15 20:01:00 cmu-130429 CROND[9591]: (root) CMD (run-parts /etc/cron.hourly)
Nov 17 21:45:53 localhost crond[796]: (CRON) STARTUP (fork ok)
Nov 17 21:45:54 localhost anacron[834]: Anacron 2.3 started on 2004-11-17
Nov 17 21:45:55 localhost anacron[834]: Will run job `cron.daily' in 65 min.
Nov 17 21:45:55 localhost anacron[834]: Jobs will be executed sequentially
```

Figure 29: A Cron Log

Crontab (Linux)

Using the *crontab* command, we can collect the currently created cron jobs for each user of the system.

A screenshot of a terminal window titled "root@LinuxWorkstation:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal shows the following commands and output:

```
[root@LinuxWorkstation root]# crontab --help
crontab: invalid option -- -
crontab: usage error: unrecognized option
usage: crontab [-u user] file
       crontab [-u user] { -e | -l | -r }
              (default operation is replace, per 1003.2)
       -e      (edit user's crontab)
       -l      (list user's crontab)
       -r      (delete user's crontab)
[root@LinuxWorkstation root]# crontab -u root -l
no crontab for root
[root@LinuxWorkstation root]#
```

Figure 30: The Crontab Command

3 Process Forensic Tasks

1. Check and verify a process's loaded DLLs (Dynamically Linked Libraries)
2. Check and verify a process image
 - Hash and compare the spoolsv.exe binary
3. Process string search and analysis on
 - *svchost1.exe*
 - *spoolsv.exe*
 - *notepad.exe:alds.exe*

Utilities used:

- *strings.exe*
- *grep.exe*
- *md5sum.exe* or *md5deep.exe*
- *sfind.exe*

2005 Carnegie Mellon University

10



2.1.8.8 Process Forensic Tasks

In the following paragraphs we are going to step through a few process forensics tasks that involve diving deeper into investigating a running process's binary. In the previous sections we demonstrated how to forensically collect eight process characteristics using either native commands or third-party utilities. Now we are going to step through three forensic tasks that will be of importance to a first responder. By performing these three forensic tasks, a first responder can gain further insight into a process's binary function and use.

2.1.8.8.1 Check and Verify a Process's Loaded DLLs

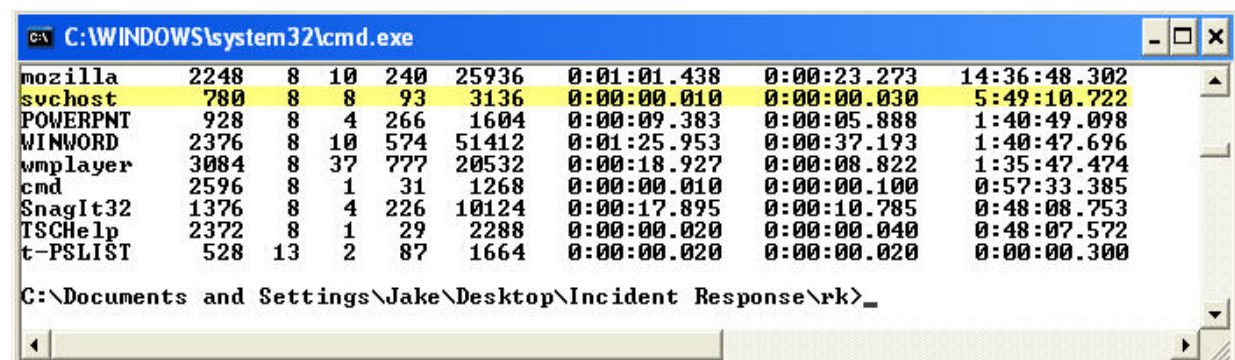
Often running processes will be utilizing one or more dynamically linked libraries (DLLs) for Windows or shared libraries in Linux. Dynamically linked and shared libraries are created so they can be used by many different applications and processes. Having these shared libraries drastically reduces the size of the actual executable or process binary and improves system efficiency.

The main reason why DLLs used by a process should not be overlooked is that malware has been known to replace critical system DLLs with malicious ones or add new malicious DLLs to the system in an attempt to cause undesired system behavior once executed. The majority of root kits use DLL injection to infect systems. For these reasons, we will demonstrate how to check and verify the loaded DLLs for a particular process against a known safe set.

To verify a process's required DLLs involves three steps: (1) identify a process to check, (2) identify the DLLs used by the process, and (3) check to see if the DLLs have been corrupted by hashing them and comparing them against a known safe set.

Step 1: Identify the Process

Identify the potential rogue process by using a process enumeration utility such as *pslist.exe*. The screenshot in Figure 31 displays the highlighted process used for this forensic task of checking and verifying its loaded DLLs.



Process Name	PID	PPID	Session ID	Session Name	Timestamp 1	Timestamp 2	Timestamp 3	
mozilla	2248	8	10	240	25936	0:01:01.438	0:00:23.273	14:36:48.302
svchost	780	8	8	93	3136	0:00:00.010	0:00:00.030	5:49:10.722
POWERPNT	928	8	4	266	1604	0:00:09.383	0:00:05.888	1:40:49.098
WINWORD	2376	8	10	574	51412	0:01:25.953	0:00:37.193	1:40:47.696
wmplayer	3084	8	37	777	20532	0:00:18.927	0:00:08.822	1:35:47.474
cmd	2596	8	1	31	1268	0:00:00.010	0:00:00.100	0:57:33.385
SnagIt32	1376	8	4	226	10124	0:00:17.895	0:00:10.785	0:48:08.753
TSCHe1p	2372	8	1	29	2288	0:00:00.020	0:00:00.040	0:48:07.572
t-PSLIST	528	13	2	87	1664	0:00:00.020	0:00:00.020	0:00:00.300

C:\Documents and Settings\Jake\Desktop\Incident Response\rk>_

Figure 31: The *svchost.exe* 780 Process

Step 2: Identify the DLLs Used by the Process

The second step is to identify all required DLLs for the identified process. To identify all DLLs required by the *svchost.exe* process, use the *listdlls.exe* utility with the command line argument *-p 780*. By using the *-p* command line argument, we are instructing the utility to display loaded DLLs only for the designated process (i.e., PID 780).

Figure 32 displays all of the DLLs required for the *svchost.exe* or PID 780 process.

```

C:\WINDOWS\system32\cmd.exe
svchost.exe pid: 780
Command line: C:\WINDOWS\System32\svchost.exe -k HTTPFilter

Base      Size      Version   Path
0x01000000 0x6000    5.01.2600.2180 C:\WINDOWS\System32\svchost.exe
0x7c900000 0xb00000 5.01.2600.2180 C:\WINDOWS\system32\ntdll.dll
0x7c800000 0xf40000 5.01.2600.2180 C:\WINDOWS\system32\kernel32.dll
0x77d00000 0x9b0000 5.01.2600.2180 C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000 0x910000 5.01.2600.2180 C:\WINDOWS\system32\RPCRT4.dll
0x5cb70000 0x260000 5.01.2600.2180 C:\WINDOWS\System32\ShimEng.dll
0x6f880000 0x1ca000 5.01.2600.2180 C:\WINDOWS\AppPatch\AcGenral.DLL
0x77d40000 0x900000 5.01.2600.2180 C:\WINDOWS\system32\USER32.dll
0x77f10000 0x460000 5.01.2600.2180 C:\WINDOWS\system32\GDI32.dll
0x76b40000 0x2d0000 5.01.2600.2180 C:\WINDOWS\System32\WINMM.dll
0x774e0000 0x13d000 5.01.2600.2595 C:\WINDOWS\system32\ole32.dll
0x77c10000 0x580000 7.00.2600.2180 C:\WINDOWS\system32\msvcrt.dll
0x77120000 0x8c0000 5.01.2600.2180 C:\WINDOWS\system32\OLEAUT32.dll
0x77be0000 0x150000 5.01.2600.2180 C:\WINDOWS\System32\MSACM32.dll
0x77c00000 0x800000 5.01.2600.2180 C:\WINDOWS\system32\USER32.dll
0x77c90000 0x814000 6.00.2900.2578 C:\WINDOWS\system32\SHELL32.dll
0x77f60000 0x760000 6.00.2900.2573 C:\WINDOWS\system32\SHLWAPI.dll
0x769c0000 0xb30000 5.01.2600.2180 C:\WINDOWS\system32\USERENV.dll
0x5ad70000 0x380000 6.00.2900.2180 C:\WINDOWS\System32\UxTheme.dll
0x773d0000 0x102000 6.00.2900.2180 C:\WINDOWS\WinSxS\x86_Microsoft.Windo
\comctl32.dll
0x5d090000 0x970000 5.82.2900.2180 C:\WINDOWS\system32\comctl32.dll
0x77690000 0x210000 5.01.2600.2180 C:\WINDOWS\System32\NIMARTIA.DLL
0x76f60000 0x2c0000 5.01.2600.2180 C:\WINDOWS\system32\WLDAP32.dll
0x71bf0000 0x130000 5.01.2600.2180 C:\WINDOWS\System32\SAMLIB.dll
0x20000000 0x2c5000 5.01.2600.2180 C:\WINDOWS\System32\xpsp2res.dll
0x5aa90000 0x70000 6.00.2600.2180 c:\windows\system32\w3ssl.dll
0x6f290000 0x160000 6.00.2600.2180 C:\WINDOWS\system32\strmfilt.dll
0x77fe0000 0x110000 5.01.2600.2180 C:\WINDOWS\System32\Secur32.dll
0x77a80000 0x940000 5.131.2600.2180 C:\WINDOWS\system32\CRYPT32.dll
0x77b20000 0x120000 5.01.2600.2180 C:\WINDOWS\system32\MSASN1.dll
0x67570000 0x90000 5.01.2600.2180 C:\WINDOWS\System32\HTTPAPI.dll
0x71ab0000 0x170000 5.01.2600.2180 C:\WINDOWS\System32\WS2_32.dll
0x71aa0000 0x80000 5.01.2600.2180 C:\WINDOWS\System32\WS2HELP.dll

```

Figure 32: listdlls.exe Output for svchost.exe

Step 3: Hash and Verify Each of the DLLs

The third step in our quest to check and verify the loaded DLLs for the *svchost.exe* process is to hash and compare each loaded DLL against a safe set. To hash each one of the DLLs, you can use any cryptographic hash utility. For this demonstration we used the *MD5deep.exe* utility to perform a recursive hash on all DLL files stored in the *C:\WINDOWS\System32* folder and its subfolders. Later, we will search for the identified DLLs to make sure they hashed correctly.

The primary reason to perform a recursive hash on all files in the *System32* folder is that it is a much quicker method of hashing all of the necessary *svchost.exe* process loaded DLLs, since most of them reside in the *System32* folder. Otherwise, we would have to hash each one of the DLLs separately, which there is no easy way to do and which would take a serious amount of time.

The *MD5deep.exe* hash utility works by first computing cryptographic hashes for all files you designate from the command line and then comparing the computed hashes against a defined file that contains a safe set of hashes. The safe set of hashes that we will be using to compare against is the National Software Research Libraries library of cryptographic hashes.

Figure 33 displays the *MD5deep.exe* utility's command line options, as well as an example of the command line syntax used to perform a recursive MD5 hash on all files in a designated directory. The recursive hash is performed on the *System32* folder and its subdirectories. The utility then compares those computed hashes against a stored set of safe hashes, which are

stored in the *NSRLFile.txt* file. The matched hashes are then output to a *DLLs.txt* text file, which we will search later. Note that when the *-M* command line argument is used, the utility will flag and output to the *DLLs.txt* text file only hashes of files that the *MD5deep.exe* utility matched correctly against the *NSRLFile.txt*.

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Jake\Desktop>md5deep.exe -h
md5deep.exe version 1.5 by Jesse Kornblum.
c:\> md5deep.exe [-v|-V|-h] [-m|-M|-x|-X <file>] [-resz@lbt] [-o fbcplsd] FILES

-v - display version number and exit
-V - display copyright information and exit
-m - enables matching mode. See README/man page
-x - enables negative matching mode. See README/man page
-M and -X are the same as -m and -x but also print hashes of each file
-r - enables recursive mode. All subdirectories are traversed
-e - compute estimated time remaining for each file
-s - enables silent mode. Suppress all error messages
-z - display file size before hash
-b and -t are ignored; present only for compatibility with md5sum
-0 - use /0 as line terminator
-l - use relative paths
-o - Only process certain types of files:
    f - Regular File      l - Symbolic Link
    b - Block Device      s - Socket
    c - Character Device  d - Solaris Door
    p - Named Pipe (FIFO)

C:\Documents and Settings\Jake\Desktop>md5deep.exe -M NSRLFile.txt -r -e -z -o f C:\WINDOWS\System32 >DLLs.txt_

```

Figure 33: MD5deep Utility

2.1.8.8.2 The Search

To find out which *svchost.exe* DLLs hashed correctly, we can perform a string search on the *DLLs.txt* file to look for the DLLs used by the *svchost.exe* process. The search criteria will be the filenames of the *listdlls.exe* utility output for the *svchost.exe* process.

Figure 34 is a screen shot of using the *grep.exe* utility for performing a string search on the *DLLs.txt* text file to see if hashes for the required DLLs exist. Note that this is not all of the DLLs that the *svchost.exe* process required, but for demonstration purposes three DLLs are provided.

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Jake\Desktop>grep -e kernel32.dll ntdll.dll ADVAPI32.dll SHELL32.dll DLLs.txt

```

Figure 34: Performing a String Search Using grep

If everything goes well, the output of your search should be hashes of the files you expected. However, if your search of the DLLs.txt file does not produce all of the searched DLLs that you provided, there are two possible reasons. The first reason is that some of the DLLs you hashed in the *System32* folder did not match the supplied hashes for those files stored in the NSRL file. The second reason is that you may have some corrupted DLLs on your system. Generally, the first is the case, because the list of hashes the National Software Research Library provides may not be up to date or might not even have the particular DLL.

2.1.8.8.3 Check and Verify a Process Image

The process of checking and verifying a running process's binary image follows the same approach as verifying a process's DLLs. The method of checking and verifying a process binary involves three steps: (1) identify the process, (2) identify the location of the process binary, and (3) hash the process binary and compare the hash against a known safe hash for that binary. Safe hashes could be either from NSRL library or (if you hashed all .exe files on your system during a baseline collection) from a baseline system.

When you build a machine, it is a best practice to perform a baseline hash of all critical files on the system such as .exe files and DLLs so that you have a known good safe state. If an incident were to occur, you could rehash all of your critical files and compare them against your stored safe baseline set.

Step 1: Identify the Process

Identify the potential rogue process by using the process enumeration utility *pslist.exe*. Figure 35 displays the process that we used for this task (i.e., *mshearts.exe* 2840).

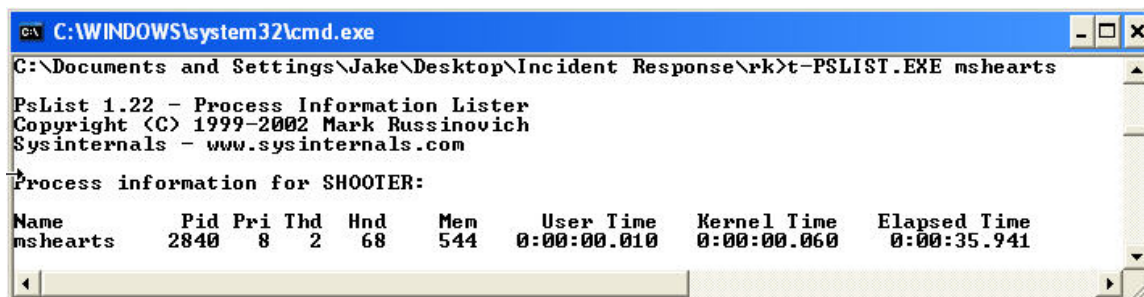


Figure 35: The *mshearts.exe* 2840 Process

Step 2: Identify the Location of the Process Binary

To identify the location of the process binary for the executing *mshearts.exe* PID 2840 process, we can use the *listdlls.exe* utility. Figure 36 is a screenshot of the *listdlls.exe* utility with the file location of the *mshearts.exe* binary.



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Jake\Desktop\Incident Response\rk>LISTDLLS.exe mshearts

ListDLLs v2.25 - DLL lister for Win9x/NT
Copyright (C) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

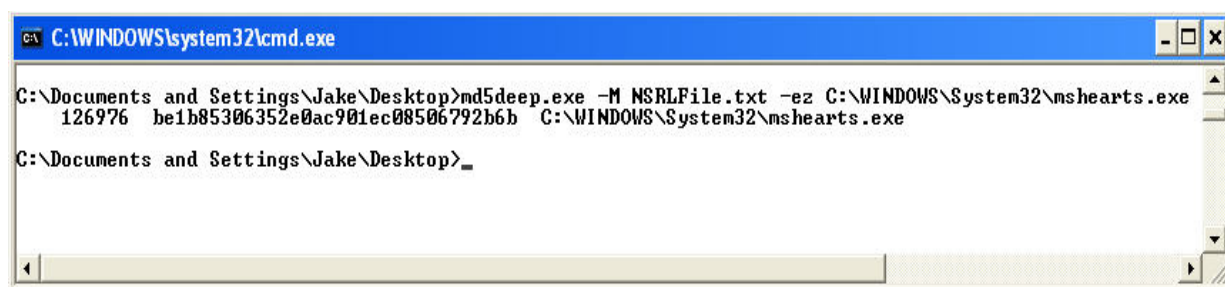
-----

mshearts.exe pid: 2840
Command line: "C:\WINDOWS\system32\mshearts.exe"
```

Figure 36: listdlls.exe Output for the mshearts Process

Step 3: Hash the Process Binary and Compare

Now that we have identified the process and the location of the process binary, the third and final step is to hash the *mshearts.exe* process binary and compare that hash against a known safe hash for the *mshearts* application. We use the *MD5deep.exe* hash utility to first hash the potential rogue *mshearts.exe* binary and compare the hash against the NSRL list of safe hashes for critical system files for a Windows XP system. Figure 37 is the command line syntax for using the *MD5deep.exe* utility to accomplish this task.



```
C:\WINDOWS\system32\cmd.exe

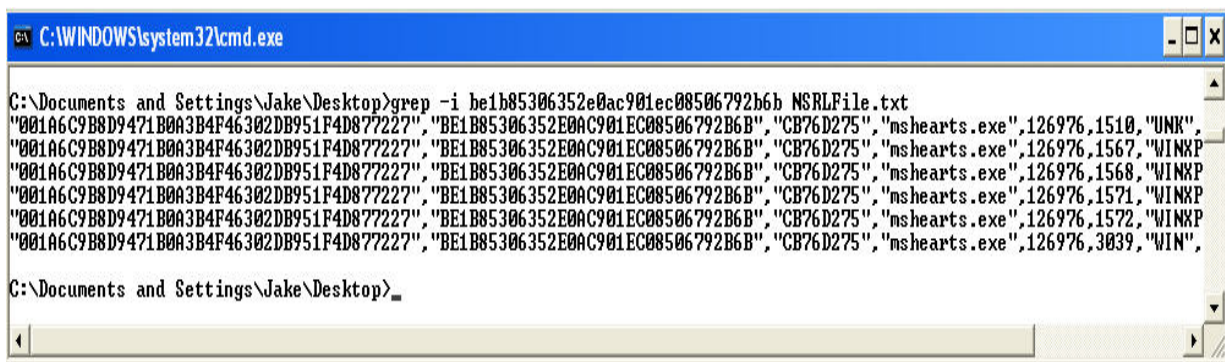
C:\Documents and Settings\Jake\Desktop>md5deep.exe -M NSRLFile.txt -ez C:\WINDOWS\System32\mshearts.exe
126976 be1b85306352e0ac901ec08506792b6b C:\WINDOWS\System32\mshearts.exe

C:\Documents and Settings\Jake\Desktop>
```

Figure 37: MD5deep.exe Command Line Arguments

As you can see, the *MD5deep.exe* utility computed the hash of the *mshearts.exe* file and, since the utility was in matching mode as defined by the *-M* flag, the *mshearts.exe* hash matched against a stored hash in the NSRL text file. So now we can assume some reliability in the *mshearts.exe* program, since the hashes matched.

To verify this, we can perform a string search for the displayed hash (be1b85306352e0ac901ec08506792b6b) in the *NSRLFile.txt* file to make sure that the appropriate hash exists.



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Jake\Desktop>grep -i be1b85306352e0ac901ec08506792b6b NSRLFile.txt
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 1510, "UNK",
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 1567, "WINXP",
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 1568, "WINXP",
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 1571, "WINXP",
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 1572, "WINXP",
"001A6C9B8D9471B0A3B4F46302DB951F4D877227", "BE1B85306352E0AC901EC08506792B6B", "CB76D275", "nshearts.exe", 126976, 3039, "WIN",
C:\Documents and Settings\Jake\Desktop>
```

2.1.8.8.4 Process String Search and Analysis

Another method of further investigating a potential rogue process is performing a string search on the binary file to see whether you can gather additional information about the process binary and its functionality.

Performing a string search on a potential rogue binary involves a few steps. The first step is to identify the potential rogue running process and the location of its binary and then use a few command line utilities to perform a string search on the process executable. To demonstrate this, we will look at the potential rogue process called *svchost1.exe*.

The *svchost1.exe* process's binary was found to be located in the *C:\WINDOWS\System32* folder. Now that we have the location of the potential rogue process's binary, the next step is to perform a string search on the binary. To do this, we used Sysinternals' *strings.exe* command utility.

*Working on NT and Win2K means that executables and object files will many times have embedded UNICODE strings that you cannot easily see with a standard ASCII strings or grep programs. So we decided to roll our own. Strings just scans the file you pass it for UNICODE (or ASCII) strings of a default length of 3 or more UNICODE (or ASCII) characters. Note that it works under Windows 95 as well.*¹⁰

Figure 38 is a screenshot of the *strings.exe* utility with command line syntax used for searching for Unicode strings within the *svchost1.exe* binary. Note that the *strings.exe* command line utility will not change the access time on the file on which you choose to perform a string search. Changing the access times on files is a huge concern in computer forensics, as it relates to the admissibility of collected information.

¹⁰ <http://www.sysinternals.com/utilities/strings.html>

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Jake\Desktop\String Tools>strings.EXE -a C:\WINDOWS\system32\svchost1.exe_

```

Figure 38: strings Command

Some of the identified strings that were found in the *svchost1.exe* binary are

- Thisisaverylongpassword
- Password355
- Longpassword
- /john.ini

Figure 39 is a screenshot of some of the more important Unicode strings that were found in the *svchost1.exe* process binary that led to the discovery of its actual use. As you can see, the masked *svchost1.exe* process is actually John the Ripper (i.e., a password cracker).

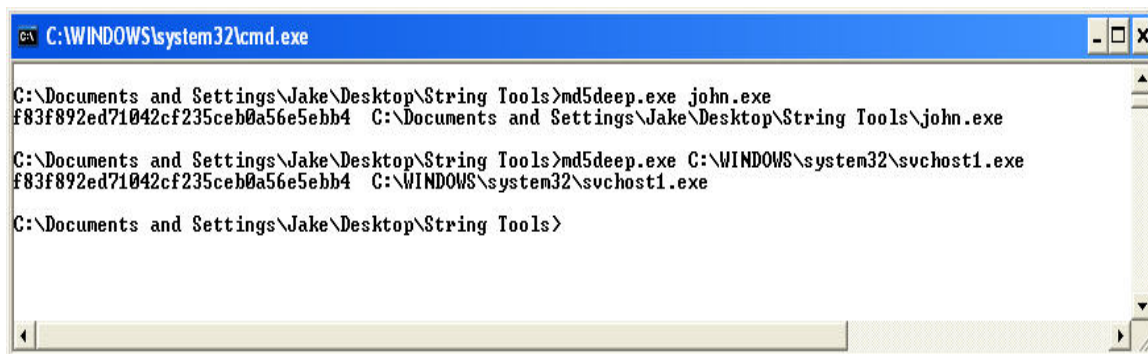
```

C:\WINDOWS\system32\cmd.exe
John the Ripper Version 1.6 Copyright (c) 1996-98 by Solar Designer
Usage: %s [OPTIONS] [PASSWORD-FILES]
-single "single crack" mode
in wordlist mode, read words from FILE or stdin
-rules enable rules for wordlist mode
-incremental[:MODE] incremental mode [using section MODE]
-external:MODE external mode or word filter
-stdout[:LENGTH] no cracking, just write words to stdout
-restore[:FILE] restore an interrupted session [from FILE]
-session:FILE set session file name to FILE
-status[:FILE] print status of a session [from FILE]
-makechars:FILE make a charset, FILE will be overwritten
-show show cracked passwords
-test perform a benchmark
-users:[-ILOGIN:UID[,...]] load this (these) user(s) only
-groups:[-IGID[,...]] load users of this (these) group(s) only
-shells:[-ISHELL[,...]] load users with this (these) shell(s) only
-salts:[-ICOUNT] load salts with at least COUNT passwords only
-format:NAME force ciphertext format NAME (DES/BSDF/BF/AFS/LM)
-savemem:LEVEL enable memory saving, at LEVEL 1..3
Invalid plaintext length requested
Password files required, but none specified
Password files specified, but no option would use them
WUS
PhP
h ;E

```

Figure 39: strings Command Output

To further verify that the *svchost1.exe* process is actually John the Ripper, we can hash the *svchost1.exe* process and see whether the hash matches the hash of a John the Ripper binary. Luckily, we can obtain the hash of John the Ripper by either downloading the hash file for the binary or downloading the binary itself from *OpenWall*. Figure 40 shows the hash signature for John the Ripper obtained by downloading the John the Ripper binary and hashing it with *MD5deep.exe* hash utility.



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Jake\Desktop\String Tools>md5deep.exe john.exe
f83f892ed71042cf235ceb0a56e5ebb4 C:\Documents and Settings\Jake\Desktop\String Tools\john.exe

C:\Documents and Settings\Jake\Desktop\String Tools>md5deep.exe C:\WINDOWS\system32\svchost1.exe
f83f892ed71042cf235ceb0a56e5ebb4 C:\WINDOWS\system32\svchost1.exe

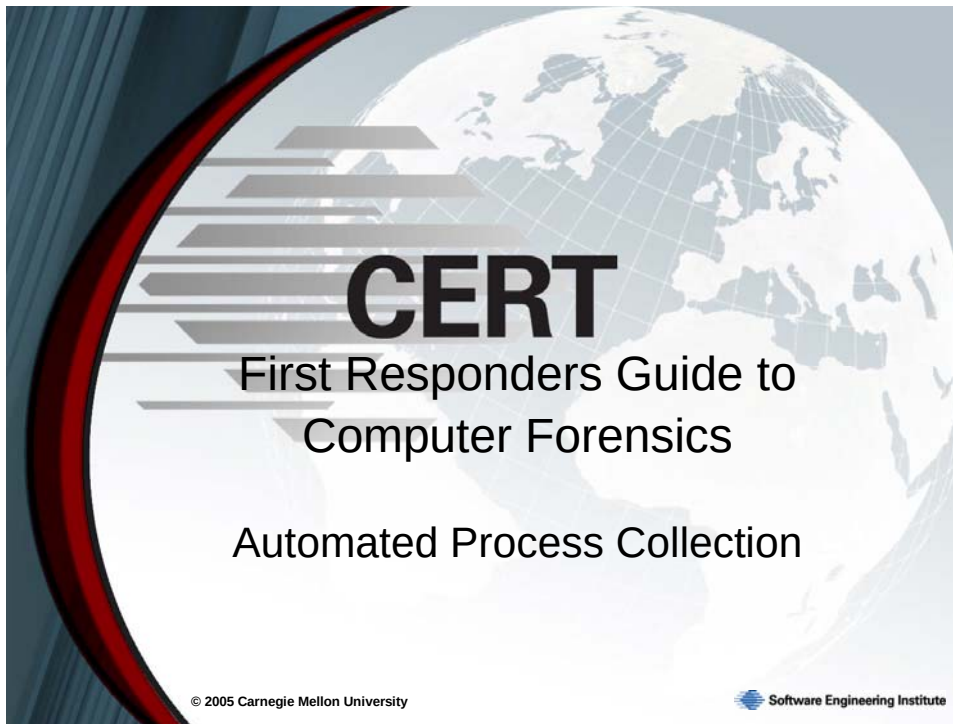
C:\Documents and Settings\Jake\Desktop\String Tools>
```

Figure 40: Hash of John the Ripper

So as you can see the hashes matched perfectly, meaning that the *svchost1.exe* process running on the system was actually John the Ripper. This method of performing string searches and hashing the binary can be applied to look further into any potential rogue processes that may be running. This was an easy example, however; some rogue process binaries may be stripped of Unicode text, making it harder to draw conclusions about what the process really is and its functionality.

Summary

- Numerous process characteristics
- Difficult to distinguish between a legitimate and a non-legitimate process
- Native commands and utilities to collect the key process characteristics
- Online resources to help identify purpose or description of many Windows processes
- Baseline documentation of the system is crucial



2.2 Automated Process Collection

In a computer security incident situation, the last thing you want to be doing is trying to piece together a set of incident response tools to use on a possibly compromised system. Automated process collection tools like developed scripts, batch files, etc., can help automate the process of collecting forensic data from a compromised machine, as well as minimize the first responder's footprint on the system.

Objectives

First Responder Utility and Forensic Server Project

- Introduction
- Configuration

Collection of process characteristics



2.2.1 Objectives

In this topic, we will be looking at an automated first responder utility called *FRUC* and how it can be used in computer security incident situations to collect volatile data and, more specifically, some of the identified process characteristics that we pointed out in the Process Characterization topic. We will also present how to properly configure and set up both the *FRUC* utility and the back-end server component *FSP*.

What are FRUC and FSP?

FRUC (First Responder Utility)

- Named after Harlan Carvey
- Command line interface tool
- Tool for collecting data (volatile and some non-volatile)

FSP (Forensic Server Project)

- Tool for retrieving data (volatile and some non-volatile)

OS Support

- Windows (2000, XP, 2003 Server)

2005 Carnegie Mellon University

3



2.2.2 First Responder Utility (FRU)

The automated process collection tool that we are going to present is Harlan Carvey's *First Responder Utility (FRU)*. *FRU* is used by first responders to retrieve volatile data from possibly compromised systems. The current version of this utility is called *FRUC*, which is a command line interface tool that uses a combination of an INI file, different command line tools and utilities, and output filenames for the collected data. The *FRUC* utility works together with the *Forensic Server Project (FSP)*, which is the server component of the *First Responder Utility*. You can use *FRUC* to collect and send captured data to the *FSP* component.

Configuration and Setup of *FRUC*

3 Components to Configure

- Server configuration section

Server IP address & port

```
1 [Configuration]
2 ; This section and information is required, but
3 ; can be overridden at the command line
4 server=192.168.2.34
5 port=7070
```

- Command section

List of commands/utilities, args, & output filenames

```
13 1=fport.exe; processopenports.txt
14 2=handle.exe; processopenfiles.txt
15 3=pslist.exe; processpriority.txt
```

- Registry section

List of registry keys and values to check

```
23 [Registry Values]
24 ; Enter the Registry key, and the value you're interested in
32 [Registry Keys]
33 1=HKCU\Software\Microsoft\Windows\CurrentVersion\Run\
34 2=HKLM\Software\Microsoft\Windows\CurrentVersion\Run\
```

2005 Carnegie Mellon University

4



2.2.2.1 First Responder Utility (FRUC) Setup

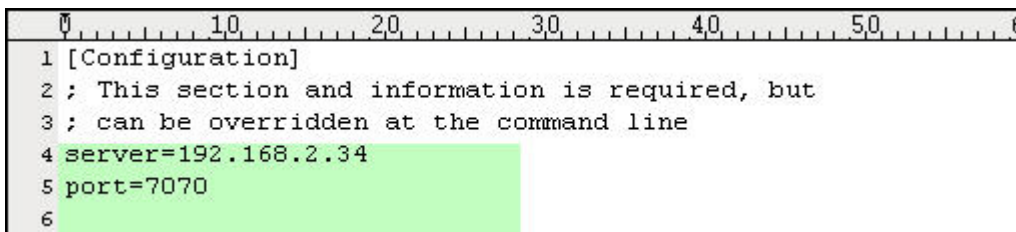
The *FRUC* utility has five components that are required for the automated tool to run.

1. *fruc.exe* – the executable that interfaces with the *fruc.ini* configuration file, *p2x584.dll* file, and the designated command line utilities
2. *fruc.ini* – the configuration file for tailoring the script to fit your collection needs. In the *fruc.ini* file, you will configure what commands and executables you want executed to collect volatile information.
3. *P2x584.dll* – the required DLL file necessary for the executable to run properly
4. list of executables – A list of executables or utilities will need to be defined so the *fruc.exe* utility can use them to collect volatile information. Tools like the discussed *pslist.exe*, *psloglist.exe*, and *etc* will need to be in the same folder as the *fruc.exe* utility.
5. *FSP* – the server component for receiving and viewing the collected volatile data

The setup for this utility to work properly requires only a few steps. The first step is to locate the *fruc.ini* file and tailor it to your needs. The *fruc.ini* file has three parts to configure.

2.2.2.1.1 Step 1: Configure the Server IP Address and Port Number

Figure 41 is a screenshot of the first part of the *fruc.ini* file that needs to be configured.



```

0      10      20      30      40      50      60
1 [Configuration]
2 ; This section and information is required, but
3 ; can be overridden at the command line
4 server=192.168.2.34
5 port=7070
6

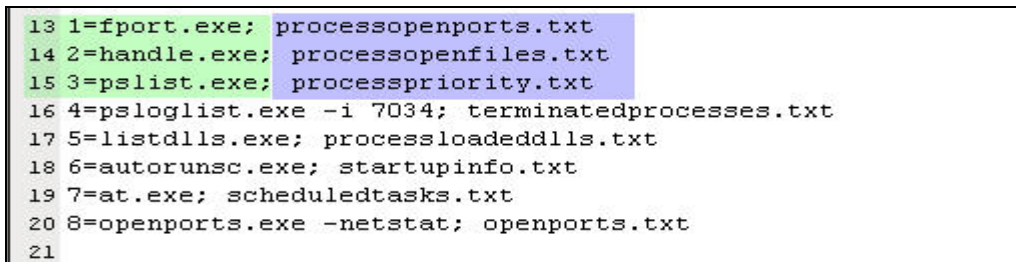
```

Figure 41: First Part of the *fruc.ini* File

The server and port settings need to be configured so that the *FRUC* utility knows where to send the collected volatile data (i.e., IP address of the *FSP* server) and on what port. Once you have this part configured, the next step is to configure the command section of the *fruc.ini* file.

2.2.2.1.2 Step 2: Configure the Command Section

The command section of the *fruc.ini* file will be the list of commands or third-party forensic utilities that will be executed in sequential order with their respected command line arguments to collect pieces of volatile data. In addition to listing the commands or utilities to be executed, you must designate a filename for the output file. Figure 42 is a screenshot of the middle part of the *fruc.ini* file that is to be configured.



```

13 1=fport.exe; processopenports.txt
14 2=handle.exe; processopenfiles.txt
15 3=pslist.exe; processpriority.txt
16 4=psloglist.exe -i 7034; terminatedprocesses.txt
17 5=listdlls.exe; processloadeddlls.txt
18 6=autorunsc.exe; startupinfo.txt
19 7=at.exe; scheduledtasks.txt
20 8=openports.exe -netstat; openports.txt
21

```

Figure 42: Second Part of the *fruc.ini* File

In the screenshot, the parts in green (lighter highlighting for black and white printing) are the commands or utilities to be executed in sequential order and the parts in blue (darker highlighting for black and white printing) are the names of the output files to be created to store the respective command output.

2.2.2.1.3 Step 3: Configure the Registry Keys Section

The registry keys section of the *fruc.ini* file will be the list of registry keys and values to collect from the compromised machine. As described, we want to check certain registry keys for auto-starting programs and services. The third part of the *fruc.ini* file allows us to enter a list of registry keys to check, as well as their key values. The final part of the *fruc.ini* file that includes a list of registry values (highlighted in green—lighter shading for black and white printing) and a list of registry keys to check (highlighted in blue—darker shading for black and white printing) is shown in Figure 43.

```

23 [Registry Values]
24 ; Enter the Registry key, and the value you're interested in
25 ; here, separated by a semi-colon
26 ; A good place to get values is from SilentRunners.org
27
28 1=HKCU\Software\Microsoft\Command Processor;AutoRun
29 2=HKLM\Software\Classes\exefile\shell\open\command;
30 3=HKCU\Software\Microsoft\Internet Explorer\TypedURLs:url1
31
32 [Registry Keys]
33 1=HKCU\Software\Microsoft\Windows\CurrentVersion\Run\
34 2=HKLM\Software\Microsoft\Windows\CurrentVersion\Run\
35
36 3=HKLM\System\CurrentControlSet\Control\Session Manager\KnownDLLs
37 4=HKLM\System\ControlSet001\Control\Session Manager\KnownDLLs
38 5=HKLM\Software\Microsoft\Windows\Current Version\Run
39 6=HKLM\Software\Microsoft\Windows\Current Version\RunOnce
40 7=HKLM\Software\Microsoft\Windows\Current Version\RunOnceEx
41 8=HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices
42
43 9=HKCU\Software\Microsoft\Windows\Current Version\Run
44 10=HKCU\Software\Microsoft\Windows\Current Version\RunOnce
45 11=HKCU\Software\Microsoft\Windows\Current Version\RunOnceEx
46 12=HKCU\Software\Microsoft\Windows\CurrentVersion\RunServices
47 13=HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows ("run=" value)

```

Figure 43: Final Part of fruc.ini File

Configuration and Setup of *FSP*

Case directory

Name of the directory/folder where the collected data will be sent

Case name

Name of the current incident response case

Investigator name

Name of the investigator or first responder

Port

Designated port to listen on

Logfile

Designated name for the case logfile

2005 Carnegie Mellon University

5

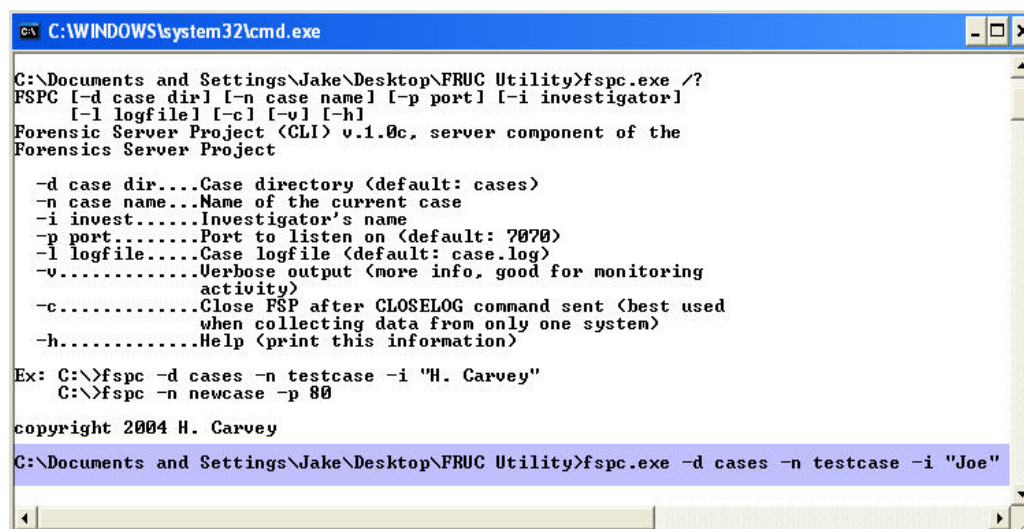


2.2.3 Forensic Server Project (*FSP*)

FSP server component works like a glorified *netcat* listener. It listens on a certain port and awaits a connection made by the *First Responder Utility*. Once a connection is established by *FRUC*, *FRUC* then sends the collected volatile data to the listening *FSP* server. Then *FSP* collects that data and puts it into separate files that are designated by each one of the executed commands *FRUC* uses.

2.2.3.1 *FSP* Setup

To set up *FSP* on the remote collection system, you will need to run the *fspc.exe* utility from the command line and pass it some of the configuration arguments highlighted in blue in Figure 44 (darker shading for black and white printing).



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Jake\Desktop\FRUC Utility>fspc.exe /?
FSPC [-d case dir] [-n case name] [-p port] [-i investigator]
      [-l logfile] [-c] [-v] [-h]
Forensic Server Project (CLI) v.1.0c, server component of the
Forensics Server Project

  -d case dir....Case directory (default: cases)
  -n case name...Name of the current case
  -i invest.....Investigator's name
  -p port.....Port to listen on (default: 7070)
  -l logfile....Case logfile (default: case.log)
  -v.....Verbose output (more info, good for monitoring
             activity)
  -c.....Close FSP after CLOSELOG command sent (best used
             when collecting data from only one system)
  -h.....Help (print this information)

Ex: C:\>fspc -d cases -n testcase -i "H. Carvey"
    C:\>fspc -n newcase -p 80

copyright 2004 H. Carvey

C:\Documents and Settings\Jake\Desktop\FRUC Utility>fspc.exe -d cases -n testcase -i "Joe"
```

Figure 44: FSP Setup

2.2.3.2 Testing *FRUC*

Now that we have made all the appropriate configuration changes to the *fruc.ini* file, it is time to test how well this utility works. We configured the server by executing the *fspc.exe* utility from the command line on the remote collection system and passing it the appropriate arguments (Figure 44). We should now be able to execute the *fruc.exe* utility to collect volatile data and send the collected data over to the listening server. Figure 45 is a display of the *fruc.exe* utility being executed from the command line with the appropriate command line arguments such as the IP address to send the collected data to, on what port, which configuration file to use, and in verbose mode.



```
D:\>cmd.exe

D:\FRUC Utility>fruc.exe -s 192.168.30.50 -p 7070 -f fruc.ini -v
Verbose mode set.
```

Figure 45: FRUC Utility Command

Execution of *FRUC* and *FSP*

Server component

```
C:\Demos\cmd.exe - fspc.exe -d Incident01 -n XPCOMPROMISED -i Jake -p 7070 -v
C:\Demos\FRUC Utility>fspc.exe -d Incident01 -n XPCOMPROMISED -i Jake -p 7070 -v
Verbose mode set.
Setup complete.
Case Name: XPCOMPROMISED\
Port      : 7070
Server started...
Awaiting connection...
Connection from 192.168.30.40
DATA command received: XPCOMPROMISEDVM- startupinfo.txt
DATA command received: XPCOMPROMISEDVM- processpriority.txt
DATA command received: XPCOMPROMISEDVM- scheduledtasks.txt
DATA command received: XPCOMPROMISEDVM- processes.txt
DATA command received: XPCOMPROMISEDVM- processopenfiles.txt
DATA command received: XPCOMPROMISEDVM- openports.txt
DATA command received: XPCOMPROMISEDVM- processopenports.txt
DATA command received: XPCOMPROMISEDVM- terminatedprocesses.txt
DATA command received: XPCOMPROMISEDVM- processloadeddlls.txt
DATA command received: XPCOMPROMISEDVM- regkeys.dat
DATA command received: XPCOMPROMISEDVM- regvals.dat
CLOSELOG command received.
```

Client component

```
C:\Demos\cmd.exe
D:\FRUC Utility>fruc.exe -s 192.168.30.50 -p 7070 -f fruc.ini -v
Verbose mode set.
```

2005 Carnegie Mellon University

6

CERT

2.2.3.3 Output of *FRUC*

Once the server component (*FSP*) starts to receive the collected data sent from *FRUC*, it will display from the command shell that a connection has been made, what collected data has been sent over, and finally a closelog message to indicate that *FRUC* has stopped collecting and transmitting volatile data.

```
C:\Demos\cmd.exe - fspc.exe -d Incident01 -n XPCOMPROMISED -i Jake -p 7070 -v
C:\Demos\FRUC Utility>fspc.exe -d Incident01 -n XPCOMPROMISED -i Jake -p 7070 -v
Verbose mode set.
Setup complete.
Case Name: XPCOMPROMISED\
Port      : 7070
Server started...
Awaiting connection...
Connection from 192.168.30.40
DATA command received: XPCOMPROMISEDVM- startupinfo.txt
DATA command received: XPCOMPROMISEDVM- processpriority.txt
DATA command received: XPCOMPROMISEDVM- scheduledtasks.txt
DATA command received: XPCOMPROMISEDVM- processes.txt
DATA command received: XPCOMPROMISEDVM- processopenfiles.txt
DATA command received: XPCOMPROMISEDVM- openports.txt
DATA command received: XPCOMPROMISEDVM- processopenports.txt
DATA command received: XPCOMPROMISEDVM- terminatedprocesses.txt
DATA command received: XPCOMPROMISEDVM- processloadeddlls.txt
DATA command received: XPCOMPROMISEDVM- regkeys.dat
DATA command received: XPCOMPROMISEDVM- regvals.dat
CLOSELOG command received.
```

Figure 46: *FSP* Command Output

When the *FSP* server receives the collected volatile data from *FRUC*, it creates text files for each one of the executed commands and appends each file with the name of the system from which it is collecting the volatile data (Figure 47). So as you can see, we have a list of eight

text files directly correlating to the eight commands that were configured to execute in the *fruc.ini* file. Also you can see, we have two .dat files containing a list of the registry keys and values that were defined to be collected.

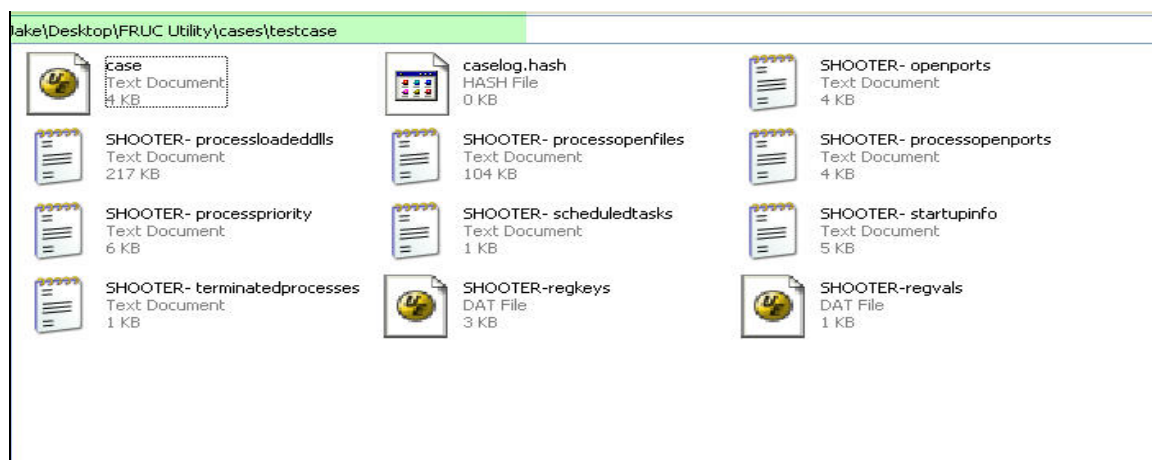


Figure 47: FRUC Output File

Another great feature with this automated collection utility is what is stored in the case text file. Within the case file is an audit trail containing the time and date the server was started (as highlighted in green—lighter shading for black and white printing), the time and date each collection utility was executed (as highlighted in blue—the line containing “openports.exe”) and finally a hash of each output file (as highlighted in orange—the line containing “openports.txt”) (Figure 48).

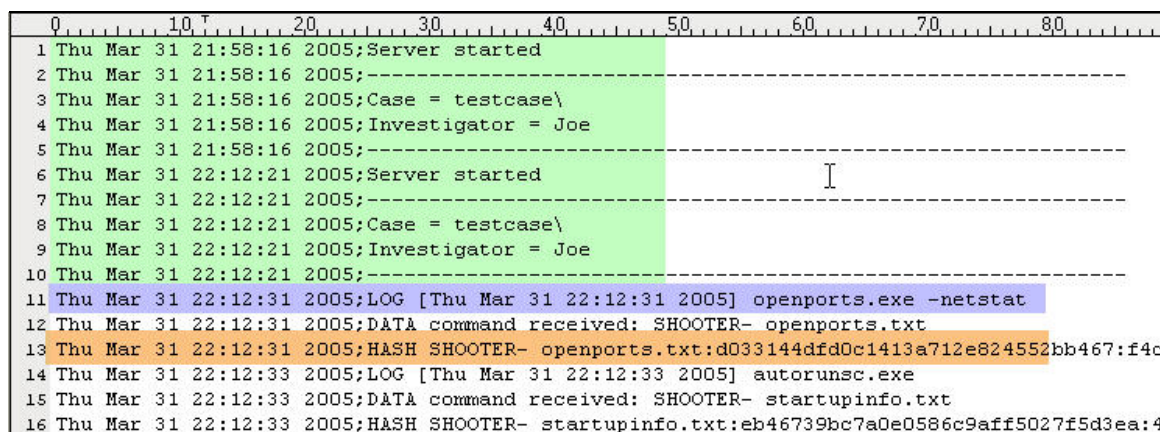


Figure 48: FRUC Audit File

Summary

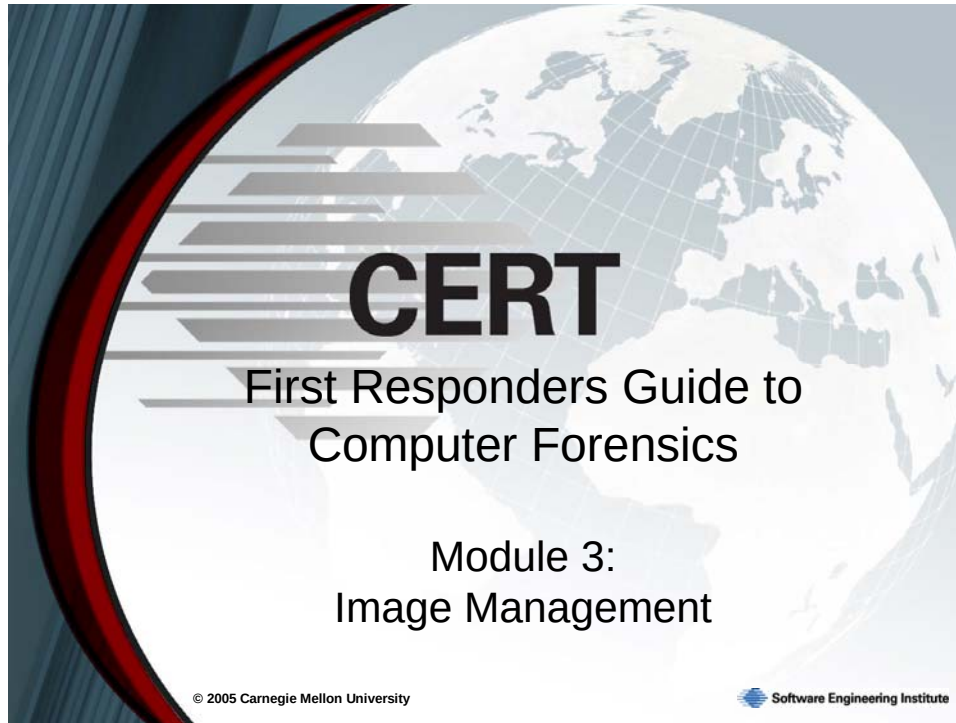
FRUC

- Versatile volatile collection tool
- Tailored to first responders' needs

FSP

- Case log acts as a forensic audit trail
- Allows for remote collection and transmission of forensic data

3 Module 3: Image Management



In this module, we take a detailed look at capturing and restoring images and image management. Included is a discussion of the *dd* tool, its syntax, and its variants. Reasons for splitting up blocks of data are enumerated, and techniques for breaking up an image and retrieving a specific file from within a captured image are described.

Additionally, we will walk through step-by-step instructions for two exercises. The first demonstrates how the *split* command can be used to break up an image. The second uses *dd* to carve a specific file out of a captured image.

dd stands for “copy and convert”



“Well, ‘cc’ was already taken for the C compiler, so the author chose the next letter in the alphabet. The syntax has sort of an evil, JCL-like quality to it. According to [The Jargon File](#), the interface was a prank.”¹

¹ <http://www.softpanorama.org/Tools/dd.shtml>

2005 Carnegie Mellon University

3



3.1 Slice and Dice with *dd*

The *dd* tool creates bit-by-bit copies, or images, of a specified file. File, in this case, is used in the “*nix” sense of the word. A file could be anything from a floppy disk to an entire hard drive. “The name *dd* stands for ‘copy and convert.’ Don’t see it? Well, ‘cc’ was already taken for the C compiler, so the author chose the next letter in the alphabet. The syntax has sort of an evil, JCL-like quality to it. According to The Jargon File, the interface was a prank.”¹¹

Initially, the syntax may look confusing, but it is pretty simple. Below is a basic example:

Table 8: dd Syntax

<code>if = file</code>	<code>[infile]</code>	(i.e., read from file vs. standard input)
<code>of = file</code>	<code>[outfile]</code>	(i.e., write to file vs. standard output)
<code>ibs = bytes</code>	<code>[input block size]</code>	(i.e., specify the number of bytes per read operation)
<code>obs = bytes</code>	<code>[output block size]</code>	(i.e., specify the number of bytes per write operation)
<code>skip= blocks</code>		(i.e., number of blocks to skip before copying starts)
<code>seek = blocks</code>		(i.e., number of blocks to skip before writing starts)
<code>count = blocks</code>		(i.e., number of blocks to copy)

¹¹ <http://www.softpanorama.org/Tools/dd.shtml>

Developed for UNIX, *dd* has since been ported to many other operating systems. There are also a few variations of the original tool. John Newbigin has written a version of *dd* for windows. It is available at <http://uranus.it.swin.edu.au/~jn/linux/rawwrite/dd.htm>. Forensic Acquisition Utilities, a suite of forensic oriented applications for Windows platforms, also contains *dd*; that can be found at <http://users.erols.com/gmgarner/forensics/>. DCFLDD is the Department of Defense Computer Forensics Laboratory's version of *dd*, which incorporates MD5 hashing and a progress status indicator. More information on that can be found at http://www.virtualwar.com/unix/cat_computerforensics.html.

Breaking up is hard to do...NOT

Why break up a perfectly good image?

- It is too big
- Inexpensive backup storage
- File size restrictions

Tools to use:

- *split*
- *md5sum*
- *cat*



2005 Carnegie Mellon University

4



There are several reasons to break up an image. The first is the issue of having a target machine too big (think RAID server or backup tapes) to feasibly have a receptacle on hand large enough to accept the entire image. In this case, on-the-fly image splitting may be needed. Additionally, it may be convenient to store a backup copy of the image on some non-volatile media to free up space for other activities. Splitting a 40GB image across several 5GB DVDs is one eminently practical and economically feasible option.

There are also cases when specific investigative tools have file size restrictions. In these cases a larger image must be broken up into manageable pieces for analysis. Imagine trying to toast an entire loaf of bread at one time. The results are much better if you take a slice at a time.

Variations on a Theme

***dd*, *dcfldd*, and *dd* for Windows share a similar syntax**

dd if=/*source* of=/*destination*

if =	file	[infile]	(i.e., read from FILE vs. standard input)
of =	file	[outfile]	(i.e., write to FILE vs. standard output)
ibs =	bytes	[input block size]	(i.e., specify the number of bytes per read operation)
obs =	bytes	[output block size]	(i.e., specify the number of bytes per write operation)
skip =	blocks		(i.e., number of blocks to skip before copying starts)
seek =	blocks		(i.e., number of blocks to skip before writing starts)
count =	blocks		(i.e., number of blocks to copy)

2005 Carnegie Mellon University

5



The *dd* tool comes with most Linux distributions. Other variations that may be downloaded, such as *dcfldd*, have enhanced features for forensics and security, including built-in MD5.

The following exercises are loosely based on the “Fun with DD” section of *The Law Enforcement and Forensic Examiner Introduction to Linux: A Beginner’s Guide*, written by Barry Grundy. Mr. Grundy’s guide is available free in PDF form online and can be found with a simple Web search.

For each *dd* exercise, detailed directions, as well as example images, have been included so that you may see the results of each step or use them as a reference for your own hands-on experience. The first exercise will cover the basics of splitting up an image and putting it back together again. The second exercise involves carving out a specific file type from a larger image.

Splitting a *dd* image and putting it back together again will be done in four steps:

1. Take a baseline hash of the original image.
2. Create an image in several parts (split).
3. Hash across the multiple image parts.
4. Put the image parts back together as a single image and hash.

For this example, a small Windows XP partition is used. We are going to split this partition into 2MB partition segments. Normally, you would have a much larger image that could be split into 2GB partitions, but for simplicity, a much smaller one is being used. Commands

that should be entered are in shaded boxes and, in most cases, are followed by the resulting output.

First, we will use MD5 to calculate the hash value of this partition. This is used to help confirm the integrity after we have split the partition and also for when it is put it back together from the split image components to confirm that it has remained unchanged.

The filename of the Windows XP partition image being used is *xpHD.dd*. The following command will return the baseline hash value of the image:

```
md5sum xpHD.dd
```

```
[root@LinuxWorkstation Demo_41]# ls
xpHD.dd
[root@LinuxWorkstation Demo_41]# md5sum xpHD.dd
251c93be7a350f148f0488b66989bcb5  xpHD.dd
[root@LinuxWorkstation Demo_41]# _
```

Figure 49: Result of Using md5 to Calculate a Hash Value

We will use the *split* command to break the 8MB image into 2MB segments. *Split* is normally used on lines in a text file. In this case, since it is a binary file, we are using *-b* to force the tool to deal with it as a binary file and ignore line breaks. The *2m* is used to specify the size of the resulting split files. Next, you specify the name of the file to be split. In this case, it's *xpHD.dd*. And finally, *xpHd.split* is the prefix of the resulting 2MB files.

```
split -b 2m xpHD.dd xPHd.split.
```

Now list the files in the directory to confirm the split. You will find the original image, *xpHD.dd*, and then the four new component split images. A suffix of *aa*, *ab*, *ac*... is appended to the end of the file prefix for each 2MB segment.

```
ls -lh
```

```
[root@LinuxWorkstation Demo_41]# split -b 2m xpHD.dd xPHd.split.
[root@LinuxWorkstation Demo_41]# ls -lh
total 16M
-rw-r--r--  1 root    root      7.8M Dec 13 14:33 xpHD.dd
-rw-r--r--  1 root    root      2.0M May 10 11:37 xPHd.split.aa
-rw-r--r--  1 root    root      2.0M May 10 11:37 xPHd.split.ab
-rw-r--r--  1 root    root      2.0M May 10 11:37 xPHd.split.ac
-rw-r--r--  1 root    root      1.8M May 10 11:37 xPHd.split.ad
```

Figure 50: Confirming the Result of Splitting Images

Check the integrity of the split images using a combination of the *cat* and *md5sum* commands. The *cat* command will put the images back together, and then we pipe that to the *md5sum* tool to find the value of the split images.

```
cat xPHd.split.a* | md5sum
```

```

[root@LinuxWorkstation Demo_41# md5sum xpHD.dd
251c93be7a350f148f0488b66989bcb5  xpHD.dd
[root@LinuxWorkstation Demo_41# split -b 2m xpHD.dd xpHd.split.
[root@LinuxWorkstation Demo_41# ls -lh
total 16M
-rw-r--r--  1 root    root      7.8M Dec 13 14:33 xpHD.dd
-rw-r--r--  1 root    root      2.0M May 10 11:37 xpHd.split.aa
-rw-r--r--  1 root    root      2.0M May 10 11:37 xpHd.split.ab
-rw-r--r--  1 root    root      2.0M May 10 11:37 xpHd.split.ac
-rw-r--r--  1 root    root      1.8M May 10 11:37 xpHd.split.ad
[root@LinuxWorkstation Demo_41# cat xpHd.split.a* | md5sum
251c93be7a350f148f0488b66989bcb5 -
[root@LinuxWorkstation Demo_41# _

```

Figure 51: Result of Using cat and md5sum to Check the Integrity of Split Images

As you see, when we compare our original MD5 hash value to the new MD5 hash value, it has remained the same.

Now we will use the *cat* command to put the split images back together into a new file. We need to specify which files we want to put back together into a new image file.

```
cat xpHD.split.a* > xpHD.new
```

Do a directory listing to confirm the new image.

```
ls -lh
```

Finally, check the integrity of the image that was put back together to confirm that it remains unchanged.

```
md5sum xpHD.new
```

```

[root@LinuxWorkstation Demo_41# md5sum xpHD.dd
251c93be7a350f148f0488b66989bcb5 xpHD.dd
[root@LinuxWorkstation Demo_41# split -b 2m xpHD.dd xpHd.split.
[root@LinuxWorkstation Demo_41# ls -lh
total 16M
-rw-r--r-- 1 root root 7.8M Dec 13 14:33 xpHD.dd
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.aa
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.ab
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.ac
-rw-r--r-- 1 root root 1.8M May 10 11:37 xpHd.split.ad
[root@LinuxWorkstation Demo_41# cat xpHd.split.a* | md5sum
251c93be7a350f148f0488b66989bcb5 -
[root@LinuxWorkstation Demo_41# cat xpHd.split.a* > xpHD.new
[root@LinuxWorkstation Demo_41# ls -lh
total 24M
-rw-r--r-- 1 root root 7.8M Dec 13 14:33 xpHD.dd
-rw-r--r-- 1 root root 7.8M May 10 11:39 xpHD.new
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.aa
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.ab
-rw-r--r-- 1 root root 2.0M May 10 11:37 xpHd.split.ac
-rw-r--r-- 1 root root 1.8M May 10 11:37 xpHd.split.ad
[root@LinuxWorkstation Demo_41# md5sum xpHD.new
251c93be7a350f148f0488b66989bcb5 xpHD.new
[root@LinuxWorkstation Demo_41# _

```

Figure 52: Result of Using md5sum to Check the Integrity of a New Image

As you can see, the value of the new hash is the same as the original file.

Data Carving

Generally done with other tools

Provides a solid understanding of what those tools are doing

Tools to use:

- *xxd*
- *grep*
- *dd*



2005 Carnegie Mellon University

6



The next example will show how to carve a specific file out of a block of data.

For the purposes of this exercise, we will be using a captured image of a floppy disk to do data carving using *dd*. While this will increase your skills using *dd*, it is not the best way to go about finding a file type. There are several automated tools that work much more efficiently. Going through the process in this manner, however, gives you a good idea of how to use *dd* in this capacity and an understanding of how the automated file searching tools actually work.

We are going to be looking in this captured image of a floppy drive for a .jpg file. In order to do that, we will start out using a hex editor to examine the *floppyimage.dd* image. Using *grep*, we are looking for the tag that delineates the beginning of a .jpg (ffd8) and finding all the places it shows up.

```
xxd floppyimage.dd | grep ffd8
```

```
[root@LinuxWorkstation flimage]# xxd floppyImage.dd | grep ffd8
0015400: b0c3 ffd8 200d 7b69 e36a a4db 94ab cc72 .... .{i.j.....r
001ae50: c643 e8c4 c80d a304 ffd8 7a74 2db0 b89a .C.....zt-...
001e550: 2d90 efd8 ffd8 fde1 007f eef8 cdeb 0156 -.....V
0028930: d4c5 9413 5df4 ffd8 f076 2364 faae ba10 ....]....v#d....
004ad40: bfbf 3df8 eafd f471 ffd8 fb3c flbe c2c3 ..=....q...<....
0074ce0: c804 2c00 80c8 042c 0080 c8fa ffd8 3300 ..,.....3.
00a5a40: 96af f471 ffd8 339f 2e34 2f97 d2a6 1167 ...q..3..4/....g
00a9120: 567c b057 ad60 fd54 b10f 80f7 ffd8 af05 V|.W.`.T.....
00ffd80: 0015 00f0 ff3b 0000 0001 0000 0000 0001 .....;.....
012c260: bb7b e08f ddee 77b3 4686 7c36 5b3d ffd8 .{....w.F.|6[=..
013ff20: 3ea3 7c47 ad9b bc34 b8c3 dc5f ffd8 d4b8 >.|G...4..._....
01489a0: bbb7 4cdb 97bc ffd8 b07d 977d dd0f b11e ..L.....}.}....
0157400: ffd8 ffe0 0010 4a46 4946 0001 0101 0048 .....JFIF.....H
```

Figure 53: Finding a .jpg Tag in a Captured Image

We are going to focus on the last line containing the .jpg tag, which reads

```
0157400: ffd8 ffe0 0010 4a46 4946 0001 0101 0048 .....JFIF.....H
```

What we have is a hexadecimal delineation of the location within the image. Translating it gives us the decimal byte offset needed to calculate the size and location of the file.

```
echo "ibase=16; 0157400" | bc
```

```
[root@LinuxWorkstation flimage]# xxd floppyImage.dd | grep ffd8
0015400: b0c3 ffd8 200d 7b69 e36a a4db 94ab cc72 .... .{i.j.....r
001ae50: c643 e8c4 c80d a304 ffd8 7a74 2db0 b89a .C.....zt-...
001e550: 2d90 efd8 ffd8 fde1 007f eef8 cdeb 0156 -.....V
0028930: d4c5 9413 5df4 ffd8 f076 2364 faae ba10 ....]....v#d....
004ad40: bfbf 3df8 eafd f471 ffd8 fb3c flbe c2c3 ..=....q...<....
0074ce0: c804 2c00 80c8 042c 0080 c8fa ffd8 3300 ..,.....3.
00a5a40: 96af f471 ffd8 339f 2e34 2f97 d2a6 1167 ...q..3..4/....g
00a9120: 567c b057 ad60 fd54 b10f 80f7 ffd8 af05 V|.W.`.T.....
00ffd80: 0015 00f0 ff3b 0000 0001 0000 0000 0001 .....;.....
012c260: bb7b e08f ddee 77b3 4686 7c36 5b3d ffd8 .{....w.F.|6[=..
013ff20: 3ea3 7c47 ad9b bc34 b8c3 dc5f ffd8 d4b8 >.|G...4..._....
01489a0: bbb7 4cdb 97bc ffd8 b07d 977d dd0f b11e ..L.....}.}....
0157400: ffd8 ffe0 0010 4a46 4946 0001 0101 0048 .....JFIF.....H
[root@LinuxWorkstation flimage]# echo "ibase=16; 0157400" | bc
1405952
```

Figure 54: Decimal Form of the Beginning of the .jpg File

The result is 1405952, which is the decimal form of the beginning of the .jpg file. We will follow the same procedure with the hex editor to find the end of the .jpg. Now that we know where the .jpg file starts, we have the starting point of our search. This time we are searching for the tag that delineates the end of a .jpg file, ffd9.

```
xxd -s 1405952 floppyimage.dd | grep ffd9
```

```
[root@LinuxWorkstation flimage]# xxd -s -1406952 floppyImage.dd | grep ffd9
004e008: e767 5092 64f0 e73f ffd9 d8dd 7fff fdb9 .gP.d..?.....
005c8b8: 254b 76ea f7df 7951 ffd9 006e 1ef0 8fb5 %Kv...yQ...n....
00a2098: d6cd 37be 4344 97ae bfd3 23ff ffd9 d096 ..7.CD....#_....
00ffd98: 0000 0002 0000 0004 0000 0005 0000 0007 .....
0132278: 7363 ddb5 90a7 7058 499f 0973 2212 ffd9 sc....pXI..s"...
0144578: ffd9 56e4 a260 5bf2 a24e 7911 1453 4d22 ..V..`[...Ny..SM"
01470a8: 8dad 5a5a ffd9 fb4b cff0 a411 7629 2c5f ..ZZ...K....v),_
014c0f8: e1f6 dfdb 7777 fffb ffd9 fc8f adfa efdd ....ww.....
[root@LinuxWorkstation flimage]#
```

Figure 55: Searching for the End of the .jpg File

Unfortunately, with this search, we did not find anything after the value specified. This process at times can be trial and error, with several search criteria attempted before finding the desired file. We will try the search again, this time spacing the ffd9 tag.

```
xxd -s 1406952 floppyImage.dd | grep "ff d9"

[root@LinuxWorkstation flimage]# xxd -s 1406952 floppyImage.dd | grep ffd9
[root@LinuxWorkstation flimage]# xxd -s 1406952 floppyImage.dd | grep "ff d9"
0159f18: b7ff d900 0000 0000 0000 0000 0000 0000 .....
```

Figure 56: Tag Delineating the End of a .jpg File

What is returned this time is what looks like the split ffd9 tag before a bunch of blank space. We'll give that a try. Using the following command, we will find the decimal address for the ending point of what we believe to be the .jpg file.

```
echo "ibase=16; 0159F18" | bc

[root@LinuxWorkstation flimage]# echo "ibase=16; 0159F18" | bc
1416984
```

Figure 57: Decimal Address for the End of the .jpg File

Now, to find out how large this file is, subtract the returned ending value from the starting point.

```
echo "1416984 - 1405952" | bc

[root@LinuxWorkstation flimage]# echo "1416984 - 1405952" | bc
11032
```

Figure 58: Calculating the Size of the .jpg File

The difference, 11032, is the size of the file. We now have the starting point and the size of the file. Here is where we can use *dd*. We are going to point *dd* at our floppy image and carve out the .jpg file (carve.jpg), skip to our starting point (1405952), take it in blocks of one (bs=1), and specify the size of the chunk to carve (11032).

```
dd if=floppyImage.dd of=carve.jpg skip=1405952 bs=1 count=11032
```

Then list the files.

```
ls -lh

[root@LinuxWorkstation flimage]# dd if=floppyImage.dd of=carve.jpg skip=1405952
bs=1 count=11032
11032+0 records in
11032+0 records out
[root@LinuxWorkstation flimage]# ls -lh
total 1.5M
-rw-r--r--  1 root    root      11K May 10 12:24 carve.jpg
-rw-r--r--  1 root    root     1.4M Dec 14 01:09 floppyImage.dd
```

Figure 59: File Carved Out Using dd

You will find the newly carved *carve.jpg* file. Use a tool such as *xview* to view the image.

```
xview carve.jpg

[root@LinuxWorkstation flimage]# xview carve.jpg
carve.jpg is a 109x110 JPEG image, color space YCbCr, 3 comps., Huffman coding
jpegLoad: carve.jpg - Premature EOF in JPEG file
_ Building XImage...done
```

Figure 60: Viewing Carved .jpg File

Summary

Images are split for a variety of reasons

- Available resources
- Backup storage
- Tool limitations

`dd` can be more than just a collection tool...
but that doesn't mean it should be.

2005 Carnegie Mellon University

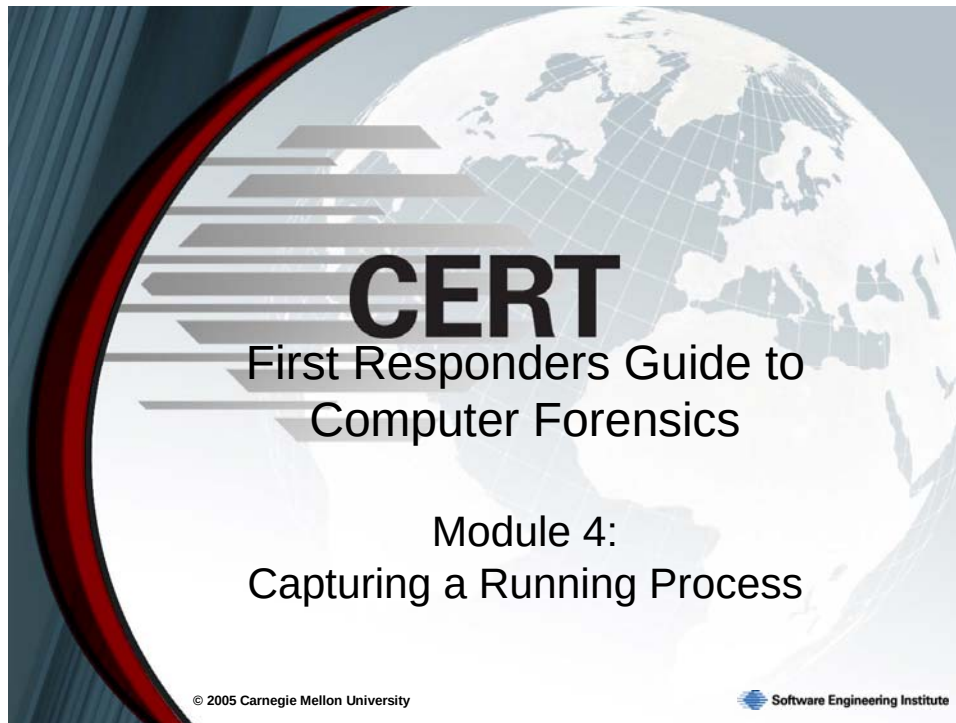
7



There are many reasons to split an image into smaller pieces. Adopting the strategy that is right for any specific situation depends on understanding the rationale behind these actions. As always, efficient and effective response to an actual security incident is largely a function of the quality of preparation carried out beforehand.

In review, this exercise showed us how to identify the beginning and the end of a file and how to use `dd` to carve out that file from within a captured image. Again, this process would not normally be done. It was used as an explanation of how `dd` and other tools work. Automated tools such as *Autopsy* and *The Sleuth Kit* will automatically identify file types and where files are located and will allow you to access the files separately from within the captured image.

4 Module 4: Capturing a Running Process



This module sets forth one technique for capturing a suspicious process from a live machine (there are other ways to perform such a capture). The important conceptual take-away from this module is to approach problems of this nature with a forensic mindset. Specifically, take pains to leave as small a footprint on the suspect machine as possible. This requires both technical and procedural preparation.

Objectives

Discuss the benefits and drawbacks of capturing a process from a live machine

Learn to capture a suspicious process on a live Windows machine

Learn to capture a suspicious process on a live Linux machine

2005 Carnegie Mellon University

2



The primary purpose of this module is to demonstrate how to capture a suspicious process from a live machine.¹² Both Windows and Linux platforms are addressed.

As you collect data (i.e., potential evidence) from a live computer, consider the data's order of volatility: that is, collect data that has the highest chance of being changed, modified, or lost first. The order of volatility for Windows and Linux computers is the same.¹³

¹² The ability to perform such a capture depends on a foundation of knowledge not contained in this section. For example, no instruction regarding the creation of a response disk consisting of safe collection tools is offered in this section.

¹³ Brezinski, D. *Guidelines for Evidence Collection and Archiving* (Network Working Group RFC 3227). <http://www.ietf.org/rfc/rfc3227.txt> (2002).

Capture a Running Process

Hedons (Pros)

- Keeps system live
- Facilitates troubleshooting
- Can collect individually targeted processes



2005 Carnegie Mellon University

Dolors (Cons)

- Leaves a footprint
- May corrupt volatile data
- Can be difficult to identify malicious processes



CERT

4.1.1 Hedons and Dolors

Hedon is a term that utilitarians use to designate a unit of pleasure. Its opposite is a *dolor*, which is a unit of pain or displeasure. There are some significant hedons associated with capturing a process from a running system. An important factor is that the system remains running. There are times when it is just not feasible to shut down a system. The ability to pull off a suspicious process for further analysis facilitates troubleshooting without sacrificing time. Additionally, it is possible to target specific processes.

It is important to remember that every silver lining has its cloud. Unlike an examination of a dead host, any action taken on a live machine leaves a footprint. The actions taken to extract the suspicious process may end up corrupting evidence. While the techniques discussed below allow for targeted extraction, it may be difficult to know exactly what to capture. There are many processes running on a machine and their names do not always provide a clear idea of what they do. Knowing what processes the machine normally runs greatly increases the chances of identifying the ones that should not be there.

Windows System

Tools

- *Netcat*, *PsList*, *ListDLLs*, *dd*, *md5sum*



Prep work

- Create response CD
- Verify someone else's response disk



Steps

- Identify suspicious processes
- Use *ListDLLs* to determine path of executable
- Copy out process using *dd* and *nc*



2005 Carnegie Mellon University

4



4.1.2 Capturing a Process on a Windows System

Table 9 shows a list of tools that can be used to capture a suspicious process on a live Windows system, as well as step by step instructions for performing the capture.

Forensic collection best practices dictate that programs on the suspect machine are not to be trusted. The tools for the collection should be put on a response disk (most likely a CD).

Table 9: Tools for Capturing Running Processes

Tool	Description
<i>PsList</i>	<i>PsList</i> is utility that shows you a combination of the information obtainable individually with <i>pmon</i> and <i>pstat</i> . You can view process CPU and memory information or thread statistics. What makes <i>PsList</i> so powerful is that you can view process and thread statistics on a remote computer. ¹⁴ Go to http://www.sysinternals.com/ntw2k/freeware/pslist.shtml to download the tool and obtain installation instructions.

¹⁴ Russinovich, Mark. *PsList*. <http://www.sysinternals.com/ntw2k/freeware/pslist.shtml> (2004).

Tool	Description
<i>ListDLLs</i>	<i>ListDLLs</i> shows you the full path names of loaded DLL modules. In addition, <i>ListDLLs</i> will flag loaded DLLs that have different version numbers than their corresponding on-disk files (which occurs when the file is updated after a program loads the DLL) and can tell you which DLLs were relocated because they are not loaded at their base address. ¹⁵ Go to http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml to download the tool and obtain installation instructions.
<i>dd</i>	<i>dd</i> (discussed in Section 3.1) is an imaging tool. It makes a bit-by-bit copy (forensic image) of the target data. The target could be an entire hard drive, a specified partition, or even the physical memory. For the purposes of this module, <i>dd</i> will be used to make an image of an executable. Go to http://uranus.it.swin.edu.au/~jn/linux/rawwrite/dd.htm to download the tool and obtain installation instructions.
<i>NetCat</i>	<i>Netcat</i> is a featured networking utility that reads and writes data across network connections using the TCP/IP protocol. It is designed to be a reliable back-end tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities. ¹⁶

Before capturing the suspicious process, prepare your response CD of safe tools for the platform from which the capture will be performed. An alternative to creating your own response CD is to use one already created. HELIX is an example of such a disk.¹⁷ In addition to being a bootable Linux environment for incident response, the disk also contains many useful tools for examining a live Windows host. FIRE is a similar resource; for more information visit <http://biatchux.dmzs.com/?section=main> or refer to the *First Responders Guide to Computer Forensics* [Nolan 05].

For the following example, two separate machines are needed. The first machine, a Windows XP box, will act as the compromised system and will be referred to as the “target machine.” This target machine has an IP address of 192.168.30.20. The second machine, a host running the WhiteBox flavor of Linux, will function and be referred to as the “collection machine.” It has an IP address of 192.168.30.50. Information will be captured from the target machine and sent to the collection machine for analysis. This example can be reproduced on any two machines connected over a network if the user has created a response CD compatible with the host operating systems and substitutes the IP addresses used in this example with the IP addresses of the machines being used.

The response CD used in this example contains trusted tools used in the capturing process. The tools have been renamed with a “t_” for clarity. For example, the executable file to call

¹⁵ Russinovich, Mark. *ListDLLs*. <http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml> (2000).

¹⁶ Giacobbi, Giovanni. *NetCat*. <http://netcat.sourceforge.net/> (2004)

¹⁷ <http://www.e-fense.com/helix/>

up a command window has been renamed from *cmd.exe* to *t_cmd.exe*. This ensures that the tool or application being used is from the response CD and not the local machine.

Now we begin the process.

1. Insert your response disk into the **target** machine.
2. Click the “Start” button.
3. Click “Run.”
4. Enter the path to the tool to be used. In this example, the CD drive is the D:\ drive and we are using *t_cmd.exe* as a trusted command shell in the WTools folder.

A command shell will open. Please note the “D:\WTools\t_cmd.exe” at the top of the command shell. This indicates that the command shell is, in fact, spawned from the trusted CD.

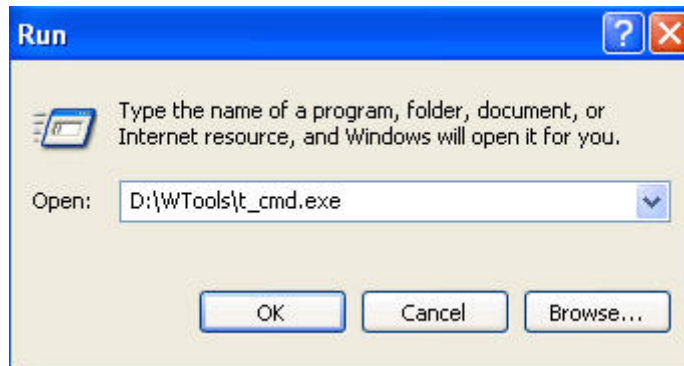


Figure 61: Running a Trusted Command

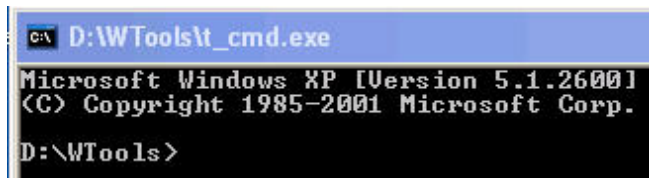


Figure 62: Command Shell Spawned from a Trusted CD

5. On the **collection** system, set up a *netcat* listener to receive the results from the *pslist* you will run on the target machine by typing the command below.

```
nc -l -p 3333 > pslistText.txt
```

The *netcat* application is called with *nc*, *-l* tells *netcat* to listen, *-p* denotes what port to listen on. Any port above the reserved ports 1-1024 can be selected for *netcat* to listen on. We chose 3333, as it's easier to keep and audit separate from other common ports. Finally, anything received on port 3333 will be written to the file *pslistText.txt*.

```
[root@LinuxWorkstation data]#  
[root@LinuxWorkstation data]# nc -l -p 3333 > pslistText.txt_
```

Figure 63: netcat Command to Listen on Port 3333

6. On the target machine, run the trusted *pslist* command and pipe the results to the collection machine via *netcat* using the command below.

```
t_pslist.exe | t_nc.exe 192.168.30.50 3333
```

We are using the trusted `pslist` and `netcat` tools from the resource CD, as denoted by the “t_.” To send the results to the collection machine, where the listener was set up, you must specify the IP address and the listening port.

```
C:\ D:\WTools\t_cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\WTools>t_pslist.exe | t_nc.exe 192.168.30.50 3333
```

Figure 64: Using Trusted `pslist` and `netcat` to Specify IP Address and Listening Port

7. Look for suspicious processes by examining the results on the **collection** machine. To do this, type the following command and then use the up and down arrows to scroll through the results.

```
cat pslistText.txt | less
```

```
[root@LinuxWorkstation data]# ls
autopsy-2.03  evidence_locker  pslistText.txt  sleuthkit-1.73
[root@LinuxWorkstation data]# cat pslistText.txt | less_
```

Figure 65: Looking for Suspicious Processes Using `cat`

8. On this example machine, take notice of a process near the bottom of the list called *tini*. *Tini* is a simple and very small (3kb) backdoor for Windows that listens on TCP port 7777 and provides a remote command prompt to anyone that connects. This process is suspicious enough for our purposes.

ccApp4	992	8	1	13	268	0:00:00.125	0:19:29.718
VMwareService	1028	13	3	43	484	0:00:00.562	0:19:29.531
tini	1196	8	3	42	420	0:00:00.156	0:19:26.593
SpoolsSV	1468	8	1	10	232	0:00:00.171	0:19:26.515
svchost1	1628	8	1	30	520	0:00:00.109	0:19:20.359
ccApp2	1648	8	1	23	4248	0:00:00.328	0:19:19.921

Figure 66: Suspicious Process Found

9. Next, we need to locate where the executable for the process is located. On the **collection** machine, set up a `netcat` listener and send the results to a text file (*tiniInfo.txt*) by typing the command below.

```
nc -l -p 4444 > tiniInfo.txt
```

Again, any port above the reserved ports can be selected to set up a listener. This session will listen on port 4444 and send the results to the file *tiniInfo.txt*.

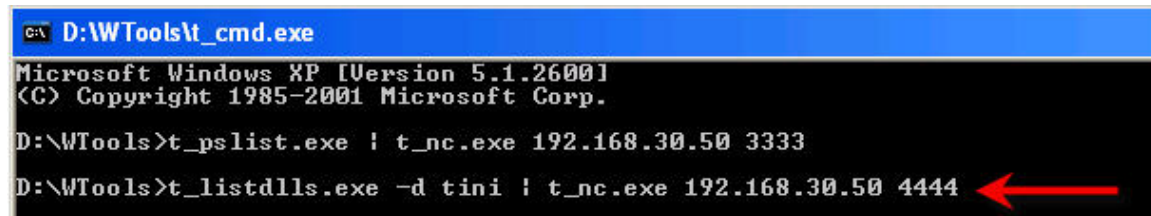
```
[root@LinuxWorkstation data]#
[root@LinuxWorkstation data]# nc -l -p 4444 > tiniInfo.txt_
```

Figure 67: netcat Command to Listen on Port 4444

10. From the **target** machine, send the data to the collection machine using the following command.

```
t_listdlls.exe -d tini | t_nc 192.168.30.50 4444
```

These tools are run from the resource CD of trusted tools and not the local machine. Specify the machine that has the *netcat* listener set up and what port it is listening on.



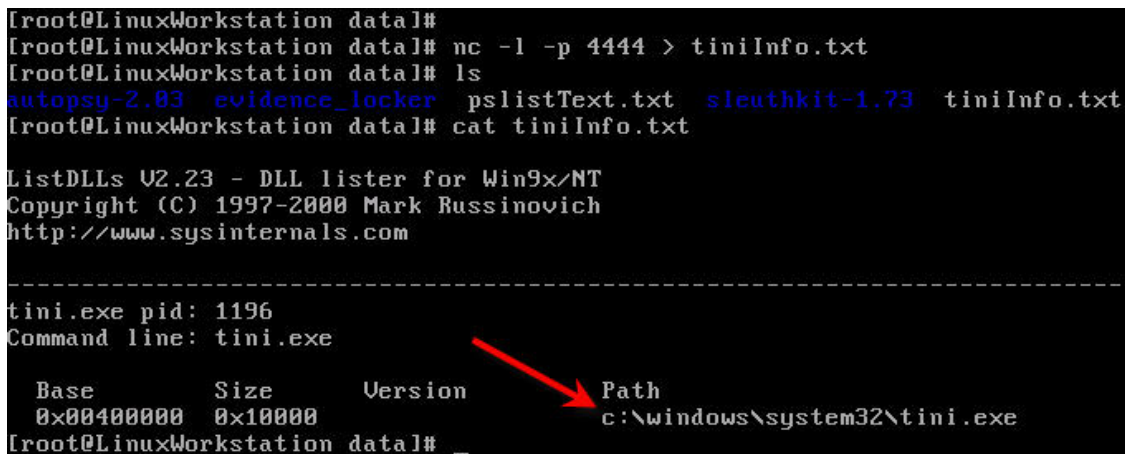
```
C:\ D:\WTools\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\WTools>t_pslist.exe | t_nc.exe 192.168.30.50 3333
D:\WTools>t_listdlls.exe -d tini | t_nc.exe 192.168.30.50 4444
```

Figure 68: Specifying netcat Listener Machine and Port

11. To view the *tiniInfo.txt* file with the results from the trusted *ListDLL* command run on the target machine, use the command below from the **collection** machine. As you can see, *tini.exe* is located at *c:\windows\system32\tini.exe*.

```
cat tiniInfo.txt
```



```
[root@LinuxWorkstation data]#
[root@LinuxWorkstation data]# nc -l -p 4444 > tiniInfo.txt
[root@LinuxWorkstation data]# ls
autopsy-2.83  evidence_locker  pslistText.txt  sleuthkit-1.73  tiniInfo.txt
[root@LinuxWorkstation data]# cat tiniInfo.txt

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

-----
tini.exe pid: 1196
Command line: tini.exe

Base      Size      Version      Path
0x00400000 0x10000
[root@LinuxWorkstation data]# _
```

Figure 69: Viewing Path to a Suspicious Process

12. Now that we know the path to the suspicious process, we are ready to collect it. On the **collection** machine, enter the following to set up a *netcat* listening session, selecting a new port and file to write the capture as previously discussed.

```
nc -l -p 5555 > capturedTINI
```

```
[root@LinuxWorkstation data]#
[root@LinuxWorkstation data]# nc -l -p 5555 > capturedTINI
```

Figure 70: Setting Up a Listening Session on a Suspicious Process

13. On the **target** machine, using the trusted tools, enter the command below to copy the executable and send it to the collection machine.

```
t_dd.exe if= c:\windows\system32\tini.exe bs=512 | t_nc
192.168.30.50 5555
```

```
D:\WTools>t_dd.exe if=c:\windows\system32\tini.exe bs=512 | t_nc.exe 192.168.30.
50 5555
Forensic Acquisition Utilities, 1, 0, 0, 1035
dd, 3, 16, 2, 1035
Copyright (C) 2002-2004 George M. Garner Jr.

Command Line: t_dd.exe if=c:\windows\system32\tini.exe bs=512
Based on original version developed by Paul Rubin, David MacKenzie, and Stuart Kemp
Microsoft Windows: Version 5.1 (Build 2600.Professional Service Pack 2)

04/03/2005 18:42:48 (UTC)
04/03/2005 13:42:48 (local time)

Current User: XPCOMPROMISEDUM\Student System

Copying c:\windows\system32\tini.exe to CONOUT$...

6+0 records in
6+0 records out
```

Figure 71: Collecting the Executable of a Suspicious Process

14. Once this is done, return to the **collection** machine and calculate a hash of the captured process. This will allow you to verify the integrity of any copies made for the purpose of analysis.

```
md5sum collectedTINI > tini.md5
```

```
[root@LinuxWorkstation data]# md5sum capturedTINI.dd > tini.md5_
```

Figure 72: Calculating a Hash of a Captured Process

Linux System

The same ... but different

- List the processes
- Identify suspicious processes
- Copy process to remote location for further analysis



Running processes *ps -aux*

Image file using *dd*

2005 Carnegie Mellon University

5



It is possible to leave a much smaller footprint on a Linux box than on a Windows machine. This is because the trusted tools on a Linux response disk can be completely self-contained. Without access to Windows source code, it is much more difficult to create completely self-contained trusted tools.

When responding to a Linux system, the procedure is pretty much the same, unless the file that spawned the process has been deleted from the running system. The *ps -aux* command will list the running processes with associated binaries or the command-line arguments used to execute them. Once the location is enumerated, the *dd* or *cp* tools can be used to copy the file(s).

However, unlike Windows, a Linux user can delete the file used to launch a running process (and other files opened by the process) once the process has been executed and is running in memory. The file space will remain protected as long as the process continues to run and won't be overwritten. Retrieving the file data becomes significantly more complicated because the file associated with the process is no longer visible to ordinary file system tools, such as *ls*. If the process is selectively terminated or a reboot or shutdown occurs, then the file space will be marked as free and the process information may be lost from physical memory, as would be expected.

It is quite feasible to recover the data from a deleted file bound to a process, partly because the operating system has to protect the disk space while the process is running. While the file name is no longer visible in the directory structure, the inode that allocates the data space for the file is preserved until the process terminates. It's easy to search for files that have been deleted (or "unlinked") but are still protected by running processes. The command *lsof +aL1*

will list all open files with an inode value of less than one, which is the case when they have been deleted. This command will display the inode and other metadata for the unlinked file(s). There are both commercial and open source tools that will take this value and recover the associated files. One such open source tool is *icat*, part of The Sleuth Kit, which is available at <http://www.sleuthkit.org>.

Collecting such information is not an overly complex process; however, it does exceed the scope of this handbook. This topic, “Recovering a Deleted Running Process in Linux,” will be dealt with in a separate security improvement module (SIM).

Summary

Have response tools ready before they are needed

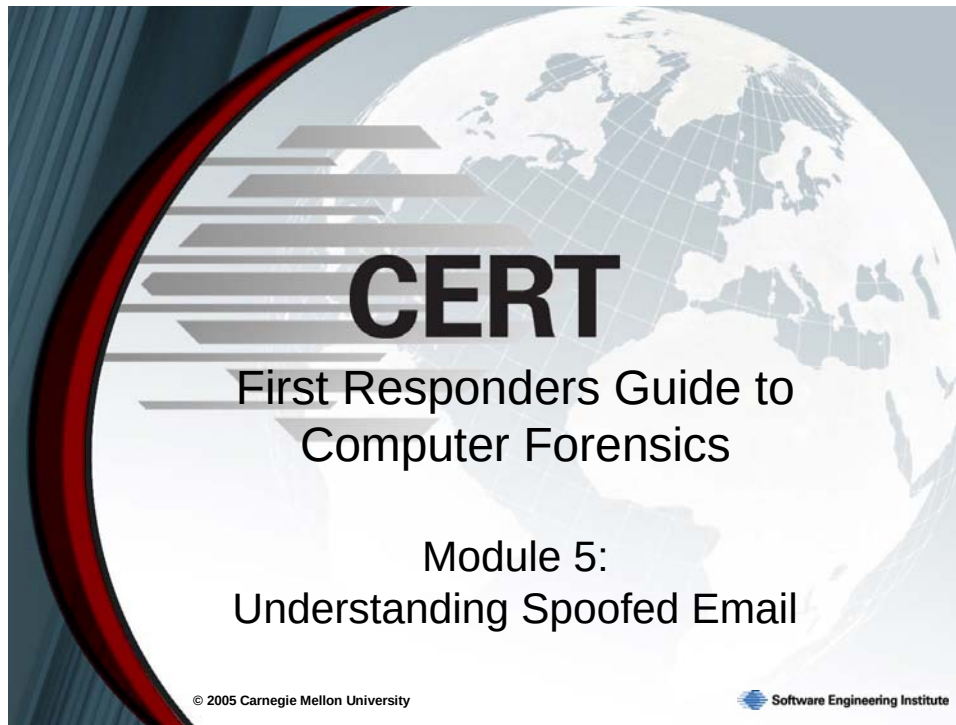
Be familiar with processes normal to the hosts on your network

Leave as small a footprint as possible during collection



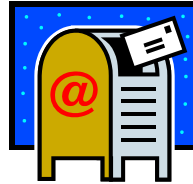
There are a few things that can be done to significantly increase the chances of successfully identifying and extracting suspicious processes from a live system. First, have the tools built and tested before they are needed. Second, have a list of processes that normally run on a system. It is much more effective to compare running processes against a list of expected processes than to rely on a gut feeling regarding what is “normal.” Finally, leave the smallest possible footprint while performing a capture.

5 Module 5: Understanding Spoofed Email



Objectives

1. Understand how email is sent and received
2. Be able to interpret email headers
3. Review how spoofed email is sent
4. Learn to identify spoofed email
5. Tools and techniques for tracing spoofed email



2005 Carnegie Mellon University

2



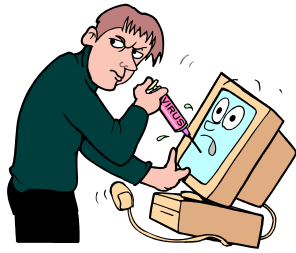
5.1 Objectives

This module has five main objectives. First, it is important for individuals to understand how email is sent and received. Understanding the life cycle of an email and its back-end processes is the crux for being able to trace back spoofed messages. Second, individuals need to be able to interpret email headers. Doing so allows one to reconstruct the path an email message takes. Third, there are a variety of ways spoofed email can be sent, and it is important to keep them in mind when attempting to trace them back. The investigative approach being used may need to be adjusted depending on the spoofing technique. Fourth, email can be spoofed with great sophistication, and it is imperative that individuals are able to distinguish well-spoofed messages from legitimate ones. Fifth, there are numerous tools and techniques that can be used to trace the origins of a spoofed email message. Understanding the purposes of each will enable a person to potentially harvest a great deal of information from a spoofed email regarding the true sender.



5.2 Identifying Spoofed Email

The Threats



Criminals

Phishing schemes and deception to extort money and personal information

Attackers

Viruses, Trojans, and worms propagated via email



2005 Carnegie Mellon University

4

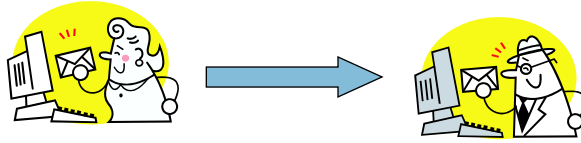


5.2.1 Definition of the Problem

Spoofed email has become a part of the daily messages that are delivered to a person's inbox. Spam has become a profitable enterprise for overzealous marketers and is a point of contention for those who receive it. However, spoofed email has become more than just a nuisance; it is a viable security threat to individual home users, organizations, and businesses. Attackers use spoofed email messages to propagate viruses, Trojans, and worms. Criminals use them for phishing schemes that attempt to extort money and information from unsuspecting users. Due to the lack of authentication in SMTP (Simple Mail Transfer Protocol), attackers and spammers can easily obfuscate their tracks and make it difficult to trace the origin of their email.

The Life Cycle of an Email ¹

Alice wants to send an email to Bob



Four computers involved

1. `alice.alphanet.com` (Alice's computer)
2. `smtp.alphanet.com` (Alice's mail server)
3. `mailhost.betanet.com` (Bob's mail server)
4. `bob.betanet.com` (Bob's computer)

2005 Carnegie Mellon University

5



5.2.2 Understanding the Process of Sending and Receiving Email

5.2.2.1 The Life Cycle of an Email

Before attempting to identify the path of an email, one must first understand its life cycle. This topic will outline the order of events from the composition of an email all the way to its delivery to the receiver. For this example we are assuming that the email is legitimate and that it is being sent outside of the sender's network.

Typically, an email is handled by a minimum of four separate computers: the computer it is sent from, the mail server of the sender, the mail server of the receiver, and the computer that receives the email. Assume that Alice wants to send an email to her friend Bob. Alice and Bob use different Internet service providers for sending and receiving email. Alice uses `alphanet.com` and Bob uses `betanet.com`. The first thing that Alice does is compose an email on her computer, which we will call `alice.alphanet.com`. When Alice completes the message, she instructs her email client to send the message. At this point, her computer, `alice.alphanet.com`, sends the email to her mail server, `smtp.alphanet.com`. When `smtp.alphanet.com` sees that the message is to be delivered to someone in the `betanet.com` domain, it sends the message to `betanet`'s mail server, `mailhost.betanet.com`. `Mailhost.betanet.com` knows that the message is for Bob and places it in his inbox. The next time Bob checks his email, Alice's message is delivered to him.

Table 10: The Life Cycle of an Email

Step 1:	Message is composed by Alice on her computer, <code>alice.alphanet.com</code>
Step 2:	<code>alice.alphanet.com</code> sends the email to <code>smtp.alphanet.com</code>
Step 3:	<code>smtp.alphanet.com</code> sends the email to Bob's email server, <code>mailhost.betanet.com</code> *
Step 4:	Bob uses his computer, <code>bob.betanet.com</code> , to check his email
Step 5:	<code>bob.betanet.com</code> retrieves Alice's email from <code>mailhost.betanet.com</code>

* At this point `smtp.alphanet.com` may not know the mail server it needs to contact; rather it may only know that it needs to send an email to someone within the `betanet.com` domain. In this case, `smtp.alphanet.com` will perform a DNS query in order to find the mail server for `betanet.com`.

Figure 73 depicts the life cycle of Alice's email to Bob.

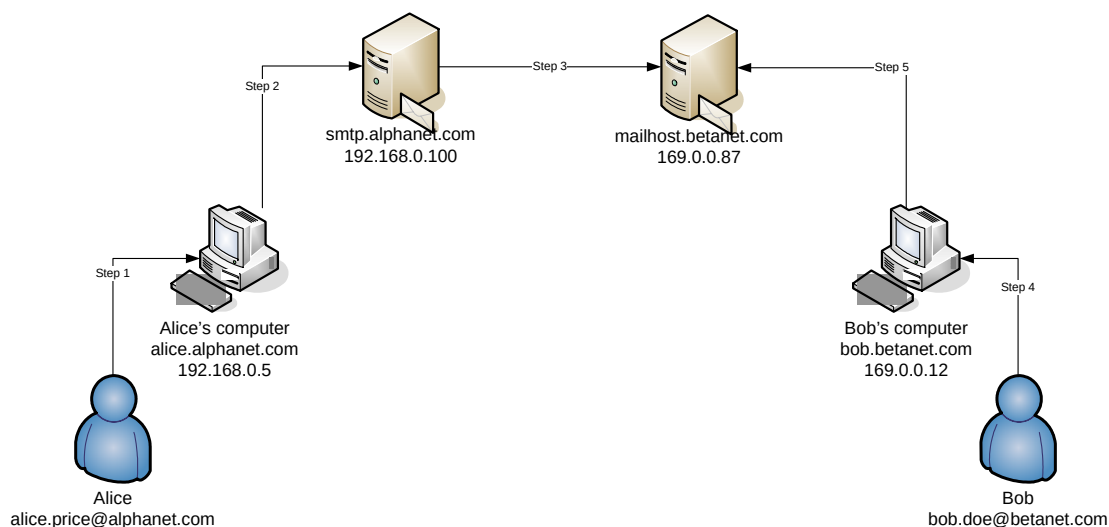


Figure 73: The Life Cycle of an Email

Overview of SMTP

Simple Mail Transfer Protocol

Developed in the early 1980s

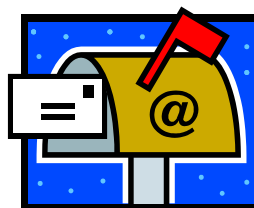
RFC 821, 2821

Acts as a push protocol

Other protocols needed to retrieve email (POP, IMAP)

Requires a TCP connection on port 25

No Authentication!



2005 Carnegie Mellon University

7



5.2.2.2 Overview of the Simple Mail Transfer Protocol

During the life cycle of an email, the process of it being sent to a mail server is handled by the Simple Mail Transfer Protocol (SMTP). SMTP was developed in the early 1980s and is outlined in RFC 821, which subsequently was obsoleted by RFC 2821 [NWG 01]. SMTP acts as a push protocol and only performs email delivery. As a result, separate protocols such as POP (Post Office Protocol) and IMAP (Internet Message Access Protocol) are needed to retrieve email messages from mail servers [Wikipedia 05e]. For the scope of this module, knowledge of POP and IMAP are not needed.

At first glance it may seem more efficient for Alice's mail client to send the email directly to mailhost.betanet.com rather than through her mail server, smtp.alphanet.com. However, Alice's mail server is much better equipped at guaranteeing delivery. During instances of connectivity interruptions or temporary computer downtime, the mail server is able to queue the message for delivery. Also, in general one can assume that a services machine will have more reliable uptime than a user machine. Lastly, mail servers have better name resolution and error handling capabilities.¹⁸

Before an email can be delivered via SMTP the client (sending computer) must initiate a TCP connection on port 25 with the receiving mail server. Once this connection is established, the client will send a sequence of commands to the server to identify itself, specify the sender, specify the receiver, pass off the email, and end the SMTP session. Other SMTP commands

¹⁸ Carnegie Mellon Computing Services. *Cyrus Technology Overview*.
<http://asg.web.cmu.edu/cyrus/1994-techoverview.html> (1994).

exist, but it is not necessary to focus on them for this topic. For a complete listing and explanation of SMTP commands, refer to RFC 2821.

SMTP Commands

Most common SMTP commands

- HELO: used by sending machine to identify itself
- MAIL: initiates a mail transaction and provides the sender's email address
- RCPT: specifies the email address of the recipient
- DATA: signifies the message portion of the email
- QUIT: signals the termination of an SMTP session

2005 Carnegie Mellon University

8



5.2.2.2.1 The HELO Command

Once the SMTP session is established, the mail server sends a 220 code (<domain> Service ready) to signal that it is ready. At this point the client will send a HELO command. The client essentially uses the HELO command to identify itself to the mail server. For example, if alice.alphanet.com was sending an email to smtp.alphanet.com, its HELO command would be "HELO alice.alphanet.com." It is important to note that the identifying information is provided by the client and there is no process of authentication to ensure that the client is who it says it is. Today, most mail servers have tools that are capable of determining the client's identity and recording it in the email headers. If the mail server accepts the client's HELO command, it replies back with a 250 code (Requested mail action okay, completed).

5.2.2.2.2 The MAIL Command

The MAIL command is used to identify the sender's email address and initiate a mail transaction. In Step 2 of Figure 73, the command would appear as "MAIL FROM: alice.price@alphanet.com." The mail server may or may not verify that the given address is valid. If the command is accepted, the server will reply with a 250 code.

5.2.2.2.3 The RCPT Command

The RCPT command is similar to the MAIL command in that it specifies the email address of the recipient. In Step 2 of Figure 73, the RCPT command would appear as "RCPT TO: bob.doe@betanet.com." This command does not verify that the email address provided is valid. If the command is accepted, the server will reply with a 250 code.

5.2.2.2.4 The DATA Command

The DATA command indicates that the client would like to transmit the message portion of the email to the mail server. If the mail server accepts this command, it responds with a code 354 (Start mail input; end with <CRLF>.<CRLF>). The client signals the end of the email by placing a “.” on a line of its own. If the command is accepted, the server will reply with a 250 code.

5.2.2.2.5 The QUIT Command

When the client wishes to terminate its SMTP session with a mail server, it issues a QUIT command.

5.2.2.2.6 SMTP Sequence of Figure 73, The Life Cycle of an Email

In Step 2 of Figure 73 the client, `alice.alphanet.com`, needs to use SMTP to deliver the email to the mail server `smtp.alphanet.com`. An SMTP transaction for this step is illustrated with client commands in bold [Lucke 04]:

```
220 smtp.alphanet.com ESMTP Sendmail 8.12.10/8.12.10
HELO alice.alphanet.com
250 smtp.alphanet.com Hello alice.alphanet.com [192.168.0.5], pleased to
meet you
MAIL FROM: alice.price@alphanet.com
250 alice.price@alphanet.com... Sender ok
RCPT TO: bob.doe@betanet.com
250 bob.doe@betanet.com... Sender ok
DATA
354 Please start mail input
From: alice.price@alphanet.com
To: bob.doe@betanet.com
Subject: Lunch

Bob,
It was good to see you again at the reunion. We should get together for
lunch the next time you are in town. Say 'hi' to your wife for me.

Regards,
Alice
.
250 Mail queued for delivery.
QUIT
221 Closing connection. Good bye.
```

Sample Email Headers

```
Return-Path: <alice.price@alphanet.com>
Received: from smtp.alphanet.com (smtp.alphanet.com [192.168.0.100])
        by mailhost.betanet.com with smtp (Exim 4.44) id 1DtsVC-0001I2-02
        Mon, 25 Jul 2005 11:40:06 -0400

Received: from alice.alphanet.com (alice.alphanet.com [192.168.0.5])
        by smtp.alphanet.com (8.12.10/8.12.10) with ESMTMP id j6PFdtHm024126
        for <bob.doe@betanet.com>; Mon, 25 Jul 2005 11:39:55 -0400
Message-ID: <42E507CC.2080100@alphanet.com>

Date: Mon, 25 Jul 2005 11:39:55 -0400
From: Alice Price <alice.price@alphanet.com>
User-Agent: Mozilla Thunderbird 1.0.6 (Windows/20050716)
X-Accept-Language: en-us, en
MIME-Version: 1.0
To: Bob Doe <bob.doe@betanet.com>
Subject: Lunch
Content-Type: text/plain; charset=ISO-8859-1Content-Transfer-Encoding: 7bit
```

2005 Carnegie Mellon University

9



5.2.3 Understanding Email Headers

During the life cycle of an email, headers are added when the email is handled by different parties. In Figure 73 headers would be added in at the time of composition, at the alphanet.com mail server and betanet.com mail server. These headers contain information regarding the computers that handle a particular email. Being able to interpret these headers is an essential component to identifying and tracing spoofed email. In an email all header names are appended by a “:”.

5.2.3.1 Interpreting Email Headers

Mail clients by default usually do not display the full headers of a message. Usually there is an option to enable the display of all the headers or to view the message source. For example, in Mozilla Thunderbird one can display the full headers through the menu option View → Headers → All. One can also view the headers by displaying the message source: View → Message Source. It is important to note that not all mail headers are identical. The exact formatting and amount of information provided depends on the configurations used by the mail clients and mail servers involved.

Assume that Bob received the email sent by Alice in Section 5.2.2.2.6. Displaying all the headers of that message would produce the following:

Table 11: Email Headers

3	Return-Path: <alice.price@alphanet.com> Received: from smtp.alphanet.com (smtp.alphanet.com [192.168.0.100]) by mailhost.betanet.com with smtp (Exim 4.44)id 1DtsVC-0001I2-02 Mon, 25 Jul 2005 11:40:06 -0400
2	Received: from alice.alphanet.com (alice.alphanet.com [192.168.0.5]) by smtp.alphanet.com (8.12.10/8.12.10) with ESMTP id j6PFdtHm024126 for <bob.doe@betanet.com>; Mon, 25 Jul 2005 11:39:55 -0400 Message-ID: <42E507CC.2080100@alphanet.com>
1	Date: Mon, 25 Jul 2005 11:39:55 -0400 From: Alice Price <alice.price@alphanet.com> User-Agent: Mozilla Thunderbird 1.0.6 (Windows/20050716) X-Accept-Language: en-us, en MIME-Version: 1.0 To: Bob Doe <bob.doe@betanet.com> Subject: Lunch Content-Type: text/plain; charset=ISO-8859-1 Content-Transfer-Encoding: 7bit

The first thing to understand about mail headers is that they are written from the bottom up. New headers are always written on top of the existing headers. In this example Alice's mail client wrote the first set of messages, Alice's SMTP server wrote the second set, and Bob's mail server wrote the third set.

5.2.3.1.1 Headers from the Client

Most of the headers added by the client such as "From:", "To:", and "Subject:" are self-explanatory. The "Date:" header in this section signifies the time the email was composed. The rest of the headers can be interpreted as follows:

```
MIME-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit
```

The email is in plain text using the ISO-8859-1 character set with 7-bit message encoding.

```
X-Accept-Language: en-us, en
```

This is an X-header, which is a non-standard header that provides additional information. The “X-Accept-Language” header informs the receiving server that email should be sent back in English.¹⁹

```
User-Agent: Mozilla Thunderbird 1.0.6 (Windows/20050716)
```

Alice used the Windows version of Mozilla Thunderbird v1.0.6 as her mail client.

5.2.3.1.2 Headers from smtp.alphanet.com

Once Alice finishes composing the email, her mail client sends it to her SMTP server, which in turn adds additional headers to the email. These headers can be found in section 2 of Table 11. Since the “Received” header consists of many components, it will be broken down line by line for better understanding.

```
Received: from alice.alphanet.com (alice.alphanet.com [192.168.0.5])
```

This message was received from a computer claiming to be alice.alphanet.com. The receiving machine determined that the sending machine’s fully qualified domain name (FQDN) is alice.alphanet.com and its IP address is 192.168.0.5.

```
by smtp.alphanet.com (8.12.10/8.12.10) with ESMTP id j6PFdtHm024126
```

This message was received by smtp.alphanet.com, which is running Sendmail version 8.12.10/8.12.10. The message was assigned an ID of j6PFdtHm024126 by smtp.alphanet.com.

```
for <bob.doe@betanet.com>; Mon, 25 Jul 2005 11:39:55 -0400
```

The message is for bob.doe@betanet.com and was received on Monday, July 25, 2005, at 11:39:55 EST (Eastern Standard Time is -0400 GMT during daylight saving time).

```
Message-ID: <42E507CC.2080100@alphanet.com>
```

The Message-ID is a unique identifier that is assigned to each message. This is usually performed by the first mail server that handles the message. The first part of the ID is usually a unique string and the second part identifies the machine that assigned the ID. This is a universal ID, as opposed to the ESMTP or SMTP ID, which is specific to the receiving machine [Lucke 04].

5.2.3.1.3 Headers from mailhost.betanet.com

Once smtp.alphanet.com processes the email, it is sent to mailhost.betanet.com, where Bob will eventually retrieve the message.

¹⁹ The A3C Connection. *Headers of a Legit Email Message*.
<http://www.uic.edu/depts/accc/newsletter/adn29/legitmail.html#Language> (2000).

Received: from smtp.alphanet.com (smtp.alphanet.com [192.168.0.100])

This message was received from a computer claiming to be smtp.alphanet.com. The receiving machine determined that the sending machine's FQDN is smtp.alphanet.com and its IP address is 192.168.0.100.

by mailhost.betanet.com with smtp (Exim 4.44)id 1DtsVC-0001I2-O2

This message was received by the computer mailhost.betanet.com, which is running Exim version 4.44. The receiving mail server, mailhost.betanet.com, assigned this message an ID of 1DtsVC-0001I2-O2 for its own records.

Mon, 25 Jul 2005 11:40:06 -0400

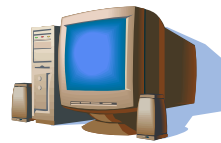
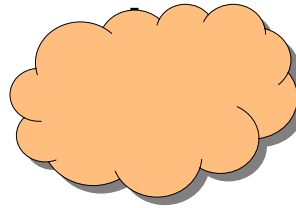
The mail server mailhost.betanet.com received this message on July 25, 2005 at 11:40:06 EST. Notice that the timestamps in the headers are in chronological order. This will help later in trying to distinguish between real headers and fake headers.

Return-Path: <alice.price@alphanet.com>

Any replies to this email should be sent to the address alice.price@alphanet.com. The "Return-Path" header is written by the SMTP server that makes the final delivery. The address in this header is the address that was provided in the MAIL command.

How Spoofed Email Is Sent

- Open relays
- Compromised machines
- Self-owned mail servers
- Temporary accounts
- Hijacked accounts



5.2.4 How Spoofed Email Is Sent

5.2.4.1 Open Mail Relay

An open mail relay refers to a mail transfer agent that will deliver mail for any sender, regardless of who it is. Up until the late 1980s, email was not delivered directly from the sender to the receiver. Instead, routes were set up where the messages would be relayed from point to point [Lucke 04]. This paradigm allowed individuals to send messages through a mail server even if they were not valid users of the system. However, this model has become open to abuse by individuals attempting to mask their origin, such as criminals and unethical advertisers. The norm now is to allow only valid users of a system to send email from it.

As an example, in the email life-cycle diagram, pretend there is a third email server called smtp.gammanet.com. Alice is a valid user of the alphanet.com domain but not of the gammanet.com domain. As a result, Alice can send email only via smtp.alphanet.com but not smtp.gammanet.com (Figure 74).

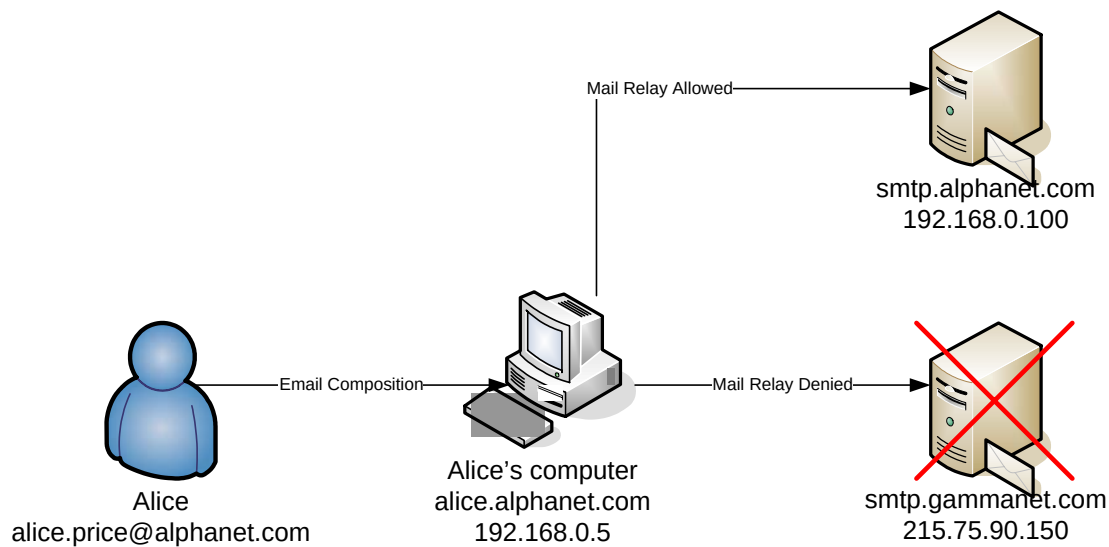


Figure 74: Mail Delivery for Valid Users

However, if smtp.gammanet.com was configured to be an open relay, Alice would not be prohibited from sending email via it. Since Alice is not a user in the gammanet.com domain and since she does not have to provide valid credentials (host identity and source email address), she can easily cover her tracks by sending email via smtp.gammanet.com. Figure 75 shows how Alice can easily spoof an email using an open relay.

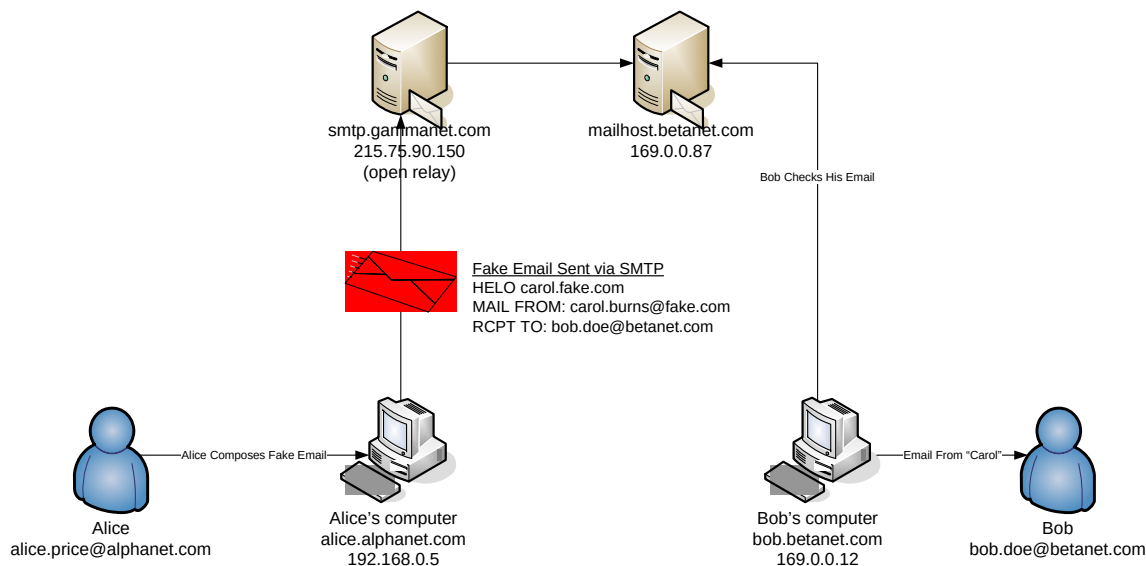


Figure 75: Spoofed Email via an Open Relay

Since most mail servers stamp the host's true identity (usually the IP address) in the email headers, Alice is not able to completely cover her trail, but it is more than enough to fool an unsuspecting recipient.

5.2.4.2 Compromised Machines

One method that spammers have used in order to gain access to open relays is to compromise machines on the Internet. Often this is through the installation of a Trojan. These applications gained their name through the similarities with the Trojan horse from the Greek story of the Trojan War. Like the horse in the myth, a Trojan application may appear to be legitimate, but in reality it is a tool used by attackers to compromise a host. Often, Trojans will be posted to file-sharing networks named as legitimate files, enticing users to download them.

The Sobig worm and its variants were received via an email with an attachment. Once the attachment was opened, it would infect any computer running Microsoft Windows. The worm would then use the host computer to spread itself by sending email to other users or copying itself to any open network shares, and would also download a proxy application from the Internet. Once downloaded, the proxy would function as a mail relay listening on a non-standard port for incoming connections or would attempt to send out mail.

5.2.4.3 Self-Owned Mail Servers

Perhaps one of the easiest ways to send spoofed email is to set up one's own mail server. There are many programs that make it easy to do this. Some readily available programs include Sendmail, Postcast and QK. Through the use of these and similar applications, a person with a broadband Internet connection could send over 1,000,000 10–50KB email messages in one hour [Sendmail 05]. When spammers use this method they can add extra received headers to obfuscate the true path of the email. Adding extra headers to an email will make it appear as though the mail was actually sent from a machine other than the spammer's mail server.

5.2.4.4 Temporary Accounts

Another method that spammers use to send spoofed email is to create temporary mail accounts with ISPs. This can be done by using false credentials or stolen credit cards. The temporary account is used until the ISP cancels it for being used to send spam. Other forms of temporary email accounts include services such as Hotmail and Yahoo. Spammers have also been known to write scripts that sign up for multiple accounts and send spam automatically.

5.2.4.5 Hijacked Accounts

An alternative to using temporary accounts or compromised machines is hijacking valid user accounts. While this is not the most popular method for spammers, there have been recorded incidents of such activity. The benefit of using temporary or hijacked accounts is that they will often be able to bypass spam filters, since they have not been used for spam and appear to be legitimate. For example, in 2002 a spammer by the name of Charles Frye used a password cracking tool called WWWHack to hijack dozens of accounts and send millions of spam messages [McWilliams 05]. In 2005 Frye was sentenced to one year in jail and six years of probation. During that time he is not permitted to use a computer.

Identifying Spoofed Email

The “Received” headers are crucial!



Received: from fusionse.com
(AMarigot-102-1-4-205.w81-248.abo.wanadoo.fr [81.248.108.205])
by mailhost@legitserver.com (8.12.10/8.12.10)
with SMTP id j6PFVvp5027789

for <realuser@legitserver.com>; Mon, 25 Jul 2005 11:31:58 -0400

2005 Carnegie Mellon University

14



5.2.5 How to Identify Spoofed Email

5.2.5.1 Carefully Examine the “Received” Headers

There are a number of telltale signs that may indicate an email is not legitimate. All of them involve interpreting a message’s headers. One of the more resourceful and useful headers is the “Received” header. Sections 5.2.3.1.2 and 5.2.3.1.3 show that this header includes the sender’s fully qualified domain name and/or its IP address. Additionally, the receiving computer determines the sender’s IP address on its own and adds that information into the header.

Received: from smtp.alphanet.com (smtp.alphanet.com [192.168.0.100])

From sender via SMTP
HELO

Stamped from the receiving machine,
mailhost.betanet.com

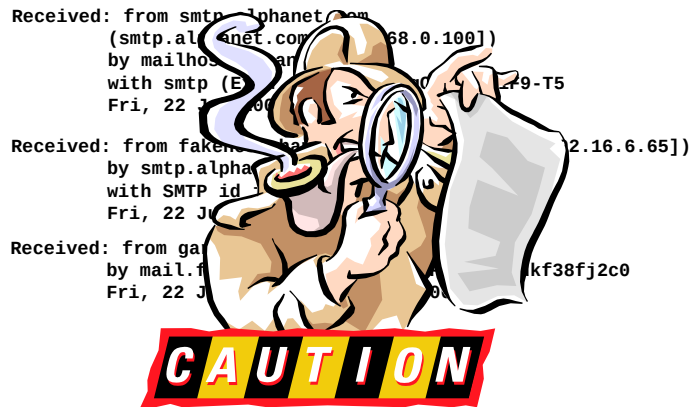
In a legitimate email the two addresses will match. However, if a sender provides invalid host information in the HELO command, it will be reflected in the “Received” header. The following “Received” header is from an actual spoofed email:

```
Received: from fusionse.com  
        (AMarigot-102-1-4-205.w81-248.abo.wanadoo.fr [81.248.108.205])  
        by mailhost@legitserver.com (8.12.10/8.12.10)  
        with SMTP id j6PFVvp5027789  
        for <realuser@legitserver.com>; Mon, 25 Jul 2005 11:31:58 -0400
```

Notice that the sender claims to be fusionse.com but the receiving mail server determined that the sender really came from an ISP in France. In this instance, it is most likely that this email was sent from a compromised host.

Fake “Received” Headers

Always read headers from the top down



2005 Carnegie Mellon University

15



5.2.5.2 Look Out for Spoofed Headers

One technique that spammers and attackers use to cover up their tracks is to add bogus headers to a message. These headers are intended to confuse individuals attempting to trace an email’s true origin. The best way not to be fooled by fake headers is to read email headers starting at the top. Since fake headers are added by the sender, they will always be beneath the real headers. Illustrated next is a set of headers from an email that contains a fake “Received” header:

```
Return-Path: <mallory@evil.org>
Received: from fakehost.happy.com (dr.evil.org [192.16.6.65])
    by mailhost.betanet.com (8.12.10/8.12.10) with
    SMTP id j6MFcRDQ015361;
    Fri, 22 Jul 2005 11:38:31 -0400
Received: from GEW@localhost by Wlr.int (8.11.6/8.11.6);
    Fri, 22 Jul 2005 12:37:22 -0400
```

In this particular set of headers, it is fairly easy to spot the fake received line. First, by reading the headers from the top down one can see there is a discontinuous set of events from the top “Received” header and the bottom one. The bottom header does not explain how this message got to dr.evil.org. Second, the information in the bottom “Received” header does not appear to contain valid host addresses (GEW@localhost and Wlr.int). Third, the timestamp in the bottom and supposedly first header is almost an hour ahead of the timestamp in the top header. This could be attributed to an erroneous clock configuration, but because of the other red flags, it is more likely to be a fake header.

5.2.5.3 Comparing Timestamps

A quick way to check the legitimacy of an email is to compare the timestamps in the “Received” headers and ensure that the chronology is reasonably accurate. Since the timestamps are written by the local, receiving machines, it is likely that they will not be perfectly in sync. A grossly misaligned timestamp may be an indication that an email is not legitimate. Fake “Received” headers will most likely contain timestamps that are out of line with the real timestamps. However, it is conceivable that a skewed timestamp is the result of bad clock configuration rather than an indicator of a spoofed email.



Tracing Spoofed Email

- *nslookup*
- *whois*



- *traceroute*
- *Sam Spade*



2005 Carnegie Mellon University

17



5.3 Tracing the Origins of a Spoofed Email

Once a spoofed email has been identified, the next step is to attempt to determine its origin. Tools such as *nslookup*, *whois*, *traceroute*, and *Sam Spade* can yield very useful information. These tools can help find information about the sending host, such as physical location, organizational affiliation, and contact information. It is possible that using these tools will not reveal the identity of the culprit; however, they will help one gather more information about the first known hop. One can then contact system/network administrators at that hop and attempt to gather further information using data from that hop's "Received" header, such as the message ID. It is important to note that even if one is able to identify the machine that sent a spoofed email, it may not be the end of the line. As has been explained in Section 5.2.4.2, many spoofed messages are sent from compromised computers. In this case, finding the true sender may involve performing forensics on the compromised machine. In this type of situation one may need to consult with legal counsel.

nslookup

Legitimate email

Received: from cpimssmtps03.msn.com [207.46.181.117]
by mx.receiver.com (mxl_mta-1.3.8-10p6)
with ESMTP id rvw67324.40386.215.y2l5;
Thu, 17 Mar 2005 06:41:14 -0500 (EST)

Do these match?

Spoofed email

Received: from cign.de ([221.153.24.156])
by mx.domain.com (8.12.10/8.12.10)
with SMTP id q6S3NUK2958123
for <receiver@domain.com>;
Wed, 27 Jul 2005 23:23:33 -0400

Do these match?



2005 Carnegie Mellon University

18



5.3.1 nslookup

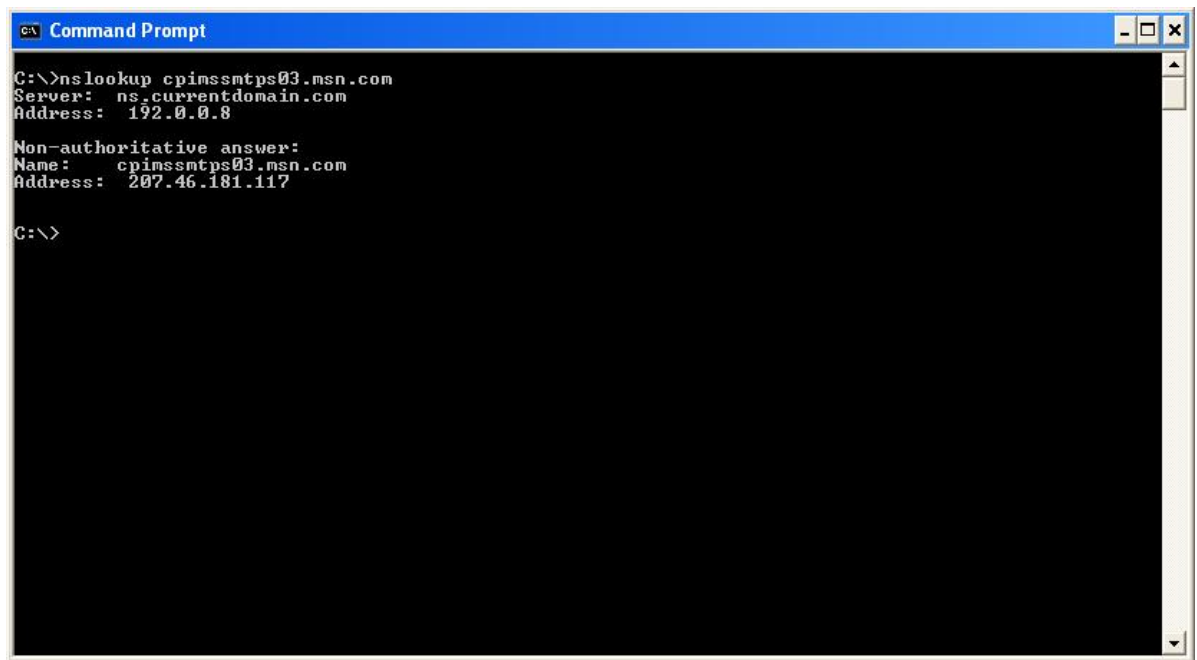
The *nslookup* name stands for name server lookup. For the purposes of spoofed email, *nslookup* is used to perform reverse DNS lookups on IP addresses and vice versa. It is a useful utility for quickly verifying the host information contained in the “Received” headers from unreliable hops. As a rule of thumb, one cannot consider hops outside of one’s own domain and control to be trustworthy. The *nslookup* tool works by querying a name server with either the IP address or FQDN provided by the user. From a Windows command prompt, the syntax for *nslookup* is as follows:

```
C:\> nslookup [computer to find] [name server]
```

Providing the name server is optional. If a name server is not provided, *nslookup* will use the default name server, which is usually the name server for the domain that the querying machine is on. The default name server may not provide an authoritative answer. To get an authoritative answer, one may need to query the name server for the computer in question.

The following “Received” header is from a legitimate email. Notice that when an *nslookup* is performed, it matches the IP address recorded by the receiving mail server:

```
Received: from cpimssmtps03.msn.com [207.46.181.117]  
by mx.receiver.com (mxl_mta-1.3.8-10p6)  
with ESMTP id rvw67324.40386.215.y2l5;  
Thu, 17 Mar 2005 06:41:14 -0500 (EST)
```



```
C:\>nslookup cpinssmtps03.msn.com
Server: ns.currentdomain.com
Address: 192.0.0.8

Non-authoritative answer:
Name: cpinssmtps03.msn.com
Address: 207.46.181.117

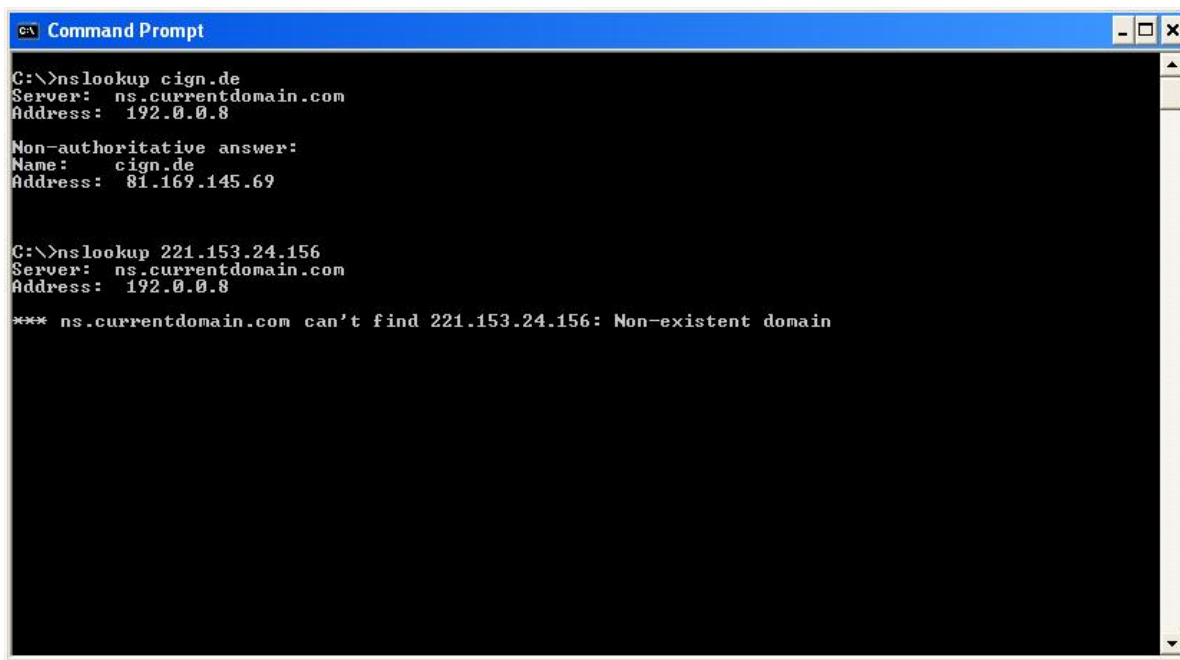
C:\>
```

Figure 76: nslookup of Valid Fully Qualified Domain Name

The following “Received” header is from a spoofed email:

```
Received: from cign.de ([221.153.24.156])
        by mx.domain.com (8.12.10/8.12.10) with SMTP id q6S3NUK2958123
        for <receiver@domain.com>; Wed, 27 Jul 2005 23:23:33 -0400
```

The sender claims to be from cign.de, but this information cannot be relied on. An *nslookup* can be used in this instance to see if the IP address for cign.de matches the IP address recorded by the receiving machine. The results from *nslookup* (Figure 77) show that the sender falsified the host information. The address for cign.de, 81.169.145.69, does not match the real address of the sender, which is 221.153.24.156. Now that the true identity of the sender has been discovered, *nslookup* can be used to attempt to find the host’s fully qualified domain name.



```
C:\>nslookup cign.de
Server: ns.currentdomain.com
Address: 192.0.0.8

Non-authoritative answer:
Name: cign.de
Address: 81.169.145.69

C:\>nslookup 221.153.24.156
Server: ns.currentdomain.com
Address: 192.0.0.8

*** ns.currentdomain.com can't find 221.153.24.156: Non-existent domain
```

Figure 77: nslookup of Falsified Host Information

Figure 77 also shows that when *nslookup* is run on the IP address 221.153.24.156, the name server cannot find information about that particular IP. The name server used by *nslookup* will not always be able to provide information on a given IP address. In these instances it will be necessary to find out the owner of the IP block that contains the address in question.

Whois – Dual Purpose

IP Block Lookup

Who owns 221.153.24.156?



Domain Name Ownership

Who does b2bpost.com
belong to?

2005 Carnegie Mellon University

19



5.3.2 whois

The *whois* utility can be used to determine the owner of a particular IP block, as well as yield greater information on a domain such as location, contact information, and name servers. All of this information can be useful in tracking down the sender of a spoofed email. *Whois* is a command line utility that is available in the Linux environment but is not native to Windows. In order to use *whois* in a Windows environment, one will need to download a third-party utility. Section 5.3.4 briefly describes the *Sam Spade* tool for Windows, which incorporates *whois*. *Whois* queries can also be performed via websites such as samspace.org. However, it is important to consider that the information contained in the WHOIS databases may not be completely accurate; in fact, it is not uncommon for WHOIS contact information to be falsified.

5.3.2.1 IP Block Identification

In Section 5.3.1, *nslookup* failed to provide information regarding the IP address 221.153.24.156. *Whois* can be used to query the ARIN database (American Registry for Internet Numbers) to find out information about the IP address in question. ARIN is responsible for the registration and administration of IP addresses in Canada, the United States, and parts of the Caribbean [Wikipedia 05a]. They maintain a publicly accessible database that contains ownership information about IP blocks within their geographic domain. To query the ARIN database in a Linux environment, the following command is used: `whois 221.153.24.156@whois.arin.net`.

```

[12:03] unix42 [18] ~> whois 221.153.24.156@whois.arin.net
[whois.arin.net]

OrgName:      Asia Pacific Network Information Centre
OrgID:        APNIC
Address:      PO Box 2131
City:         Milton
StateProv:    QLD
PostalCode:   4064
Country:      AU

ReferralServer: whois://whois.apnic.net

NetRange:     221.0.0.0 - 221.255.255.255
CIDR:         221.0.0.0/8
NetName:      APNIC7
NetHandle:    NET-221-0-0-1
Parent:
NetType:      Allocated to APNIC
NameServer:   NS1.APNIC.NET
NameServer:   NS3.APNIC.NET
NameServer:   NS4.APNIC.NET
NameServer:   NS-SEC.RIPE.NET
NameServer:   TINNIE.ARIN.NET
Comment:      This IP address range is not registered in the ARIN database.
Comment:      For details, refer to the APNIC Whois Database via
Comment:      WHOIS.APNIC.NET or http://www.apnic.net/apnic-bin/whois2.pl
Comment:      ** IMPORTANT NOTE: APNIC is the Regional Internet Registry
Comment:      for the Asia Pacific region. APNIC does not operate networks
Comment:      using this IP address range and is not able to investigate
Comment:      spam or abuse reports relating to these addresses. For more
Comment:      help, refer to http://www.apnic.net/info/faq/abuse
Comment:
RegDate:
Updated:      2005-05-20

OrgTechHandle: AWC12-ARIN
OrgTechName:   APNIC Whois Contact
OrgTechPhone:  +61 7 3858 3100
OrgTechEmail:  search-apnic-not-arin@apnic.net

```

Figure 78: WHOIS Query of ARIN

Figure 78 shows that the results of the query found the IP address registered to a net block within the Asia Pacific Network Information Centre (APNIC), which is the Asian equivalent of ARIN. The results also show that APNIC has its own database (whois.apnic.net) that can be queried in order to find more specific information. As a result, the following command is used to query APNIC's database: `whois 221.153.24.156@whois.apnic.net`. This time the results of the query yield information about the net block owner:

```

inetnum:      221.144.0.0 - 221.168.255.255
netname:      KORNET
descr:        KOREA TELECOM
descr:        Network Management Center
country:      KR
admin-c:      DL248-AP
tech-c:       GK40-AP
remarks:      *****
remarks:      KRNIC of NIDA is the National Internet Registry
remarks:      in Korea under APNIC. If you would like to
remarks:      find assignment information in detail
remarks:      please refer to the NIDA Whois DB
remarks:      http://whois.nida.or.kr/english/index.html
remarks:      *****
status:       Allocated Portable
mnt-by:       MNT-KRNIC-AP
mnt-lower:    MNT-KRNIC-AP
changed:      hm-changed@apnic.net 20030417
changed:      hm-changed@apnic.net 20041007
source:       APNIC

person:       Dong-Joo Lee
address:      128-9 Yeong-Dong Jongro-Ku Seoul
address:      Network Management Center
country:      KR
phone:        +82-2-766-1407
fax-no:       +82-2-766-6008
e-mail:       ip@ns.kornet.net
nic-hdl:      DL248-AP
mnt-by:       MAINT-NEW
changed:      hostmaster@nic.or.kr 20010425
source:       APNIC

person:       Gyung-Jun Kim
address:      KORNET
address:      128-9, Yeong-Dong, Jongro-Ku
address:      SEOUL
address:      110-763
country:      KR
phone:        +82-2-747-9213
fax-no:       +82-2-3673-5452
e-mail:       ip@ns.kornet.net
nic-hdl:      GK40-AP
mnt-by:       MNT-KRNIC-AP
changed:      hostmaster@nic.or.kr 20010906

```

Figure 79: WHOIS Query of APNIC

Figure 79, the second *whois* query, shows that the IP address in question is from Korea Telecom, an ISP in Korea. From this information one can speculate that either the spoofed email came from a home user's computer that was compromised or the culprit set up a temporary account to send spoofed email; however, one cannot be certain without more information. The WHOIS information provided by APNIC also indicates a third WHOIS database, whois.nida.or.kr, which should be queried to see whether it provides even more information. At this point, if one wishes to seek out the identity of the sender, the ISP will need to be contacted. Since most home users do not have a static IP address, one address can shuffle through multiple users. Therefore, it is important to be able to determine the time the email was sent so that the IP address can be matched with the correct user account. Note that determining the user's identity may hinge on the ISP's policies, their willingness to cooperate, how long they maintain archived logs, the level of detail contained in their logs, and the accuracy of their logs. Depending on the situation and location, legal issues and implications may also need to be considered.

5.3.2.2 WHOIS Information for a Domain Name

The most efficient method to find accurate WHOIS information for a domain name is to start from the top-level domain and work down. The following “Received” header will be used as an example:

```
Received: from b2bpost.com (b2bpost.com [209.51.220.12])  
        by mx.domain.com (8.12.10/8.12.10) with ESMTP id j6R9fhDQ014750  
        for <receiver@domain.com>; Wed, 27 Jul 2005 05:41:43 -0400
```

This header shows that the receiving mail server recorded the sender’s IP address to be 209.51.220.12 and its domain name to be b2bpost.com, which matches the HELO information provided by the sender. A quick *nslookup* of the recorded IP address verifies that the email did, in fact, come from the b2bpost.com domain.

The first step to take is to find out the WHOIS server for the .com domain. This is done by querying the WHOIS database for the Internet Assigned Number Authority (IANA), which is whois.iana.org. The command for this query is `whois com@whois.iana.org`. Figure 80 displays the results.

```
Technical Contact:  
  Name: Registry Customer Service  
  Organization: VeriSign Global Registry Services  
  Address1: 21345 Ridgetop Circle  
  Address2:  
  Address3:  
  City: Dulles  
  State/Province: Virginia  
  Country: United States  
  Postal Code: 20166  
  Phone: +1 703 925-6999  
  Fax: +1 703 421-5828  
  Email: info@verisign-grs.com  
  Registration Date: 01-January-1985  
  Last Updated Date: 01-January-1985  
  URL for registration services: http://www.verisign-grs.com  
  Whois Server (port 43): whois.verisign-grs.com
```

Figure 80: WHOIS Query of IANA

Among the information that is returned from the query is the WHOIS server, whois.verisign-grs.com, for the .com domain. One can be assured that querying this server will yield information regarding b2bpost.com. Sure enough, the following query yields the information shown in Figure 81:

```
whois b2bpost.com@whois.verisign-grs.com
```

```

[16:13] unix47 [36] ~> whois b2bpost.com@whois.verisign-grs.com
[whois.verisign-grs.com]

Whois Server Version 1.3

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

    Domain Name: B2BPOST.COM
    Registrar: REGISTER.COM, INC.
    Whois Server: whois.register.com
    Referral URL: http://www.register.com
    Name Server: DNS11.REGISTER.COM
    Name Server: DNS12.REGISTER.COM
    Status: ACTIVE
    Updated Date: 10-jun-2005
    Creation Date: 17-dec-2003
    Expiration Date: 17-dec-2007

>>> Last update of whois database: Tue, 2 Aug 2005 13:53:31 EDT <<<

```

Figure 81: Query of .com WHOIS Database

This query returned some basic information regarding the domain b2bpost.com and also provided the WHOIS server, whois.register.com, of the company where b2bpost.com registered their domain. This WHOIS server should provide definitive information regarding the domain in question. The third query, whois b2bpost.com@whois.register.com, produced detailed information about the domain, as shown in Figure 82.

```

Organization:
  Direct Media Network...
  Web Master
  PO Box 811736
  Boca Raton, FL 33481-1736
  US
  Phone: 888-807-1002
  Email: sinfotech@gmail.com

Registrar Name....: Register.com
Registrar Whois...: whois.register.com
Registrar Homepage: http://www.register.com

Domain Name: B2BPOST.COM

    Created on.....: Wed, Dec 17, 2003
    Expires on.....: Mon, Dec 17, 2007
    Record last updated on..: Tue, Jul 12, 2005

Administrative Contact:
  Direct Media Network...
  Web Master
  PO Box 811736
  Boca Raton, FL 33481-1736
  US
  Phone: 888-807-1002
  Email: sinfotech@gmail.com

```

Figure 82: Query of the Registrar's WHOIS Database

Further investigation will need to be done to determine the validity of the WHOIS information. The same issues mentioned at the end of Section 5.3.2.1 may apply.

5.3.3 *Traceroute*

While WHOIS information may yield contact information, it may not necessarily correspond to the location of the computer being investigated. It may be the case that the computer in the WHOIS database is registered to someone in a different geographical location than the actual machine. *Traceroute* determines the path a packet takes to a specific computer. This is done by sending Internet Control Message Protocol (ICMP) packets bound for the target computer and incrementing the time-to-live (TTL) for each packet. The first packet is sent with a TTL of 1 so that the packet dies after the first hop. The router at that hop responds and the *traceroute* utility now knows the first hop along the path. *Traceroute* continues to do this so that the second hop, third hop, etc. respond. By the time an ICMP packet reaches its destination, *traceroute* has mapped the entire path to it. Since the ICMP packets are sent directly from the computer that is running the *traceroute*, one may desire a different approach if a low-profile needs to be kept. There are many websites that allow users to perform *traceroutes* online. The benefit to performing a *traceroute* online is that the ICMP packets will originate from the web server instead of one's own machine. A good website for performing *traceroutes* is <http://www.dnsstuff.com>.

The *traceroute* utility can help one pinpoint the true geographic location of a system. In Windows the command to use is *tracert*. The following example illustrates how an organization can reside in one location and one of their servers can be in another. The SANS Institute is a computer security organization located in Bethesda, Maryland. Performing a reverse DNS lookup on their website, <http://www.sans.org>, reveals that the IP address of one of their web servers is 64.112.229.132. From a Windows command prompt, the following command is used to determine the location of this machine: `tracert 64.112.229.132` (Figure 83).

```
Command Prompt
C:\>tracert 64.112.229.132

Tracing route to maverick32.sans.org [64.112.229.132]
over a maximum of 30 hops:
  0  <1 ms    <1 ms    <1 ms    POD-A-UL64.GW.CMU.NET [128.237.224.11]
  1  103 ms    <1 ms    <1 ms    CORE0-UL914.GW.CMU.NET [128.2.0.155]
  2  1 ms      1 ms     1 ms     HYPER-UL501.GW.CMU.NET [128.2.33.225]
  3  1 ms      <1 ms    <1 ms     bar-cmu-ge-4-0-0-0.3rox.net [192.88.115.181]
  4  1 ms      1 ms     1 ms     minime-ge-0-1-0-0.3rox.net [192.88.115.5]
  5  23 ms     23 ms    22 ms    so-2-1-1.ar3.JFK1.gblx.net [208.50.254.45]
  6  60 ms     23 ms    23 ms    so7-0-0-2488M.ar3.NYC1.gblx.net [67.17.79.10]
  7  23 ms     23 ms    23 ms    sprint.ar3.NYC1.gblx.net [208.51.134.26]
  8  47 ms     47 ms    47 ms    sl-bb21-chi-9-0.sprintlink.net [144.232.9.149]
  9  47 ms     47 ms    47 ms    sl-bb25-chi-13-0.sprintlink.net [144.232.26.90]
 10  77 ms     77 ms    77 ms    sl-bb20-sea-1-0.sprintlink.net [144.232.20.85]
 11  78 ms     78 ms    78 ms    sl-bb22-tac-6-0.sprintlink.net [144.232.9.151]
 12  81 ms     81 ms    81 ms    sl-bb21-tac-4-0.sprintlink.net [144.232.17.93]
 13  79 ms     79 ms    79 ms    sl-gw6-tac-10-0.sprintlink.net [144.232.17.1]
 14  84 ms     84 ms    84 ms    sl-micro23-1-0-0.sprintlink.net [160.81.243.66]
 15  85 ms     87 ms    86 ms    64-112-224-3-ips-eug-or-core02.tcpiptservices.net [64.112.224.3]
 16  86 ms     91 ms    85 ms    64-112-227-69-ips-eug-or-lb01-2.tcpiptservices.net [64.112.227.69]
 17  maverick32.sans.org [64.112.229.132] reports: Destination host unreachable.

Trace complete.
C:\>_
```

Figure 83: Traceroute Example

The final destination, 64.112.229.132, did not reply to the ICMP packet, which is typical of systems that have been hardened for security purposes. The point of interest is the hop before the final destination. Hop 17 is in the tcpiptservices.net domain. A quick WHOIS query on this domain shows that the tcpiptservices.net appears to be some type of service provider in Eugene, Oregon. Therefore, while the SANS Institute is located in Maryland, their web servers are in Oregon [Mandia 01].

5.3.4 Sam Spade

Much of the information gathered using *nslookup* and *whois* can also be collected using *Sam Spade*, a Windows tool that contains various network utilities such as *whois*, *tracert*, *IP block lookup*, *DNS check*, and *ping*. *Sam Spade* is a freeware tool; it can be downloaded from the *Sam Spade* website at <http://www.samspade.org/ssw/>. Some of the utilities in *Sam Spade* are also available directly from the *Sam Spade* home page: <http://www.samspade.org>. Another website containing some very useful online tools is <http://www.dnsstuff.com>. All of the data collection techniques outlined in the previous sections can be performed using *Sam Spade*.

Summary



No legitimate use for spoofed email

Lack of authentication in SMTP

TCP connection leaves traceable fingerprint

Spoofed email sent many different ways

Header interpretation is the key

Tracing email requires investigative work

2005 Carnegie Mellon University

20



5.4 Summary

There is no legitimate reason for spoofed email to be sent. At best, email spoofing is used by unethical advertisers who churn out billions of messages and hide their tracks to avoid the repercussions of their actions. At worst, email spoofing is used to propagate all types of malicious software and to aid in various criminal activities, the consequences of which extend beyond the digital realm. The lack of authentication in SMTP is the major contributor to the spoofed email problem currently facing society. It was designed at a time when the number of users was so few that everybody knew everyone else. However, the vast frontier of the Internet has changed the paradigm to one requiring security. While a change may not be on the horizon in the immediate future, dealing with spoofed email is not a lost cause. Because TCP is used for mail delivery, senders are not able to completely cover their tracks. The TCP handshake allows the receiving mail hosts to stamp the sender's true identity in the "Received" header. As a result, spammers have responded with a variety of techniques to muddle their tracks as much as they can. However, through email header interpretation and various tools and techniques, security professionals have the ability to trace the true origins of spoofed messages.

References

URLs are valid as of the publication date of this document.

- [Ash 95]** Ashton, P. *Algorithms for Off-Line Clock Synchronization* (Technical Report TR COSC 12/952). Christchurch, New Zealand: Department of Computer Sciences, University of Canterbury, 1995.
- [Barnett 02]** Barnett, Ryan. *Monitoring VMware Honeypots*. http://honeypots.sourceforge.net/monitoring_vmware_honeypots.html (2002).
- [Bauer 05]** Bauer, Michael D. *Linux Server Security, 2nd Edition*. Sebastopol, CA: O'Reilly, 2005.
- [Duda 87]** Duda, A.; Harrus, G.; Haddad, Y.; Bernard, G. "Estimating Global Time in Distributed Systems." *Proceedings of the 7th International Conference on Distributed Computing Systems (ICDCS '87)*. Berlin, Germany, Sept. 1987. Los Alamitos, CA: IEEE Computer Society Press, 1987. <http://www.informatik.uni-trier.de/~ley/db/conf/icdcs/icdcs87.html>.
- [Galleon 04]** Galleon. *GPS Time Server*. <http://www.ntp-time-server.com/gps-time-server/gps-time-server.htm> (2004).
- [Haas 04]** Haas, Juergen. *Linux / Unix Command: checkconfig*. http://linux.about.com/library/cmd/blcmdl8_chkconfig.htm (2004).
- [Lamp 78]** Lamport, L. "Time, Clocks, and the Ordering of Events in a Distributed System." *Communications of the ACM* 21 (1978): 558-565.
- [LP 05]** *Log Parser 2.2 Documentation*. Distributed with *Log Parser 2.2*.
- [Lucke 04]** Lucke, Ken. *Reading Email Headers*. <http://www.stopspam.org/email/headers.html> (2004).
- [Mandia 01]** Mandia, Kevin & Proise, Chris. *Incident Response*. Berkley, California: McGraw-Hill, 2001.

- [McWilliams 05]** McWilliams, Brian. "Hijacked by Spammers." *O'Reilly Network*. <http://www.oreillynet.com/pub/a/network/2005/03/14/spammerhijack.html> (2005).
- [Mills 91]** Mills, D. L. "Internet Time Synchronization: The Network Time Protocol." *IEEE Trans. Communications* 39, 10 (October 1991): 1482-1493.
- [Nolan 05]** Nolan, Richard; O'Sullivan, Colin; Branson, Jake; & Waits, Cal. *First Responders Guide to Computer Forensics* (CMU/SEI-2005-HB-001). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, 2005. <http://www.sei.cmu.edu/publications/documents/05.reports/05hb001.html>.
- [NWG 01]** Network Working Group. *RFC 2821*. <http://www.faqs.org/rfcs/rfc2821.html> (2001).
- [Ristenpart 04]** Ristenpart, Thomas; Templeton, Steven; & Bishop, Matt. "Time Synchronization of Aggregated Heterogeneous Logs." <http://ultimate.cs.ucdavis.edu/SecSemApril04.ppt> (2004).
- [Sendmail 05]** Sendmail, Inc. *Datasheet: Sendmail Outbound Mail Management Solution*. http://www.sendmail.com/pdfs/datasheets/ds_hvms.pdf (2005).
- [SourceForge 04]** SourceForge. *Project Info – Swatch*. <http://sourceforge.net/projects/swatch> (2004).
- [Tan 02]** Tanenbaum, Andrew S. & van Steen, Maarten. *Distributed Systems: Principles and Paradigms*. Singapore: Pearson Education, 2002.
- [ULP 05]** The Unofficial Log Parser Support Site. <http://www.logparser.com/> (2005).
- [Wikipedia 05a]** Wikipedia. *American Registry for Internet Numbers*. <http://en.wikipedia.org/wiki/ARIN> (2005).
- [Wikipedia 05b]** Wikipedia. *Coordinated Universal Time*. <http://en.wikipedia.org/wiki/UTC> (2005).
- [Wikipedia 05c]** Wikipedia. *Global Positioning System*. http://en.wikipedia.org/wiki/Global_positioning_system (2005).
- [Wikipedia 05d]** Wikipedia. *Regular Expressions*. http://en.wikipedia.org/wiki/Regular_expression (2005).

- [Wikipedia 05e]** Wikipedia. *Simple Mail Transfer Protocol*.
http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
(2005).
- [Wikipedia 05f]** Wikipedia. *SQL*. <http://en.wikipedia.org/wiki/SQL> (2005).

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2005	3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE First Responders Guide to Computer Forensics: Advanced Topics		5. FUNDING NUMBERS FA8721-05-C-0003	
6. AUTHOR(S) Richard Nolan, Marie Baker, Jake Branson, Josh Hammerstein, Kris Rush, Cal Waits, Elizabeth Schweinsberg			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-HB-003	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE 169	
13. ABSTRACT (MAXIMUM 200 WORDS) This handbook expands on the technical material presented in SEI handbook CMU/SEI-2005-HB-001, <i>First Responders Guide to Computer Forensics</i> . While the latter presented techniques for forensically sound collection of data and explained the fundamentals of admissibility pertaining to electronic files, this handbook covers more advanced technical operations such as process characterization and spoofed email. It describes advanced methodologies, tools, and procedures for applying computer forensics when performing routine log file reviews, network alert verifications, and other routine interactions with systems and networks. The material will help system and network professionals to safely preserve technical information related to network alerts and other security issues.			
14. SUBJECT TERMS computer forensics, information security, spoofed email, log file analysis, data recovery, computer security incident		15. NUMBER OF PAGES 168	
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL