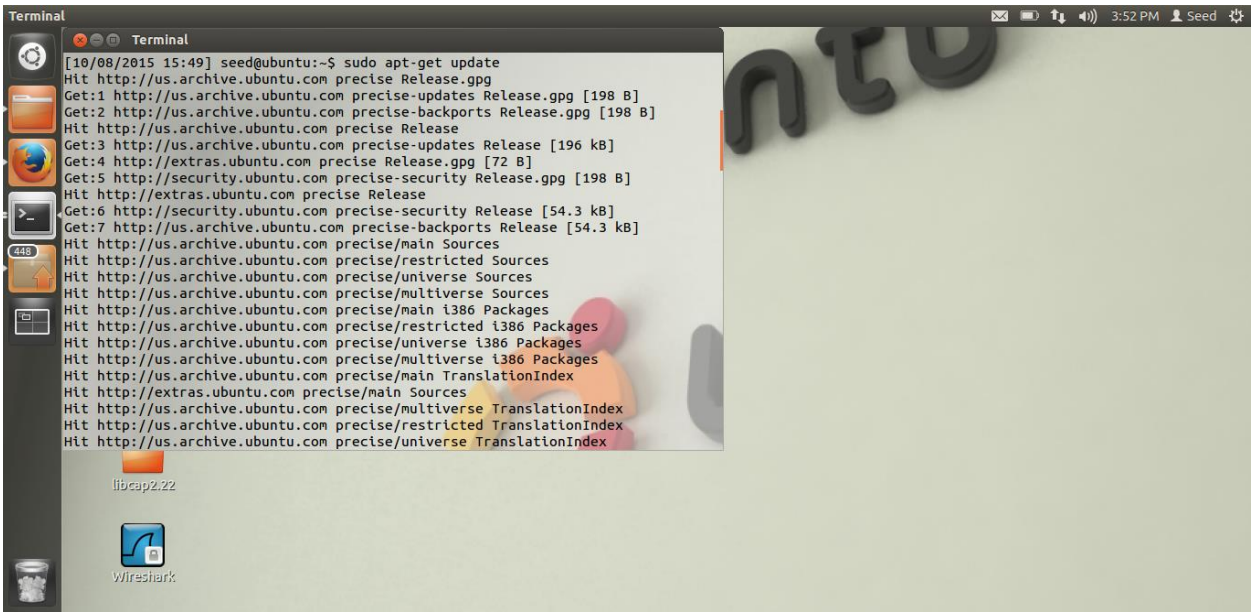
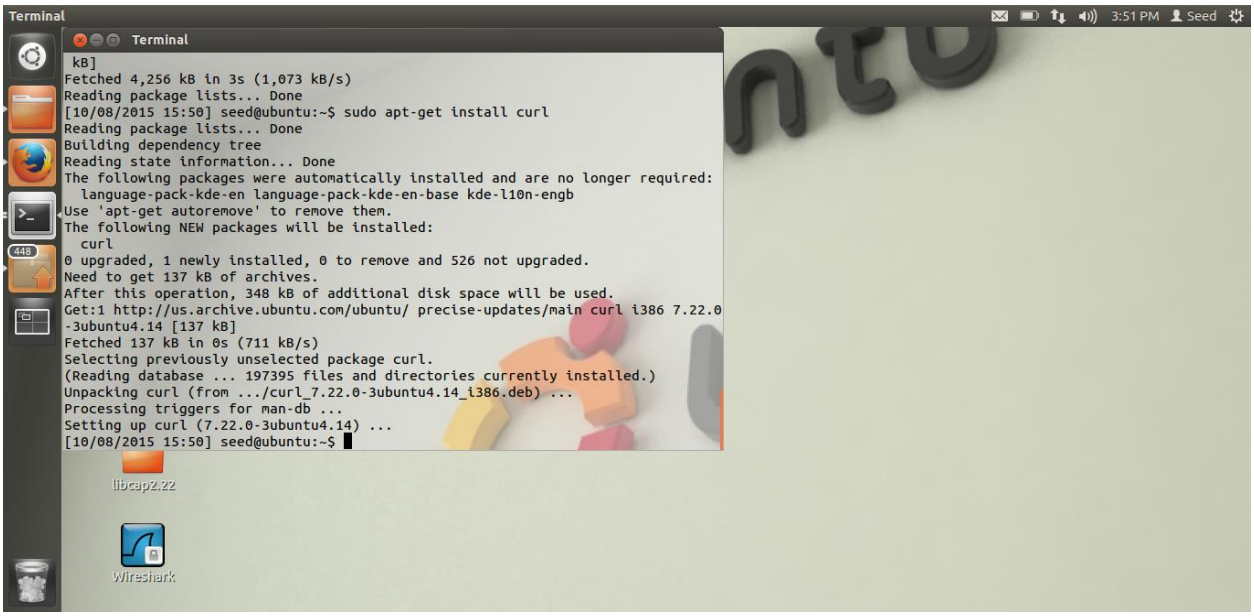
Yogesh Chaudhari
SUID: 244195971

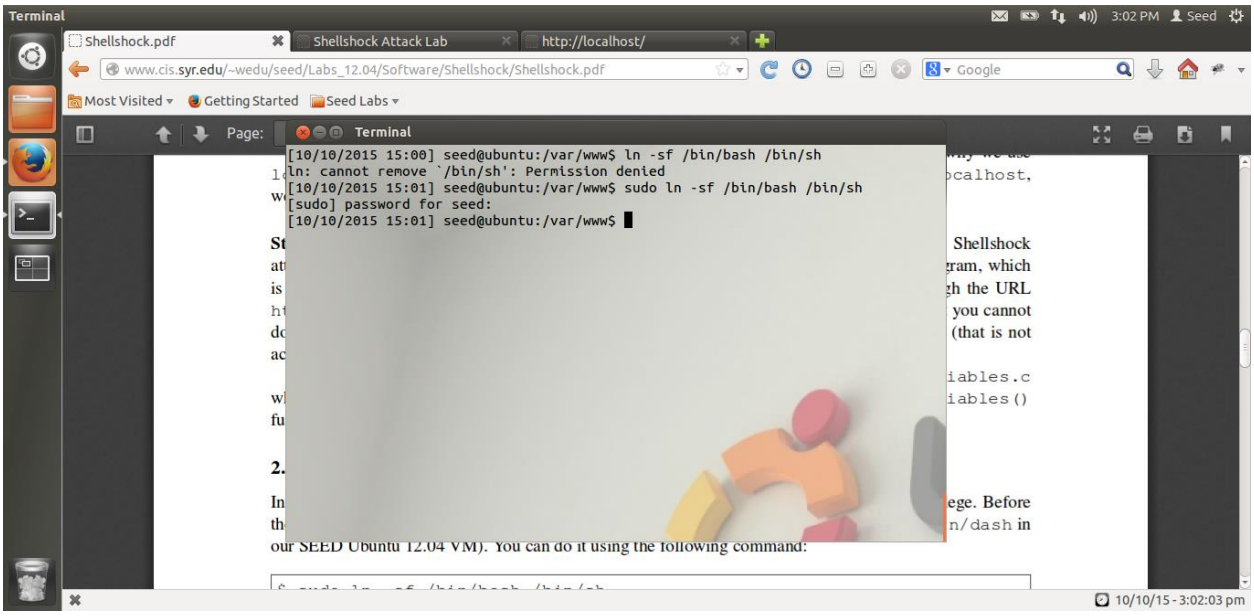
Initial Setup:



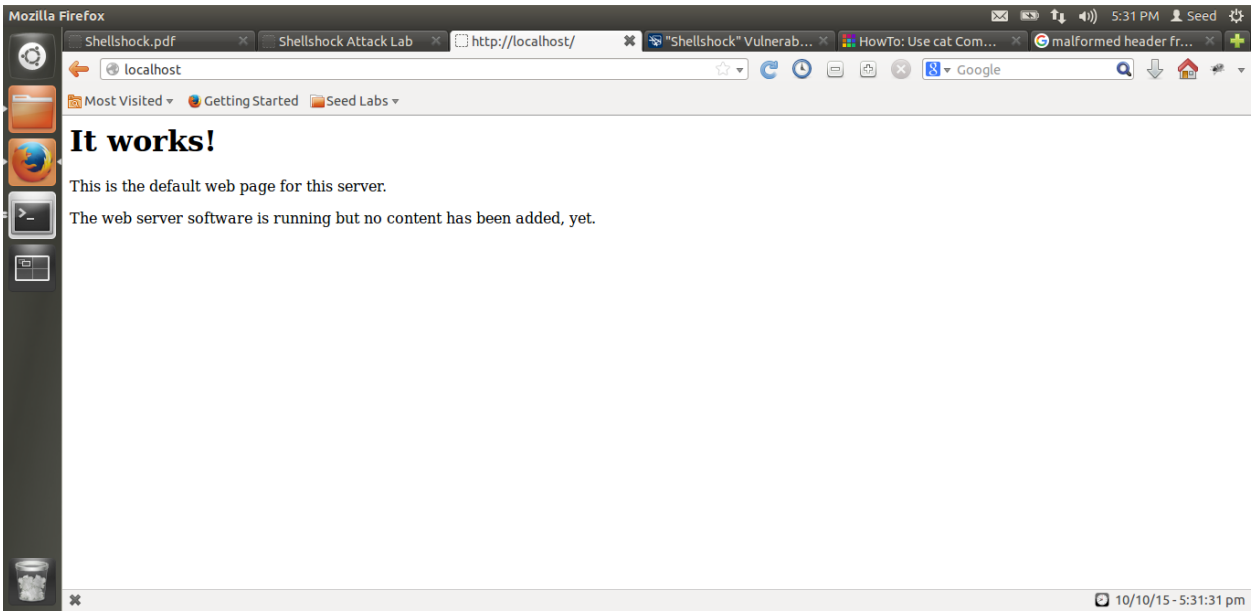


Here we are updating our libraries list for UNIX system using apt-get update command and then we installed curl using apt-get install curl command.

Yogesh Chaudhari
SUID: 244195971



Here we are setting symbolic link from /bin/sh program to /bin/bash program using ln –sf command. So that our shell starts or opens bash shell on default.

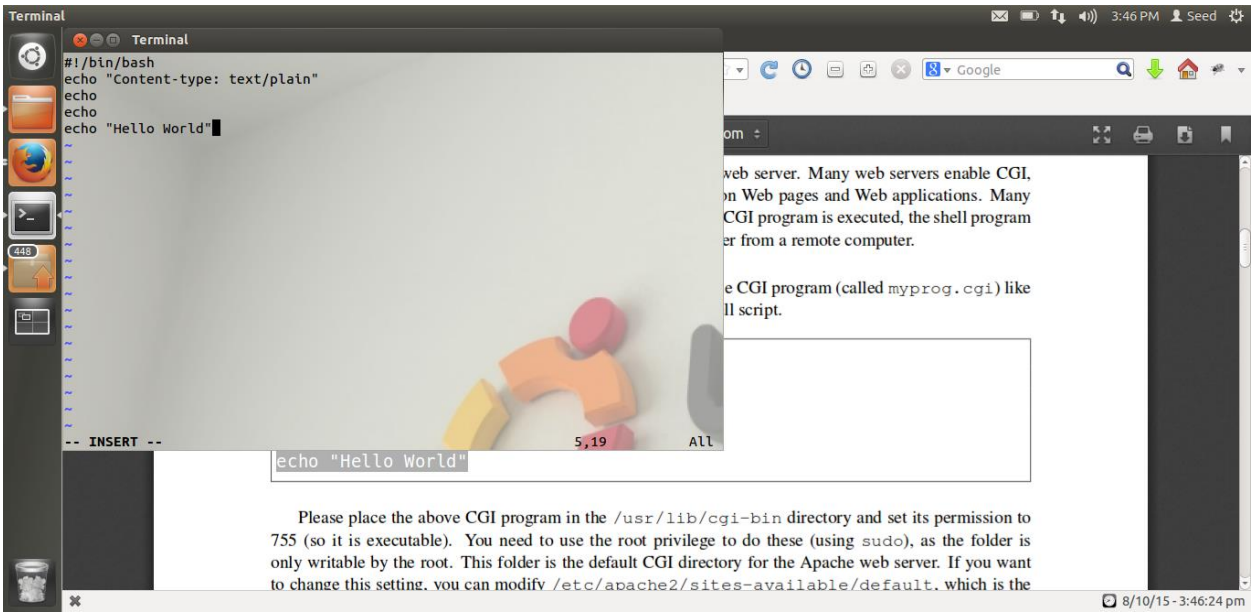


Here we can see that our apache server is working as localhost shows index.html contents.

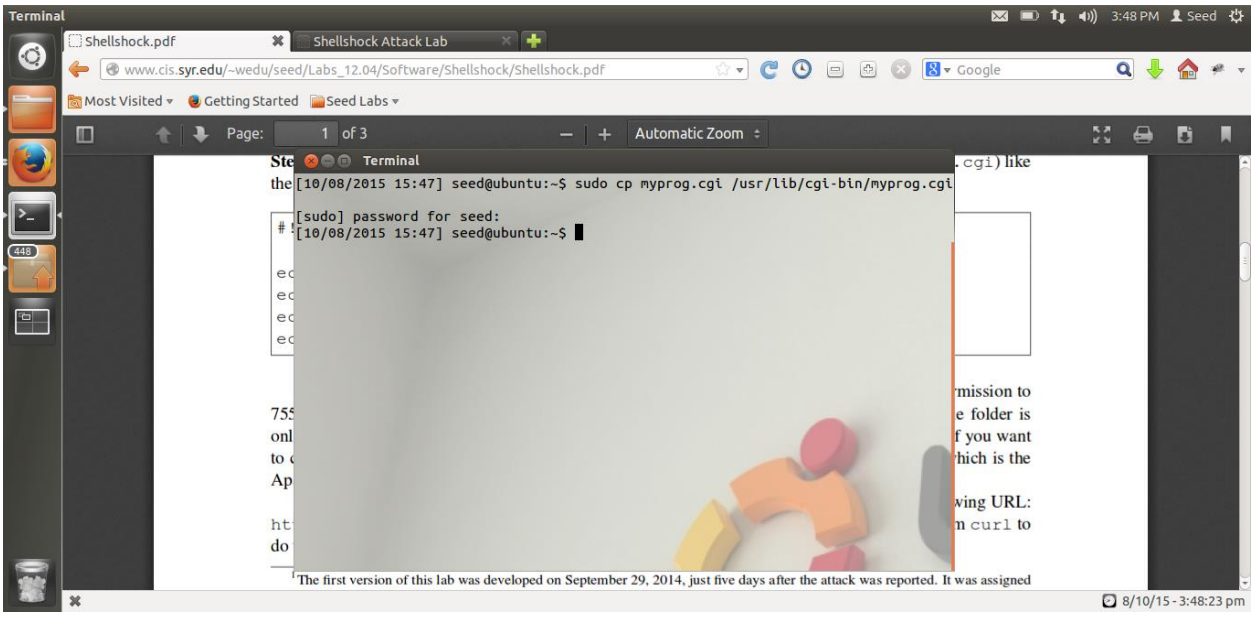
Yogesh Chaudhari
SUID: 244195971

Task1:

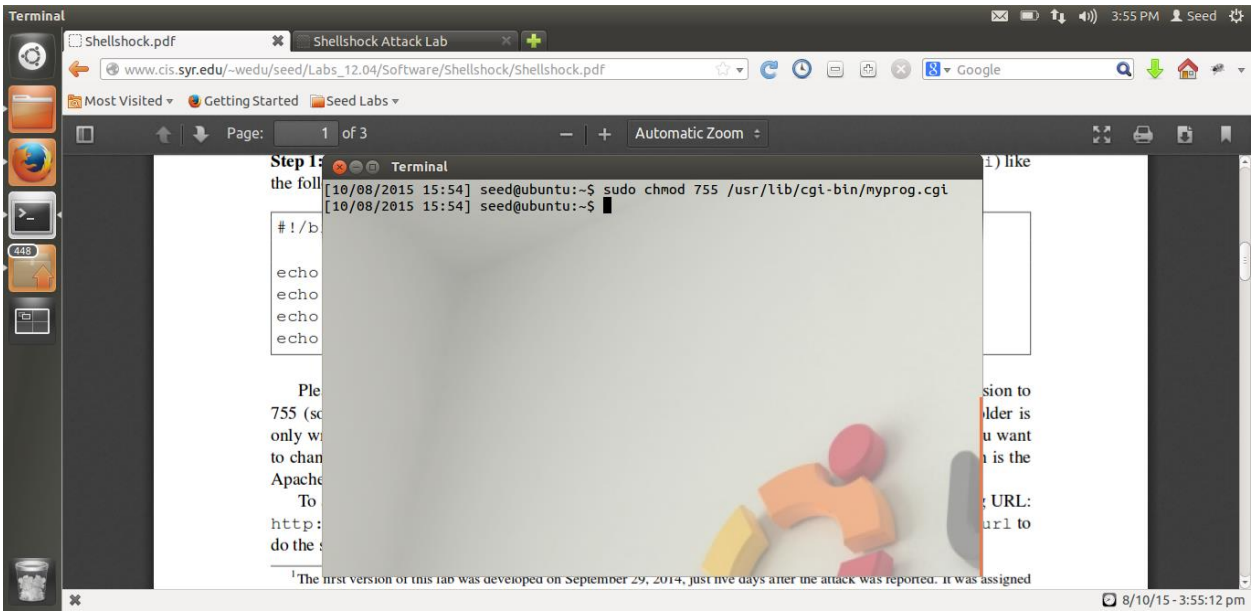
Myprog.cgi:



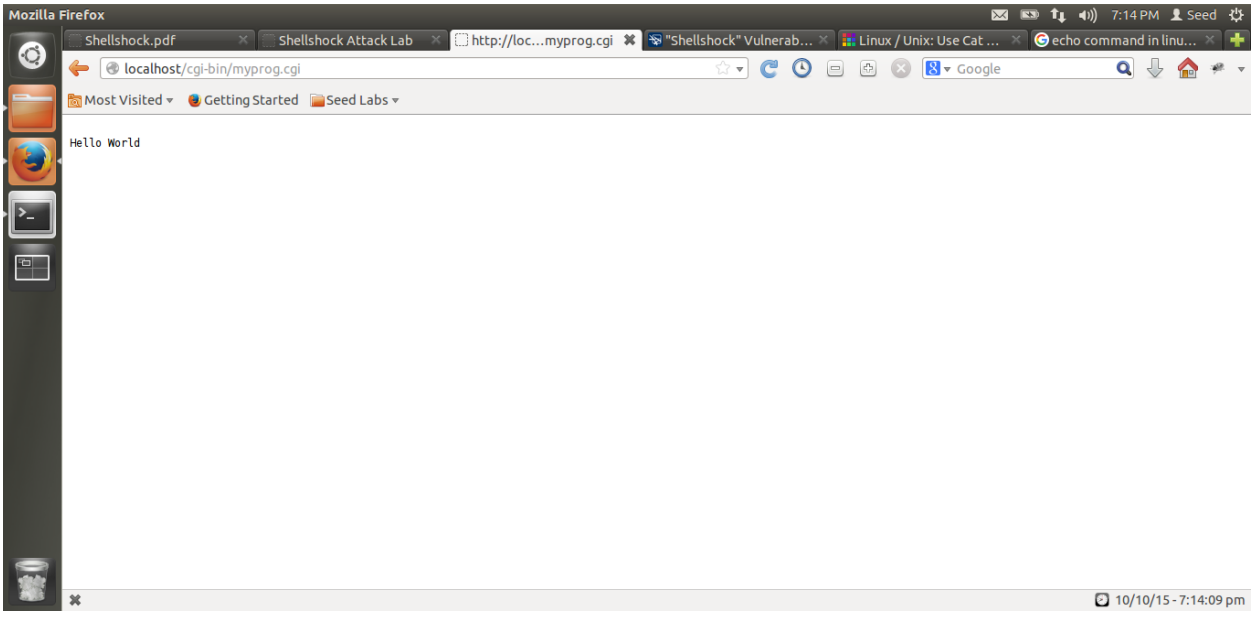
Here we are writing a cgi script myprog.cgi to print hello world in shell when executed.



Yogesh Chaudhari
SUID: 244195971

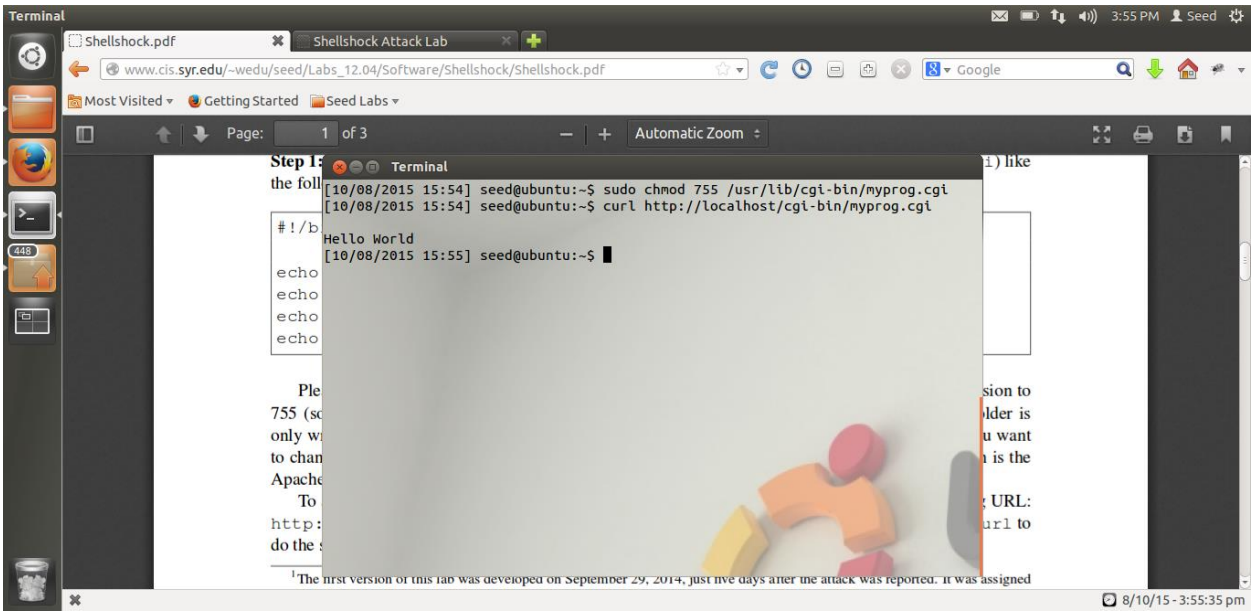


Here we are placing myprog.cgi in /usr/lib/cgi-bin directory, which is a cgi files directory for apache server. Then using chmod 755 command we are making myprog an executable file. For all this operations we are using sudo command because /usr/lib/cgi-bin is a root owned directory. So we need root priviledges to write into thois directory. Hence, we can call our myprog.cgi with apache server using http://localhost/cgi-bin/myprog.cgi URL in browser and apache server will run myprog.cgi to generate the output.

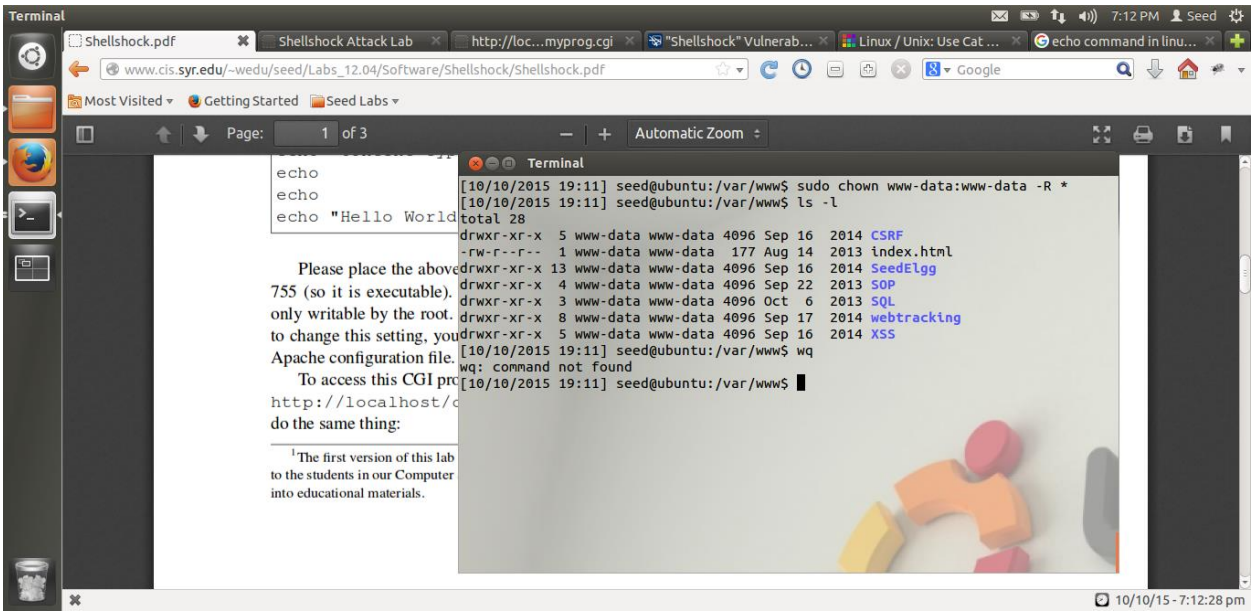


Here we can see output of myprog.cgi in browser, when its executed.

Yogesh Chaudhari
SUID: 244195971

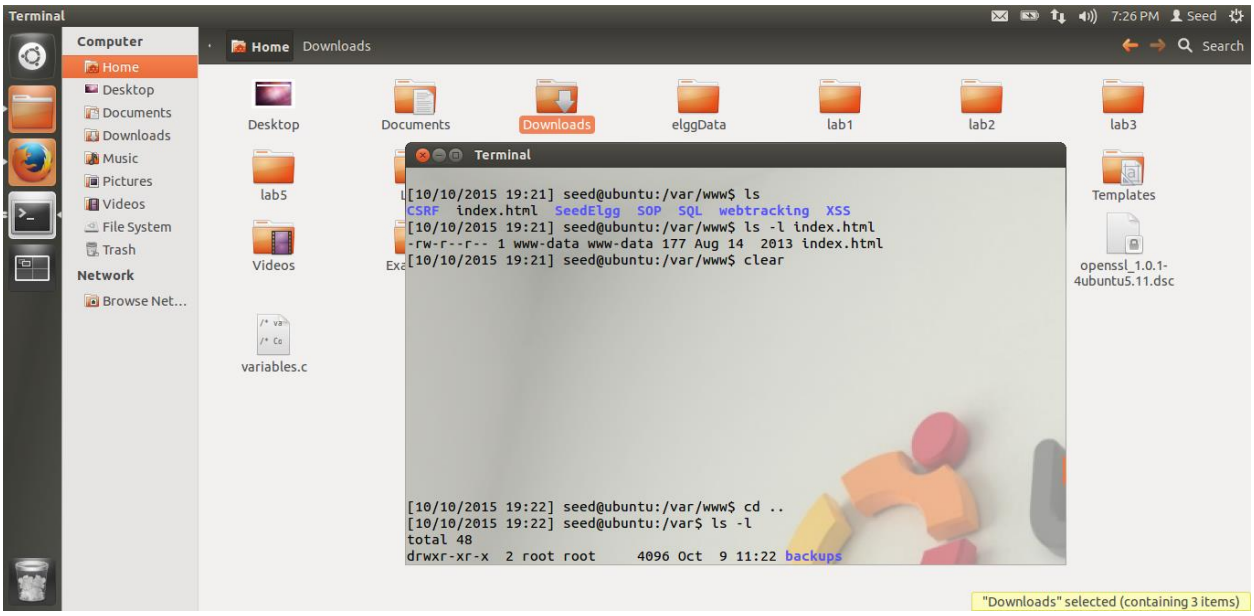


Using curl command we can see output of cgi file on apache server in shell terminal, as it executes that cgi file in shell opened by apache server.

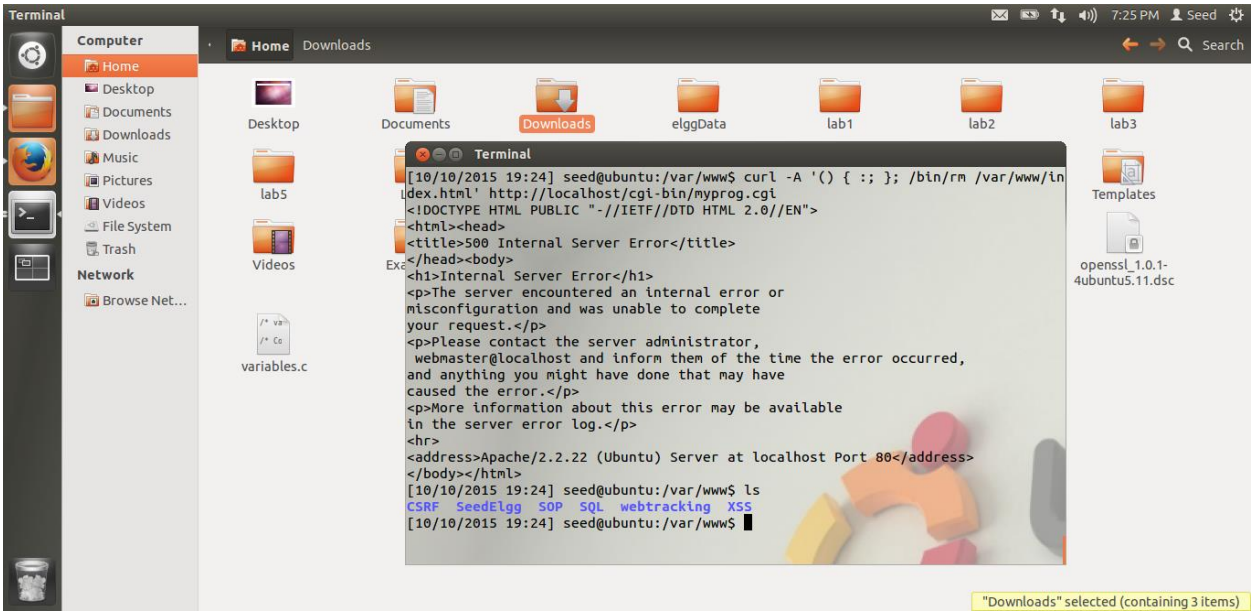


Now, we are making all files in /var/www directory to be owned by www-data using chown command. We have also changed owner of /var/www directory to www-data using chown command. We need to do this becase using shell openend by apache server we can only modify data owned by www-data user.

Yogesh Chaudhari
SUID: 244195971

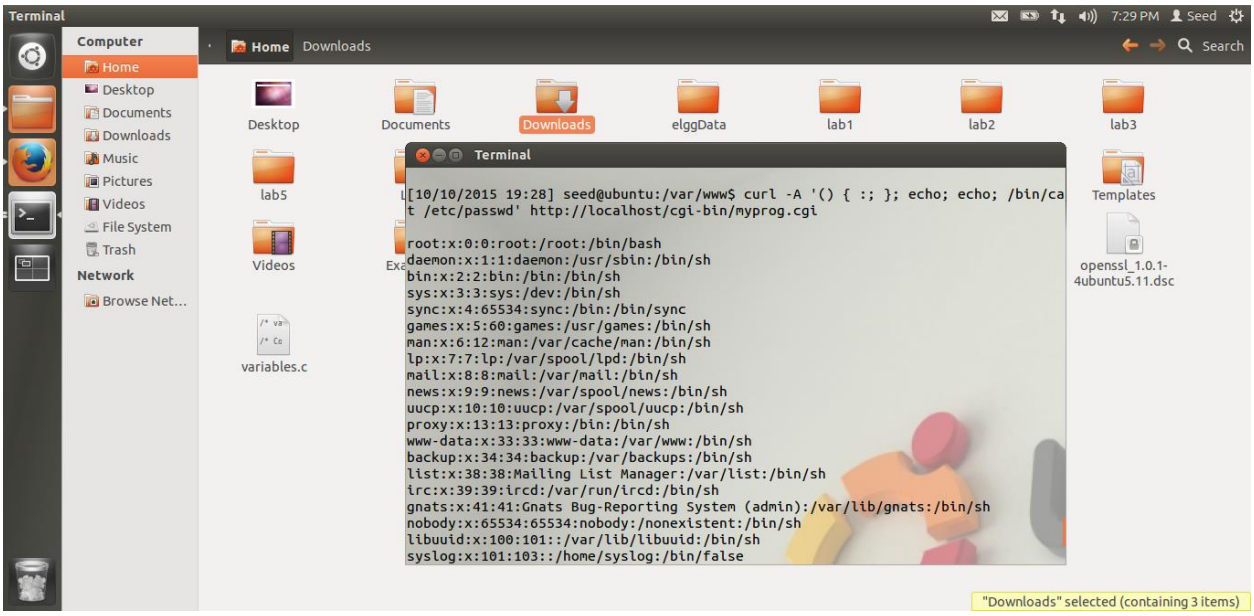


We can see all the contents of /var/www directory before we perform our shellshock attack.



Here we can see that our shellshock attack was successful and index.html was removed from /var/www directory. For this we are using curl –A '() { :; }; bad/attack command' URL command. –A option with curl defines String contents to be passed to environment of user-agent when executing cgi script on apache server. Here we are using shellshock vulnerability to delete some files which are owned by apache server.

Yogesh Chaudhari
SUID: 244195971



Here we are using shellshock vulnerability of apache server to display contents of /etc/passwd file. This file is owned by root.

In variable.c we can see that the vulnerability is at

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

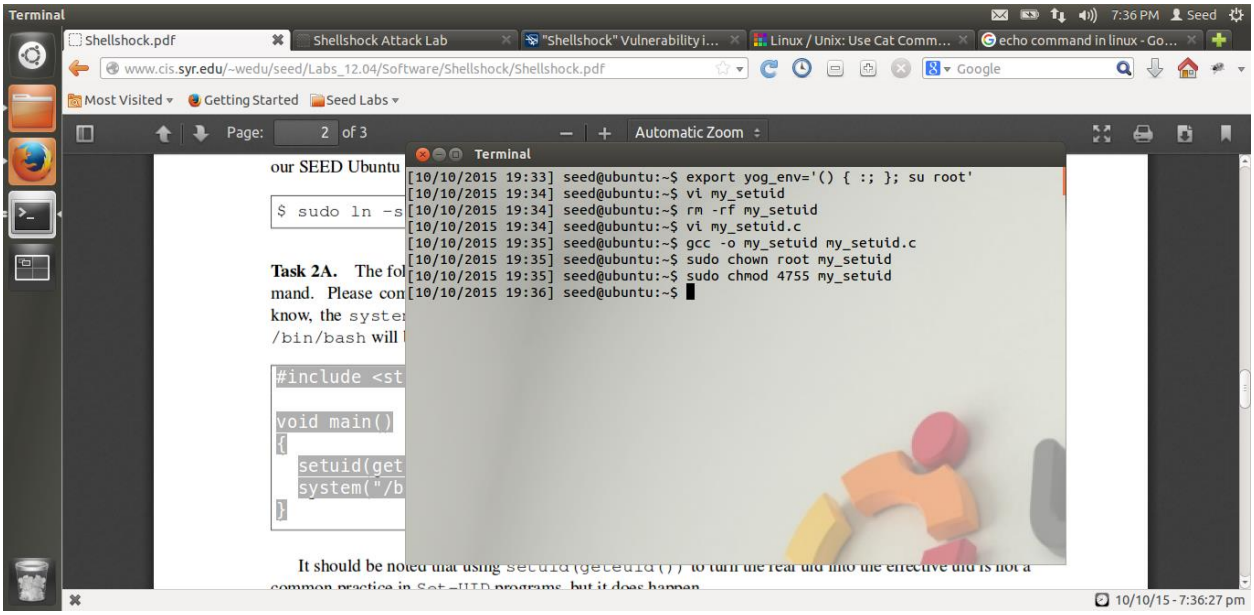
here in if condition while providing string as an input to STREQN program doesnot check for any closing brackets or function termination syntaxes or calls while converting given equation/function value to String, hence our malicious code gets into the function and with specific syntax structure we can actually make parser think of our malicious code as an command to run.

Line 349: strcpy (temp_string + char_index + 1, string);

In this line all the contents given as env_variable value are put into temp_string without doing boundary check or content check. And then these are passed to function parse_and_execute(), so our attack line which got copied in temp_string due to strcpy() function got parsed and excuted. This is how shellshock attack works.

Yogesh Chaudhari
SUID: 244195971

Task 2a:



Now we are setting an environment variable yog_env with value “(){ :; }; su root” which exploits shellshock vulnerability of system and if exploited correctly gives us root shell.

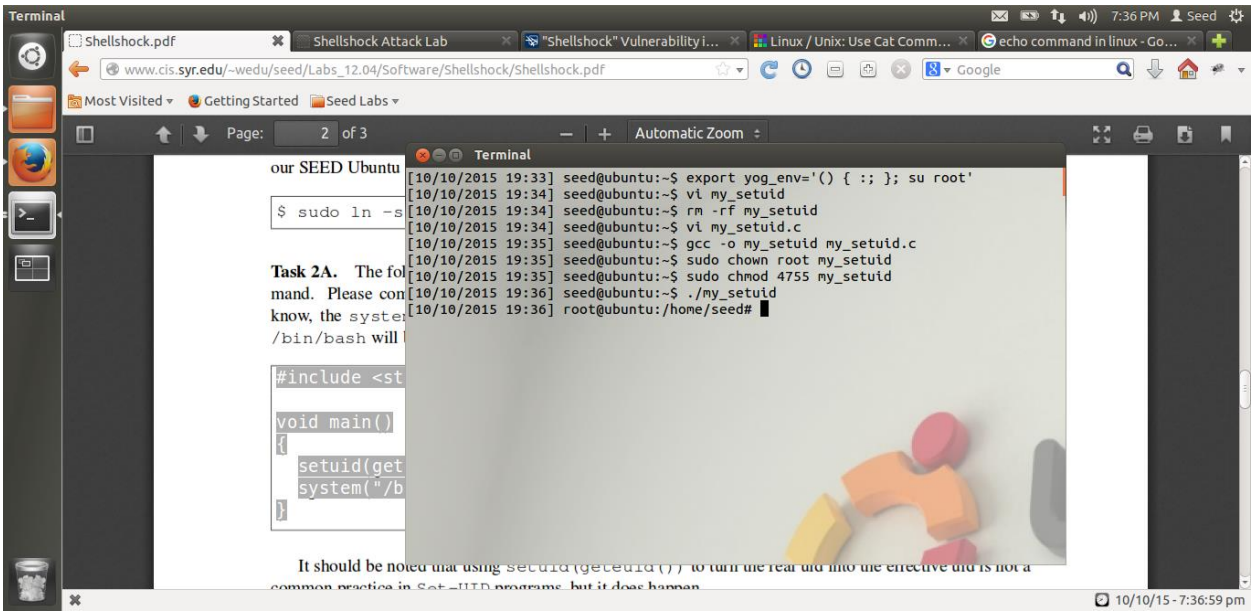
my_set-uid.c:

```
#include <stdio.h>

void main()

{

setuid(geteuid()); // make real uid = effective uid.

system("/bin/ls -l");

}
```

above program has system() call which opens a new bash shell when this program is executed to executed the command given in system() call.

Here we are setting userid of user equal to effective user-id of user i.e. root.

Yogesh Chaudhari
SUID: 244195971

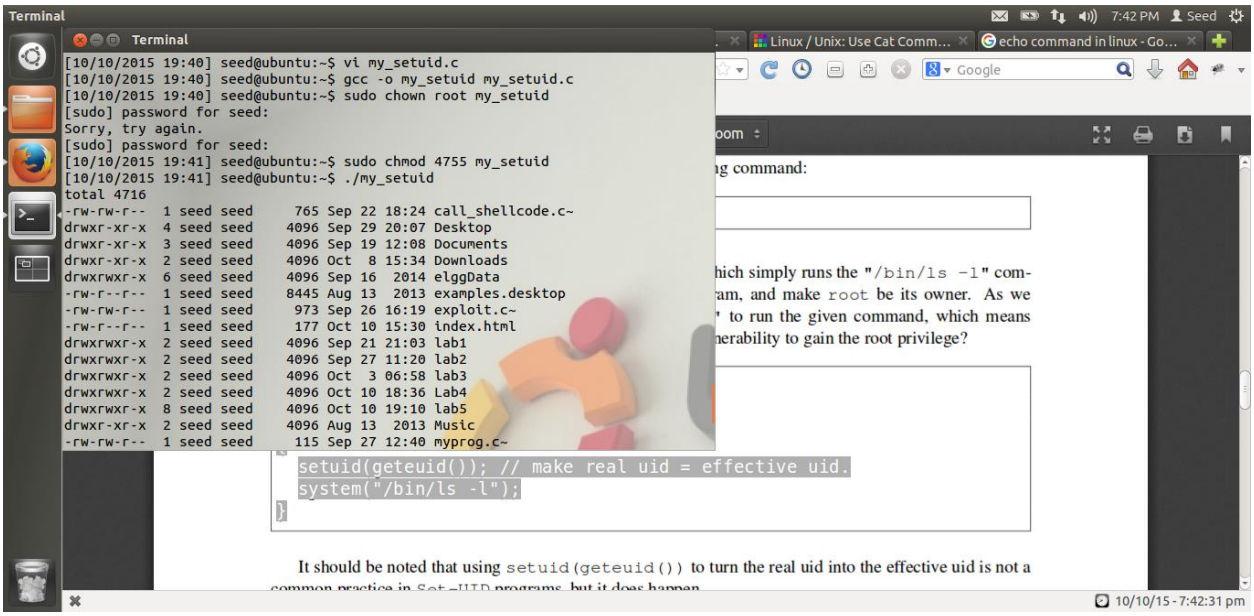


Here we are compiling our my_setuid.c program and saving compiled data to my_setuid file. Now, we are making my_setuid file set-uid root program by using chown root and chmod 4755 command.

Now, after executing my_setuid program we get root shell and hence our shellshock attack was successful. When new bash was opened system was setting up all of its environment variables in that it executed our injected code in yog_env variable and changed current user to root.

Yogesh Chaudhari
SUID: 244195971

Task 2b:



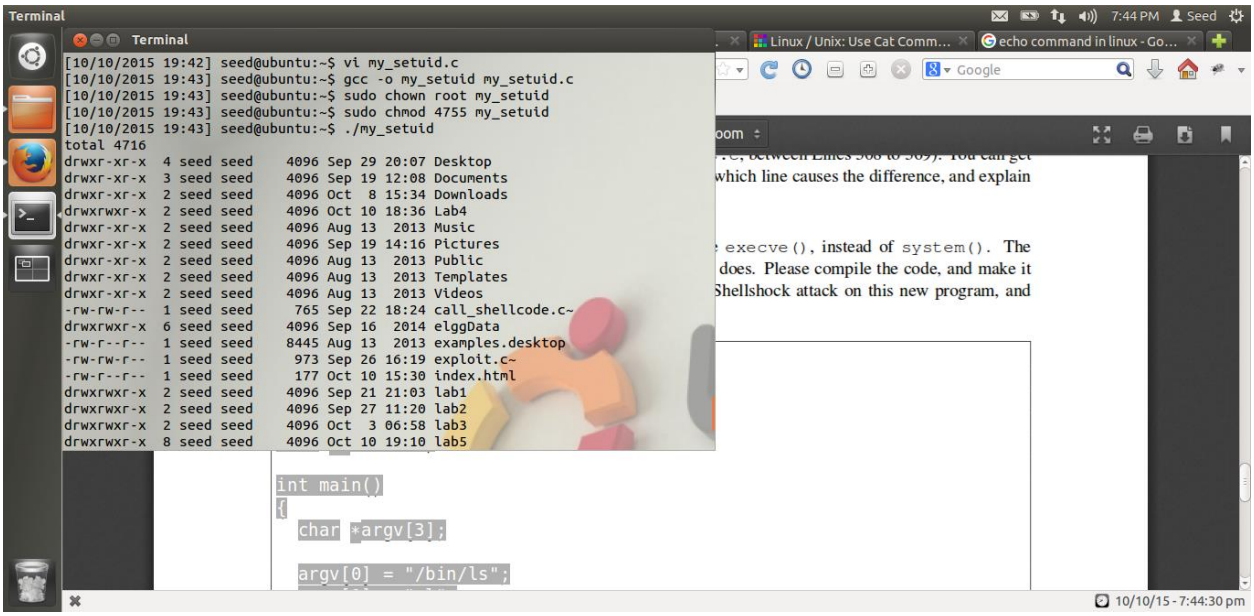
my_set-uid.c:

```
#include <stdio.h>

void main()

{

//setuid(geteuid()); // make real uid = effective uid.

system("/bin/ls -l");

}
```

In this case we have removed setuid(geteuid()); command from our program and again compiled the program and made the compiled file set-uid root program using chown root and chmod 4755 command. After executing above program we can see that our attack was not successful. In this case also, system() opened a new bash shell to execute command in system() call. It executed our injected code su root but was not able to change user to root as this shell doesnot have root privileges but it has seed user privileges. When a bash shell is invoked it always checks for user's real UID rather than effective UID, and invokes a shell with user UID privileges. Hence, the new opened shell was not able to call su root command and our attack was not successful. It actually displayed all files present in seed user home directory.

Yogesh Chaudhari
SUID: 244195971

Task 2c:



my_set-uid.c:

```
#include <string.h>

#include <stdio.h>

#include <stdlib.h>

char **environ;

int main()

{

char *argv[3];

argv[0] = "/bin/ls";

argv[1] = "-l";

argv[2] = NULL;

setuid(geteuid()); // make real uid = effective uid.

execve(argv[0], argv, environ);

return 0 ;

}
```

Here now we have replaced system() call with execve() call. After executing the program we can see that attack was not successful. This is because when we use system() call, it starts a new bash shell to execute the program and parses all the contents passed to it within the environment variables, but in case of execve() call shell doesnot parse any of environment variables passed to it in string format. Hence, our environment variable was not parsed and executed the attack command in it. So, attack was unsuccessful.

Task3:

2.

The fundamental issue with shellshock is that bash automatically imports functions for environment variables. For this bash saves all environment variables in temp_string and then passes it to parse_and_execute() function for importing and executing functions for environment variables.

Shellshock vulnerability is a design issue, because the developers used strcpy() function to copy value of env_variable into temp_string before passing it to parse_and_execute() function we get this vulnerability. strcpy() is a function which doesnot check for boundaries of the given input as well as validate the given input before copying it, our injected malicious code gets copied into the temp_string. And by using specific type of code structure while providing this env_variable value we can fool system to execute our code using parse_and_execute() function.

From developers point of view after seeing this vulnerability we can understand the importance of always validating all inputs and using functions which always perform validation and boundary check operations on all input values before actually using them to process.

1.

We can do this attack on oopenSSH server. All git servers actually use openSSH as there main program to connect with other git clients. In this case, when a client wants to connect with git server it actually has to pass its public ssh key (generated using ssh keygen in rsa format). Then server stores clients public key into its authorized keys folder. And while storing this key on server we use cat command for copying client's public key into server's authorized keys files. When client tries to connect with the server, server actually opens a new shell and checks if clients current public key matches with any of the public key value present in its authorized keys file. If yes then it gives client the access to the git repository it wants to, if not it will again asks for permission and client's public key. Now, before client passes its public key to server if we wrote a command in client's public key file (syntax: command=bad command) which we want to execute at server side, we can run this command on git server when server tries to check for public key in its authorized keys file. Thus, we can attack a git server using shellshock attack.