CSE3501-Information Security Analysis and Audit

Lab 9+10

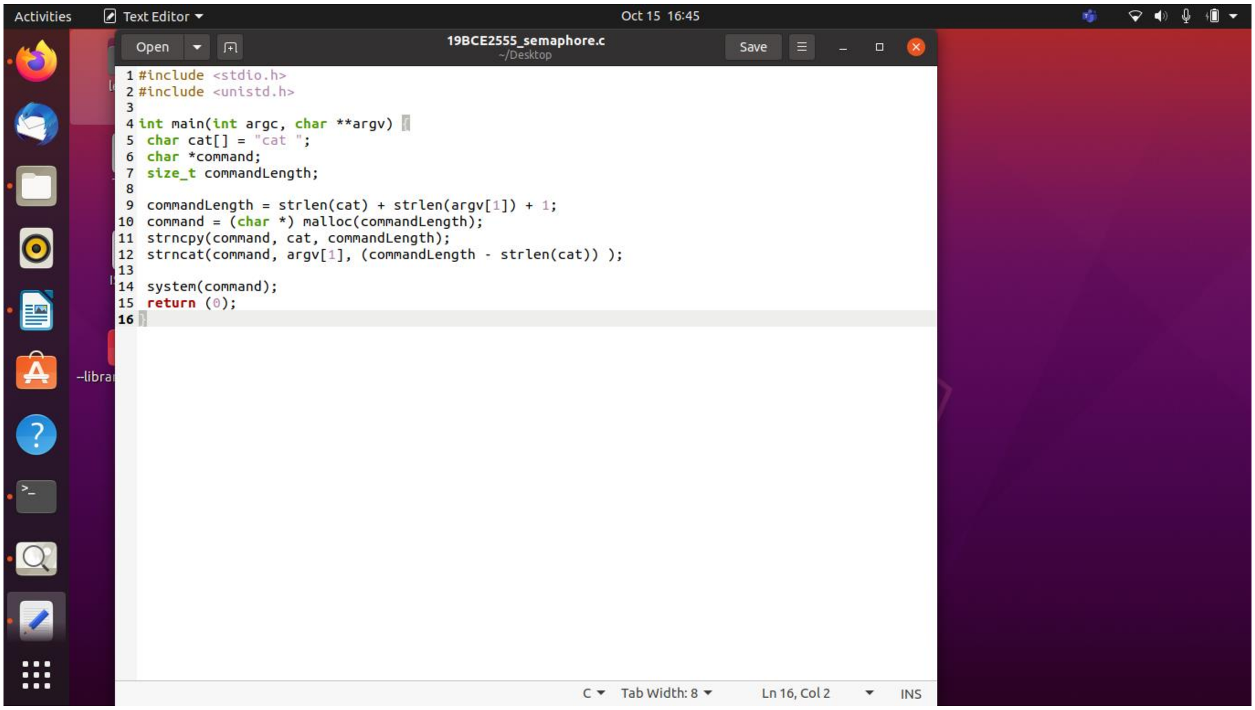
Digital Assignment-5

Submitted by: Alokam Nikhitha

Reg No:19BCE2555

Exercise:

Command Injection:

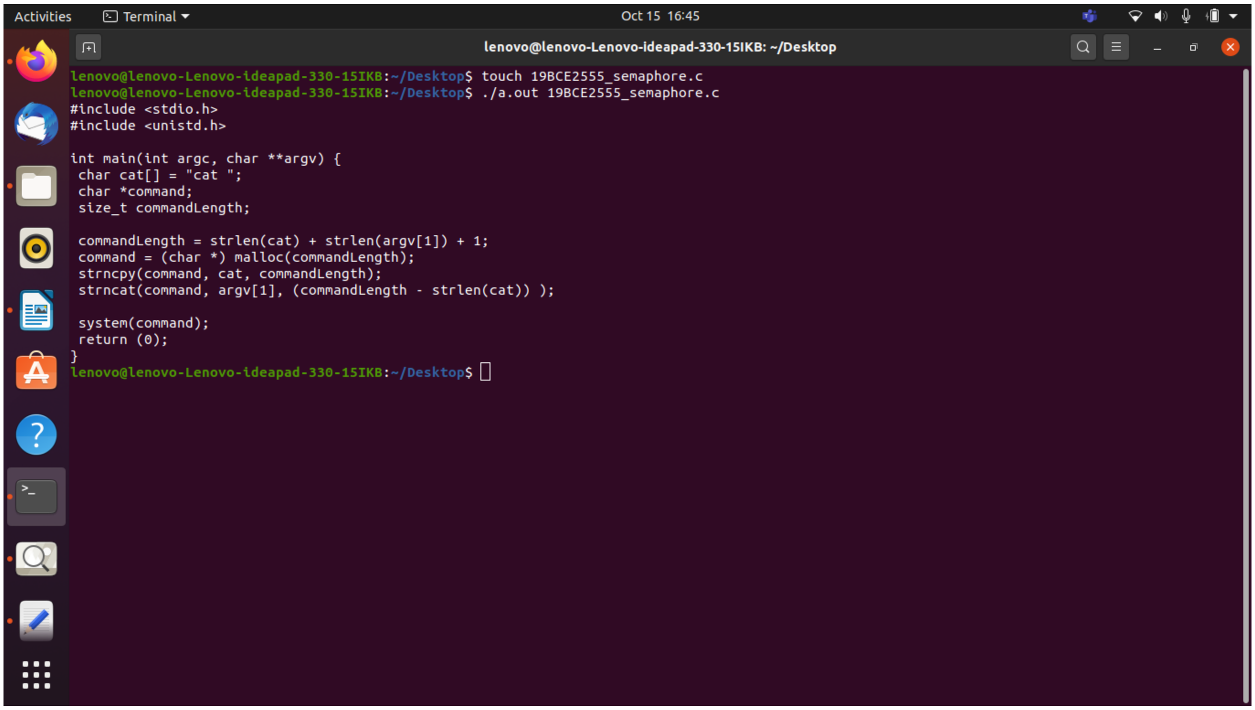


```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
```

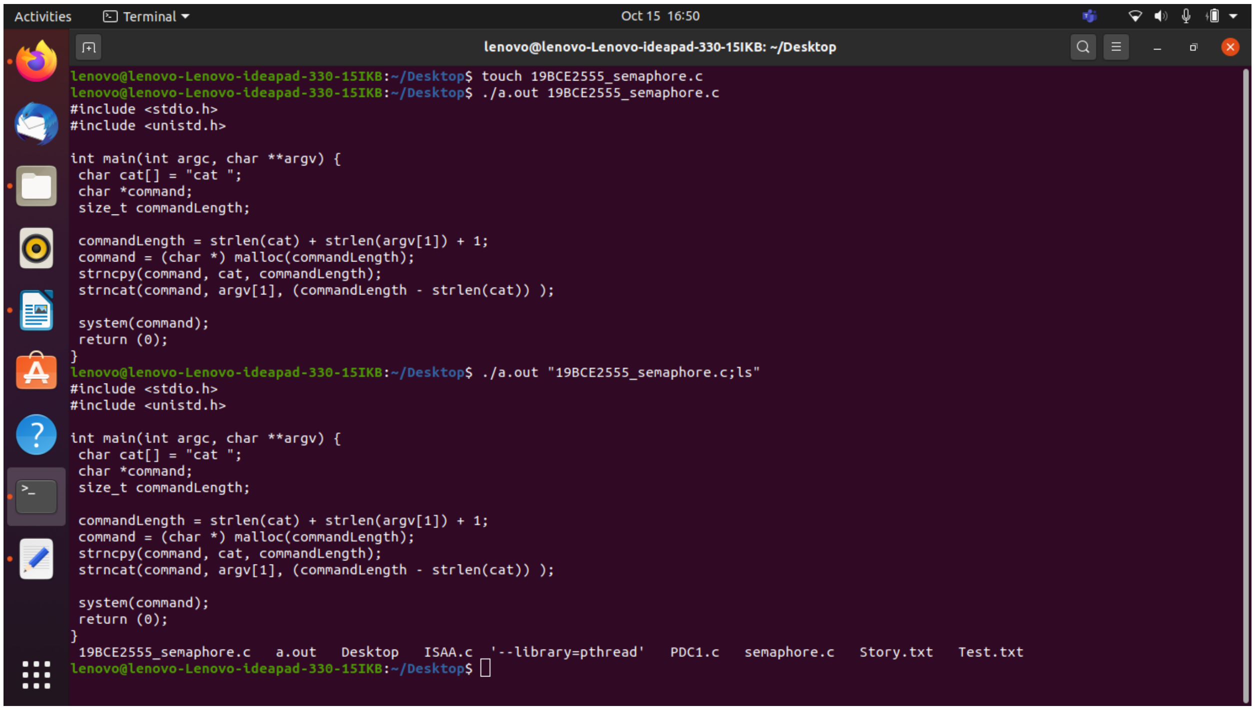


```
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ touch 19BCE2555_semaphore.c
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ ./a.out 19BCE2555_semaphore.c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ 
```



```
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ touch 19BCE2555_semaphore.c
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ ./a.out 19BCE2555_semaphore.c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ ./a.out "19BCE2555_semaphore.c;ls"
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
 19BCE2555_semaphore.c   a.out   Desktop   ISAA.c  '--library=pthread'   PDC1.c   semaphore.c   Story.txt   Test.txt
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ 
```

Defensive Mechanism against the following attacks

1. Avoid calling OS commands directly

The primary defense is to avoid calling OS commands directly. Built-in library functions are a very good alternative to OS Commands, as they cannot be manipulated to perform tasks other than those it is intended to do.

For example use mkdir() instead of system("mkdir /dir_name").

If there are available libraries or APIs for the language you use, this is the preferred method.

2. Escape values added to OS commands specific to each OS

TODO: To enhance.

For examples, see escapeshellarg() or escapeshellcmd() in PHP.

escapeshellarg()

escapeshellarg(string $arg): string

escapeshellarg() adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument. This function should be used to escape individual arguments to shell functions coming from user input. The shell functions include exec(), system() and the backtick operator.

On Windows, escapeshellarg() instead replaces percent signs, exclamation marks (delayed variable substitution) and double quotes with spaces and adds double quotes around the string.

```
<?php
system('ls '.escapeshellarg($dir));
?>
```

escapeshellcmd()

escapeshellcmd(string $command): string

escapeshellcmd() escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands. This function should be used to make sure that any data coming from user input is escaped before this data is passed to the exec() or system() functions, or to the backtick operator.

Following characters are preceded by a backslash: &#;`|*?~<>^()[]{}$\, \x0A and \xFF. ' and " are escaped only if they are not paired. On Windows, all these characters plus % and ! are preceded by a caret (^).

```
<?php
// We allow arbitrary number of arguments intentionally here.
$command = './configure '.$_POST['configure_options'];

$escaped_command = escapeshellcmd($command);

system($escaped_command);
?>
```

3. Insertion of special characters

We can inject some special characters to see if the application blocks anything that could be used for command injection:

&

;

Newline (0x0a or \n)

&&

|

||

In case the application doesn't throw any error messages, we can try injecting our command after using one of these delimiters.

https://vulnerable-website/endpoint?parameter=1|whoami

Where a list of good, allowed characters and the maximum length of the string are defined. Ensure that metacharacters like ones specified in Note A and white-spaces are not part of the Regular Expression. For example, the following regular expression only allows lowercase letters and numbers and does not contain metacharacters. The length is also being limited to 3-10 characters:

```
^[a-z0-9]{3,10}$
```

4. Time delays

The time delay exploitation technique is very useful when the tester find a Blind SQL Injection situation, in which nothing is known on the outcome of an operation. This technique consists in sending an injected query and in case the conditional is true, the tester can monitor the time taken to for the server to respond. If there is a delay, the tester can assume the result of the conditional query is true. This exploitation technique can be different from DBMS to DBMS (check DBMS specific section).

```
http://www.example.com/product.php?id=10 AND IF(version() like '5%', sleep(10), 'false'))--
```

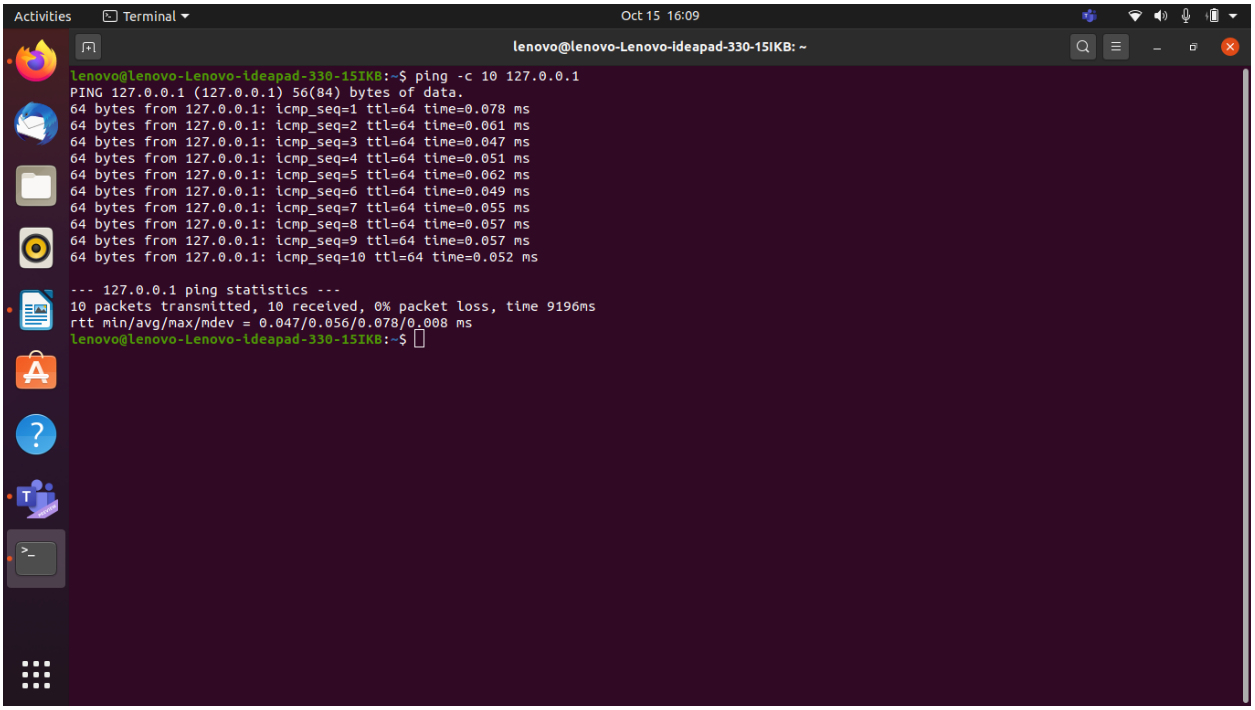
In this example the tester is checking whether the MySql version is 5.x or not, making the server delay the answer by 10 seconds. The tester can increase the delay time and monitor the responses. The tester also doesn't need to wait for the response. Sometimes they can set a very high value (e.g. 100) and cancel the request after some seconds.

Most of the OS command injections are Blind, which doesn’t give any output for the executed command. To verify the vulnerability, after detecting allowed special characters, we can verify the command injection using time delays as below:

```
https://vulnerable-website/endpoint?parameter=x||ping+-c+10+127.0.0.1||
```

Time Delay Commands

```
& ping -c 10 127.0.0.1 &
```



```
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~$ ping -c 10 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.078 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.052 ms

--- 127.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9196ms
rtt min/avg/max/mdev = 0.047/0.056/0.078/0.008 ms
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~$
```

5. Redirecting output

You can also redirect the output of the command in an output file and then retrieve the file on your browser. A payload similar to the following can be used:

https://vulnerable-website/endpoint?

parameter=||whoami>/var/www/images/output.txt||

Redirecting output
& whoami > /var/www/images/output.txt &