CS4351: Computer Security **Instructor: Dr. Deepak Tosh**
Fall 2023 **Due Date:** 10/15/2022, Midnight

ASSIGNMENT# 2: Buffer Overflow Vulnerability

INTRODUCTION:

In this lab, students will be given a program with a buffer-overflow vulnerability; their task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that have been implemented in the operating system to counter against buffer-overflow attacks. Students need to evaluate whether the schemes work or not and explain why.

Suggested Environment:
The suggested environment to conduct this attack is SEED Virtual Machine. Please check the syllabus for download link and setup instructions. Once the VM is successfully set, you have to disable "address space randomization" using the following command. (*sudo sysctl -w kernel.randomize_va_space=0*) for conducting this attack.

Task# 1: Exploiting Buffer Overflow [65%]

You are provided sample C program named "stack.c", which has a buffer-overflow vulnerability. Your job is to exploit this vulnerability and gain the root privilege.

1. **Make this program** a **root-owned Set-UID** program so that if a normal user can exploit this buffer overflow vulnerability, he might be able to get a root shell.

It should be noted that the program gets its input from a file called ***badfile***. This file is under users' control. Now, your objective is to create the contents for ***badfile***, such that when the vulnerable program copies the contents into its buffer, a root shell can be spawned.

Instructions to compile "stack.c":

To compile the above vulnerable program, do not forget to turn off the StackGuard and the non-executable stack protections using the **-fno-stack-protector** and "**-z execstack**" options. Here is the command to run. Ensure to set N= 15 + last 2-digits of your UTEP 800 number.

```
$ gcc -DBUF_SIZE=N -m32 -z execstack -fno-stack-protector -g -o stack
stack.c
```

Do not forget to make this stack program a root-owned Set-UID

2. You are given with a python program (*exploit.py*) that constructs the ***badfile*** for you that has shellcode already built-in. But a few items need to be figured out to successfully create the ***badfile***. You need to read through the code and develop the missing pieces.

Exploiting buffer overflow:

Please run your vulnerable program (*exploit.py*) first. Please note that the program *exploit.py*, which generates the ***badfile***, can be compiled with the default *StackGuard* protection enabled. This is because we are not going to overflow the buffer in this program. We will be overflowing the buffer in *stack.c*, which is compiled with the StackGuard protection disabled.

```
$ python3 exploit.py    // This will create the badfile
```

```
$./stack          // launch the attack by running the vulnerable program
# <---- Bingo! You've got a root shell!
```

What to submit:

a. Both *exploit.py* and *stack.c* programs
b. Report your outputs via taking screenshots of each step.
c. Report the screenshot if you were able to get the root shell. If not, describe the reason.

Task# 2: Defeating Address Randomization [25%]

On 32-bit Linux machines, stacks only have 19 bits of entropy, which means the stack base address can have 2^{19} = 524, 288 possibilities. This number is not that high and can be exhausted easily with the brute-force approach. In this task, we use such an approach to defeat the address randomization countermeasure on our 32-bit VM. First, we **turn-on** the Ubuntu's address randomization using the command: *sudo /sbin/sysctl -w kernel.randomize_va_space=2*.
Then we run the same attack developed in Task 1.

Use a brute-force approach to attack the vulnerable program repeatedly, hoping that the address we put in the ***badfile*** can eventually be correct. You can use the following **shell script** to run the vulnerable program in an infinite loop. If your attack succeeds, the script will stop; otherwise, it will keep running. Please be patient, as this may take a while. Let it run overnight if needed. Please describe your observation.

```
#!/bin/bash
SECONDS=0
value=0
while [ 1 ]
  do
  value=$(( $value + 1 ))
  duration=$SECONDS
  min=$(($duration / 60))
  sec=$(($duration % 60))
  echo "$min minutes and $sec seconds elapsed."
  echo "The program has been running $value times so far."
  ./stack
done
echo Hello World
```

Report to submit [10%]:

- Report if you were able to get the root shell by repeatedly running along with how long it took to exploit.

Notes on Report Submission:
You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.

Grading:

The assignment will be graded based on

(1) Clarity of the report,
(2) Correctness of programs, and
(3) Demonstration of evidences

Note: If needed, the student might have to demonstrate the live attack in front of the instructor.

Submission:

Please submit your reports in PDF format and codes in ZIP file on **BLACKBOARD only**. Make sure to answer the questions in order and CLEARLY STATE YOUR ASSUMPTIONS, if you are unsure about something.

If you have any questions, comments, concerns, doubts, or confusions, please stop by my office to clarify. You can also email me on dktosh@utep.edu. I will be very happy to help.