Buffer Overflow Workout

Prerequisites

- Programming skills in a high-level language like C++ or Python recommended
- Familiarity with basic Linux commands

Introduction

Welcome to the buffer overflow workout where you will be given a high level overview of exploiting the buffer area in a program. Any program, no matter how well programmed is bound to have a small bug somewhere in its code. These little bugs in the program are called **vulnerabilities.** Taking advantage of a vulnerability to breach a system or gain unauthorized access is called **an exploit.** Some security researchers take the time finding the security holes in a program and then reporting them before attackers use them to exploit systems.

First off, what's a buffer?

A buffer or memory buffer is an area in computer memory or RAM that temporarily stores data. It usually has a fixed-length defined in the code.

Where are buffers used in real programs?

Buffers are usually used when you have to enter information into a program. For instance, login credentials or user input fields like search bars typically use buffers.

What is a buffer overflow?

A buffer overflow is an attack that allows a user to write data outside of the buffer area. This can lead to a user overwriting other parts of a program and then inserting their own code.

How does a buffer overflow happen behind the scenes?

Let's say some user input with a maximum of 10 bytes is required. If you put in 16 bytes, 6 of those bytes will exceed the buffer causing an overflow. This may cause the program to crash or return corrupt info. Sometimes, an attacker can use this vulnerability to write their own code since it overwrites other instructions.

How do I find a buffer overflow vulnerability?

There's a technique called **fuzzing** which is providing random inputs into a program and checking how it reacts. If the program responds with an error or another unintended consequence, then the program may be vulnerable to a buffer overflow.

Example code snippet:

```
char buffer[10]; //Here a character buffer with 10 bytes is
declared
```

ASLR Protection:

On most modern Linux systems, a buffer overflow won't work due to certain security measures like ASLR. **ASLR** or Address Space Layout Randomization, randomizes different memory segments in a program to make exploiting them more difficult. For the sake of the lab, we will be turning it off.

Run this command:`sudo sysctl -w kernel.randomize_va_space=0`

This will temporarily disable ASLR on the system.

Logging onto your computer:

- Use the generated username and password to login through Guacamole

The Mission

- In this lab you will be given multiple files (exploit.py, fuzz.py, vuln, flag.txt, getenvaddr)
- You must exploit the vulnerable program to gain root access to be able to read flag.txt
- Use fuzz.py to check if the program crashes. Then try and use the exploit to gain access. You may need to modify the address of the code to gain access.
- The program called getenvaddr will be used to check what memory address in the vulnerable program will be exploited. This may vary on your system so be sure to change the address in the exploit.py script.
- Use the following command to put the shellcode in an environmental variable:

-
  ```
  export PWN=`python -c 'print
  "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7
  \xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"'`
  ```
- (Optional) GDB or GNU Debugger will allow you to see what happens in memory after using fuzz.py
- Hint: You may have to string multiple commands together. Use **grep** and **|** to do so.