Lecture: Memory Corruption Attacks Challenge Problems

Advanced Course in Engineering

Instructor: Joseph Callen

Created by: Vahid Rajabian-Schwart

July 2022

<u>Instructions (0 Points)</u>
Provide a written description of your solution to each problem. Be sure to include a screen shot of each successfully solved problem. Staff may ask you to demonstrate your solution to any of the problems. There are 100 possible points for this assignment. You are only required to prepare a presentation of your solution to Problem 5 (part A and B).

<u>Problem 0: NOP Sled (6 points)</u>
An intrusion detection system may scan inputs for patterns of multiple NO-OP (0x90) bytes in a sequence. Please describe an alternative method in which a NOP sled may be crafted which does not consist of using NO-OP instructions.

<u>Problem 1: Canary Brute Force (6 points)</u>
A network enabled process forks a new process to handle each incoming connection. A forked child process shares the identical memory layout of its parent process and therefore shares the same canary value. The process provides feedback to the attacker that it has detected a stack smashing attempt. The attacker may craft packets of any byte length. At most how many attempts are required to guess the canary value when performing a buffer overflow against this process if it is protected by a 32-bit nullterminated canary? Describe the process. HINT: The answer is fewer than 2^24.

<u>Problem 2: Authentication ID (12 points)</u>
Buffer overflows can be leveraged to control an application without actually executing input shellcode. In this task you must demonstrate you are able to obtain administrator access to the program authFlagAdmin without providing the administrator password. Furthermore, you must solve this problem without mangling the return address. You may examine the source code to craft your input.

<u>Problem 3: Shell Bind TCP (16 points)</u>
The exercises presented thus far have used shellcode which spawns a shell on the local machine terminal. In this problem you are provided with a piece of shellcode that will listen for incoming TCP connections on port 1337 when executed. Once a connection is established, the shellcode binds a shell to the connection socket and effectively grants remote shell access to the target machine.

Your task is to create and demonstrate an exploit on the program “readData6” with the shellcode provided. Please disable ASLR for this problem. Note that this program is similar to the problem studied in lecture readData2. Recall that GDB may be used to gather information about the memory layout of the target program. In order to facilitate the exploit you may use a NOP sled as large as you wish. Demonstrate that the exploit was successful by connecting to the target machine over the network using Netcat as follows:

```
$ nc 127.0.0.1 1337
```

Note: The exploit may fail if there is an active connection on the designated port. To check if there if there is an active connection on a port use the command:

```
$ ss -tulpan | grep "1337"
```

Problem 4: Staging Access (5 points)
A memory corruption attack is commonly used as an initial access vector. These types of attacks can be limited by multiple constraints. For this reason, they are commonly used to stage additional payloads. Discuss how a memory corruption attack can be used to enable further persistent access, communications and control after initial exploitation.

Problem 5: readBinaryDataChallenge
Part A (25 points): Your task is to gain root privileges by exploiting the program readBinaryDataChallenge with all defenses enabled. The program contains both a printf() memory information leak and a buffer overflow. The program is configured to pause before processing the input file. This will allow you to perform your exploit in two stages.

While you plan your approach, consider the following questions:

- How will you trigger a memory information leak in order to 1) find the address of system(); and 2) expose the value of the canary?
- How will you find the address of the command (i.e '/bin/sh') you want to execute?
- Do canary values change from function to function?

Part B (30 points): When a program crashes abnormally an entry may be generated in a log file (/var/log/*). So far, we have not concerned ourselves with exiting cleanly in our return2libc exploits. However, for the sake of stealthiness, we must no longer neglect this requirement.

In this problem you are required to find the offsets of several functions and your command string from the C Library. We provide the template exploit code where you will need to fill in the missing values. You will need to use the tools “readelf”, “strings”, and “grep” to help you find the offsets needed. The C library you must examine is located here: /lib/i386-linux-gnu/libc.so.6

Once you have determined the offsets of the important functions, you must demonstrate that your exploit exits cleanly on the vulnerable program with all defenses enabled. What ‘exit code’ means a program exited cleanly?

Note: Because of the way pwntools works you will need to type ‘exit’ from your exploit shell to check the ‘exit code’ that is printed.