Buffer Overflows

Christopher De Jesus
christopher.dejesus@upr.edu
Computer Science Department
University of Puerto Rico - Río Piedras

Advisor:
Humberto Ortiz-Zuazaga
humberto@hpcf.upr.edu
Computer Science Department
University of Puerto Rico - Río Piedras

Abstract

In computer security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. This is a special case of the violation of memory safety.

1 Introduction

This independent study was based on a graduate course sponsored by MIT and given access by them for public availability. It was mostly based on the first lab of that course which emphasises in the exploitation and detection of buffer overflows. By detecting these buffer overflows we could then study them and get to a possible exploitation for it. The lab consists in detecting buffer overflows in a web server called Zoobar, which is written in C. We could then, after detecting the buffer overflow, run a python exploit for it to either crash the server, make denial of service or make it do something else, like running a shell which can access sensitive data inside the server.

2 Methodology and Tools Used

The first step of the lab was to detect buffer overflows so we can later on do an exploit for it. The buffer that we studied and tried to exploit was in the file http.c in the function http_serve(). (Figure 1).

The second step was to study the buffer overflow to see how it behaves and what can it do to harm the server in any way. One of the approaches was to

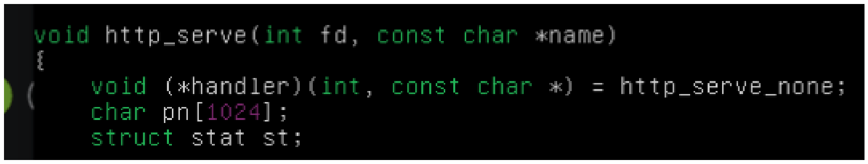

Figure 1: The buffer in http.c that can cause an overflow

insert 3,000 letter A into it and check what would happen if we did that. It did harmed the server so we chose this buffer to exploit. (Figure 2).

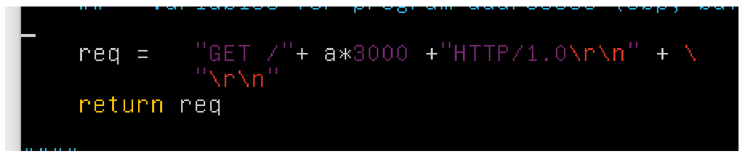

Figure 2: The request with the 3,000 A to check the buffer

The third step would be to then come up with an exploit that can harm the server, by either getting sensitive data, denial of service or crashing.

The tools used for each step were, obviously, the image with the server that was provided by the MIT course. The second tool used for the process was Virtual Box. (Figure 3).

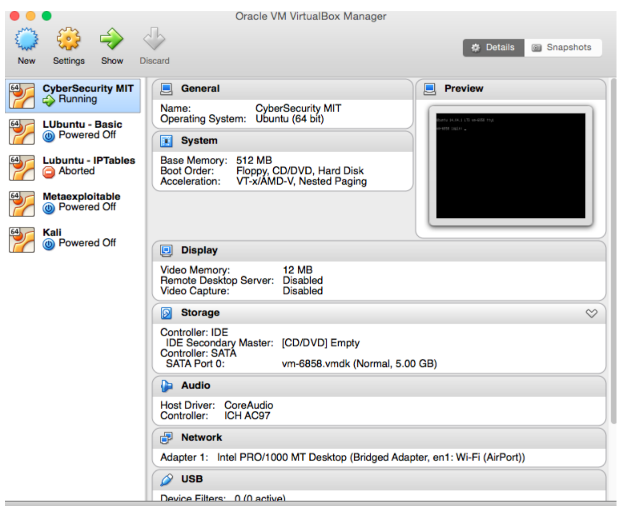

Figure 3: Virtual Box

Another helpful tool, used for debugging, was GNU gdb. With this tool we could search and detect what was being written inside the stack by the buffer overflow. We could use helpful commands like "print, break, continue, next, x (examine), info frame, info registers". (Figure 4).

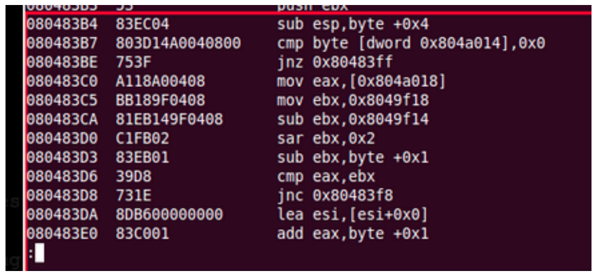

Figure 4: The GDB tool to check the stack

3 Finding the buffer overflows

To find a buffer we have to simply search for a temporary placeholder (variables in many programming languages) in memory (ram/disk) on which data can be dumped and then processing can be done. The buffer we found was called pn and had a 1024 bound. This means that if we put 1025 of whatever data type it needs, it will overflow. (Figure 5).

```
char buff[10] = {0};
```

```
strcpy(buff, "This String Will Overflow the Buffer");
```

Figure 5: An example of a buffer overflow

Programs in C are susceptible to these types of attacks. The standard C library provides unsafe functions (such as gets) that write an unbounded amount of user input into a fixed size buffer without any bounds checking.

4 After finding the buffer overflows

When the buffer overflow is actually found, we proceed study it and see how it behaves. This was done with GNU gdb, since some of the "buffers" inside the program sometimes aren't actual buffer overflows, but static buffers which will not overflow and will not accept anything more after the buffer is full. This happened to us when finding buffer overflows because they might look like they are susceptible to overflow but are not. That's why it is important to study it by injecting random strings of data to see where do they end up in the stack and to see if it writes the leftovers of the string elsewhere.

5 Future Work

Our future work is to do the other part of the lab of the MIT course which consist of injection of a shell code. In theory, we know how it works. In practice, it is much more complicated than it sounds, since you have to check the stack consistently to see what it is writing and the stack grows in a way that when you inject the code, it can go to the opposite way of what you were trying to do in the first place. (Figure 6).

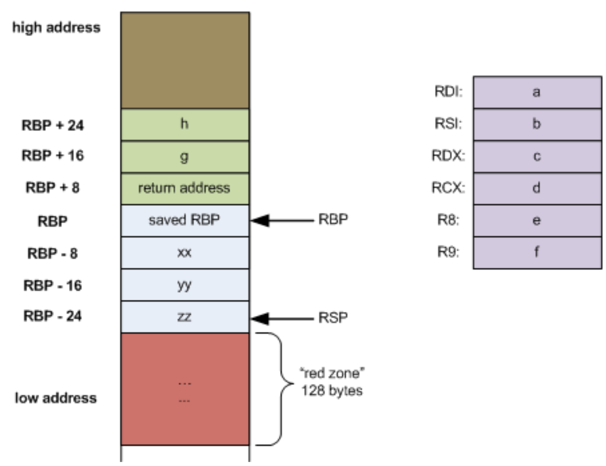

Figure 6: An example of a stack and how it works

6 Conclusion

The lab consisted in 3 steps: finding a buffer overflow, study the buffer and inject code. The first 2 steps were done and we can conclude that buffer overflow are a serious threat to any program or server that works with them since we can access sensitive data(or even worse) which was done with just injecting 3,000 A's, crashing the server.

Acknowledgement

Special thanks to my advisor/professor Humberto Ortiz-Zuazaga and partner Abimael Carrasquillo who worked in the same independent study and actually injected the code.

References

[1] Statically Detecting Likely Buffer Overflow Vulnerabilities. David Larochelle, David Evans. source: http://lclint.cs.virginia.edu/usenix01.pdf

[2] 6.858 Fall 2014 Lab 1: Buffer overflows. source: http://css.csail.mit.edu/6.858/2014/labs/lab1.html