Name:\_\_Xiang, Xin\_\_ Date: \_\_\_\_ \_April 10, 2020\_ \_ \_

Section: \_ 005\_ \_ \_ Github Username: \_\_FreyaXiang\_ \_

**Lab 3**

Total in points (100 points total):

Professor’s Comments:

Honor Pledge: I have neither given nor received aid on this assignment.

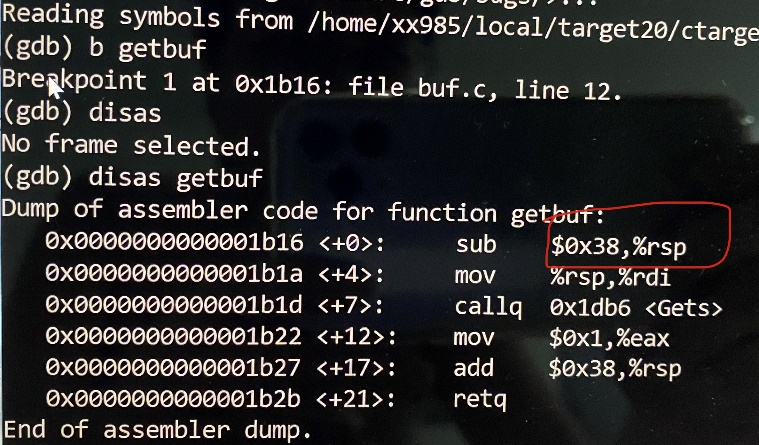
Signature: Xin Xiang

1. **Security Exploits Programming**

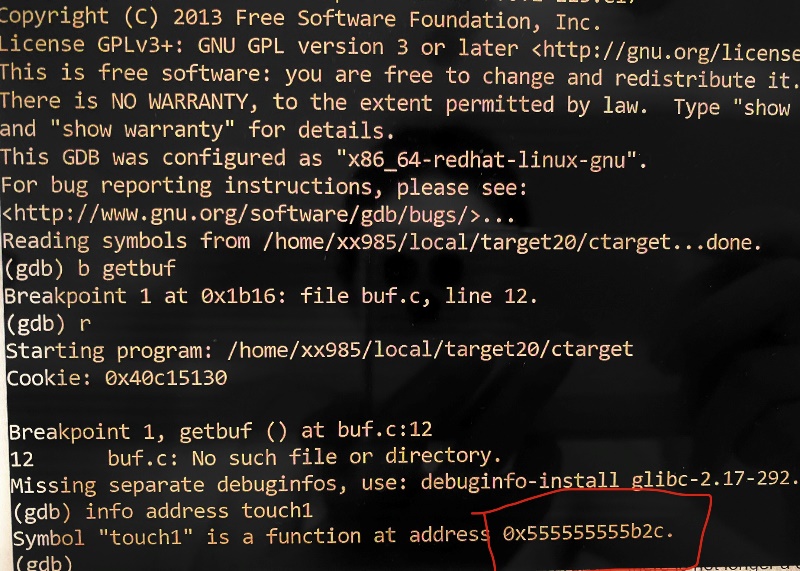
**Phase 1:**

We need to overflow the stack with the exploit string and change the return address of getbuf function to the address of touch1 function.

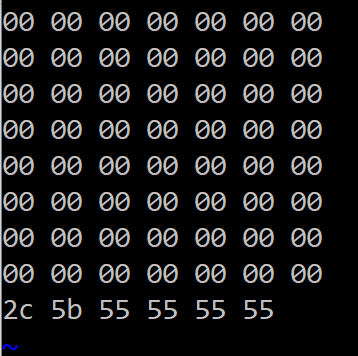
1. Run ctarget executable in gdb and set a breakpoint at getbuf. Then disassemble the getbuf function by “disas” command. Since the size of the stack is a run time constant, we need to look at the assembly code to figure it out.

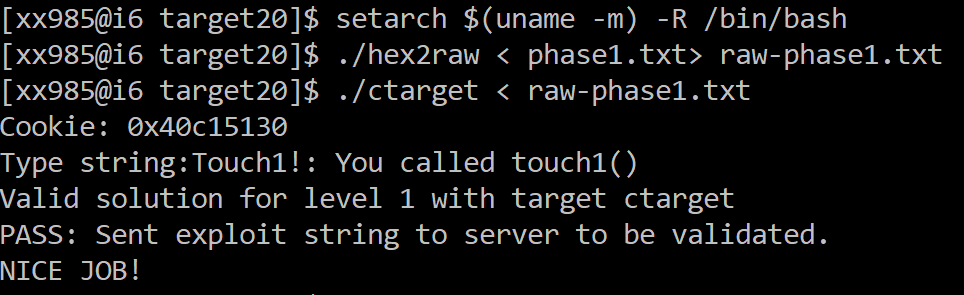


1. At the first line, we can see that 56 (0x38) bytes of buffer is allocated for getbuf().
2. Now we know the stack size and we need to input 24 bytes of padding followed by the return address of the touch1 address. Use gdb, set a breakpoint at getbuf, run the program, and use “info address touch1” to find the run-time address of touch1.



So the address is at 0x555555555b2c and the reverse order is 2c 5b 55 55 55 55.

1. Finally create a text file named phase1.txt which will look like below:
2. Now in the terminal type in “./hex2raw < phase1.txt > raw-phase1.txt” and then “./ctarget < raw-phase1.txt”. Then we can see that we are done with phase 1.



**Phase 2:**

1. Phase 2 injects a small amount of code and calls touch2 while passing the cookie as the first argument to touch2. We need to modify register %rdi and store our cookie in touch2.
2. First create a file called phase2.s and write the assembly code below with cookie.

movq $0x40c15130, %rdi /\* move my cookie to register %rdi \*/

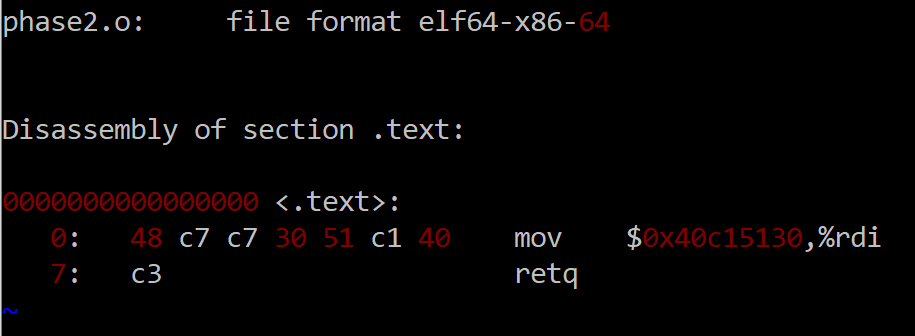
retq

1. Now we need the byte representation of the above assembly code. Compile it with gcc then disassemble it.

gcc -c phase2.s

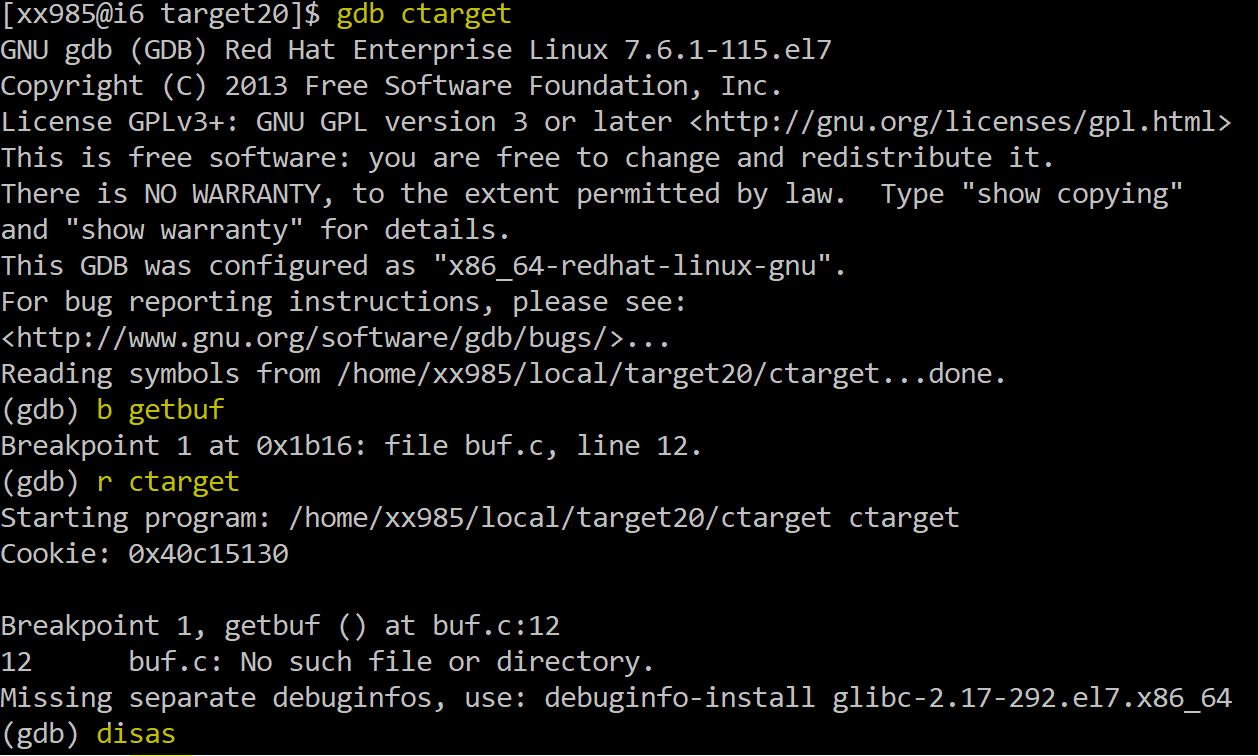
objdump -d phase2.o > phase2.d

1. Open the file phase2.d.

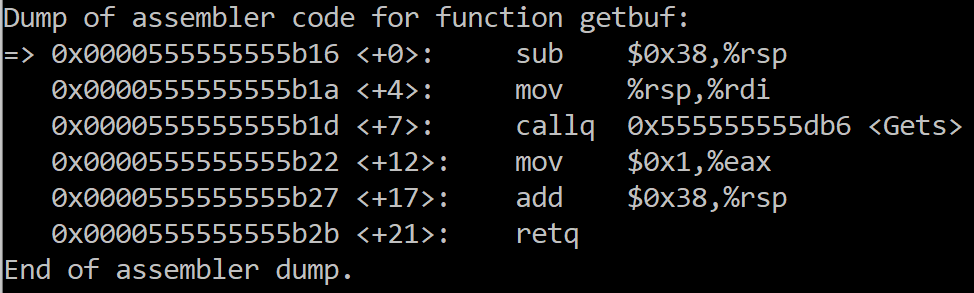


The byte representation of the assembly code is 48 c7 c7 30 51 c1 40 c3

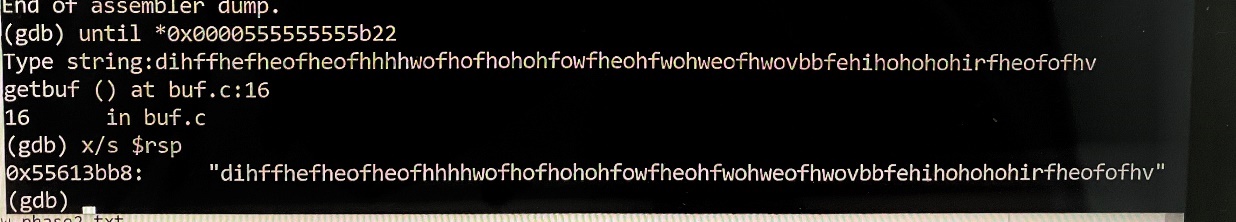
1. Now we need to find the address of %rsp register. Run ctarget through gdb, set a breakpoint at getbuf, run ctarget, and then use “disas” command.



1. We will get the assembly code below:

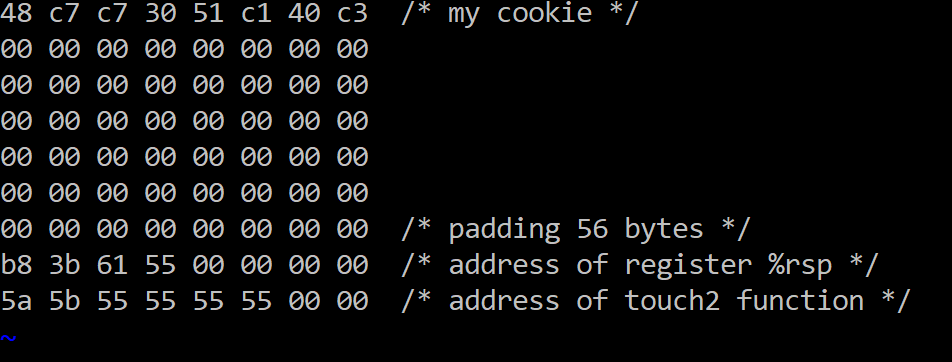


1. To get %rsp:

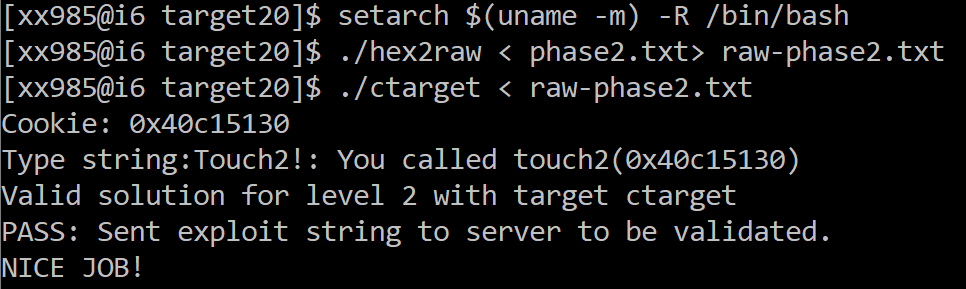


The address on the left side is what we want: 0x55613bb8.

1. Create another text file named phase2.txt.



1. Run it through hex2raw and then run the raw file:



**Phase 3:**

1. Phase 3 is similar to phase 2. We are trying to call touch3 and pass our cookie as an argument to it as string.
2. First pass the address for the cookie to register %rdi. The total bytes before the cookie are:

buffer + 8 bytes for return address of %rsp + 8 bytes for touch3

0x38 + 8 + 8 = 0x48

1. Grab the address for %rsp from phase 2: 0x55613bb8.

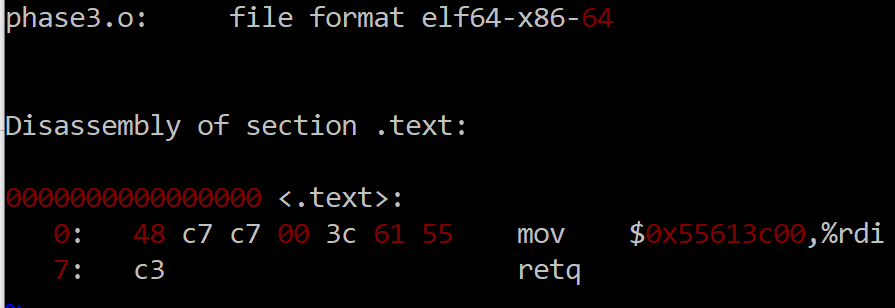
0x55613bb8 + 0x48 = 0x55613c00

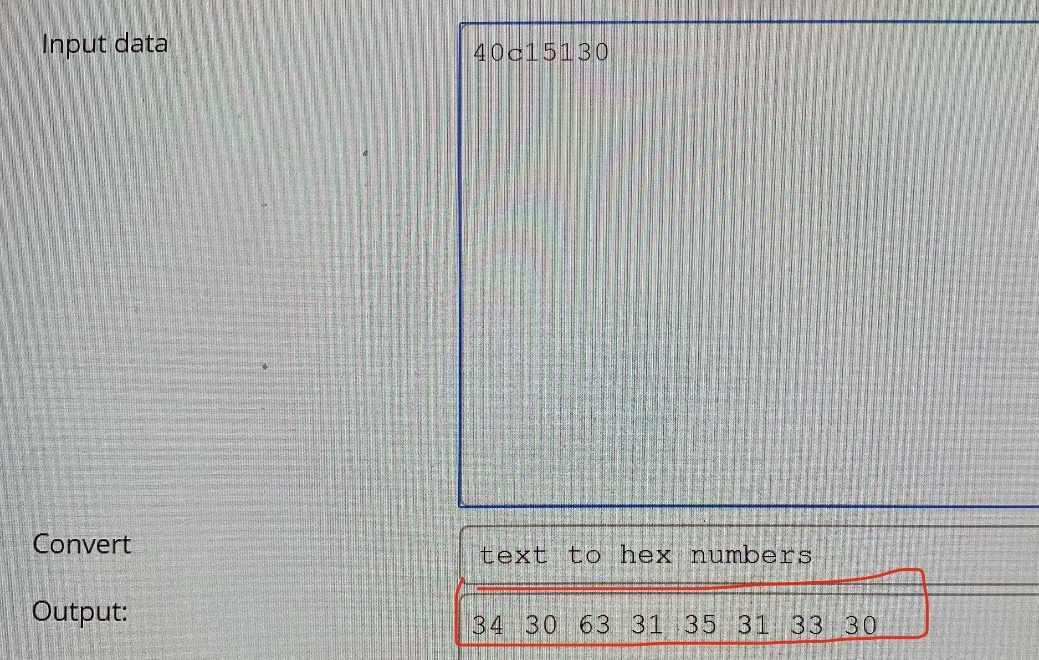
1. First create a file called phase2.s and write the assembly code below.

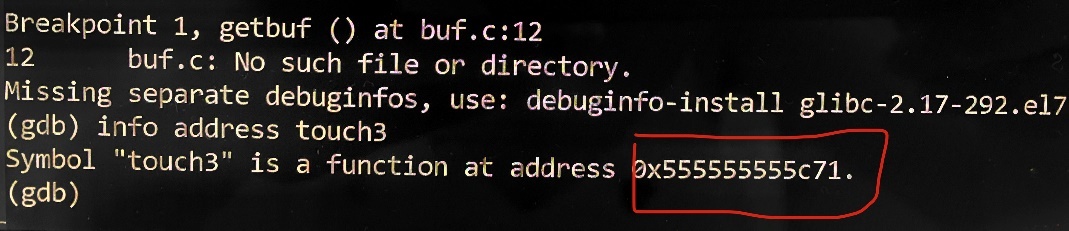
movq $0x55613c00, %rdi /\* %rsp + 0x48 \*/

retq

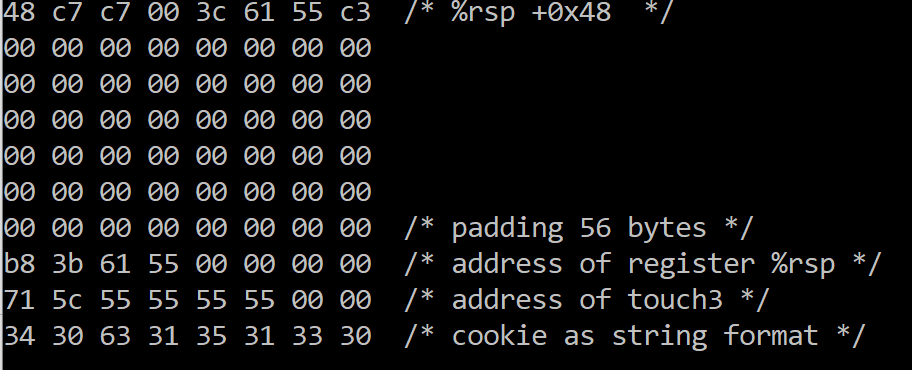
1. Use the same method to generate phase3.d and open it. The byte representation is as follows:



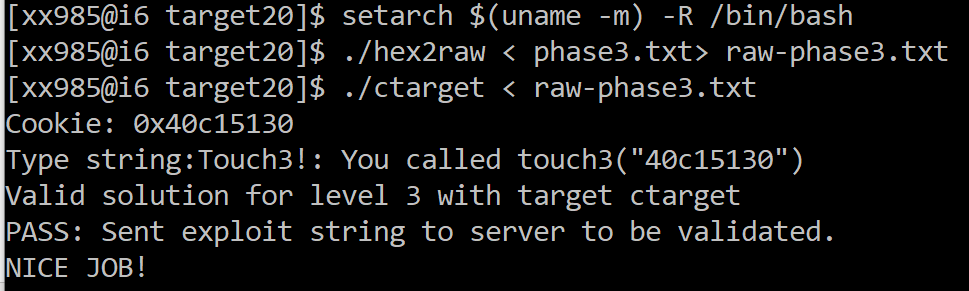
1. Then we should convert our cookie to from hex format to text format: 
2. Find the address of touch3:



1. Create a new file named phase3.txt:



1. Last step is to generate the exploit string using the hex2raw program and run the program:



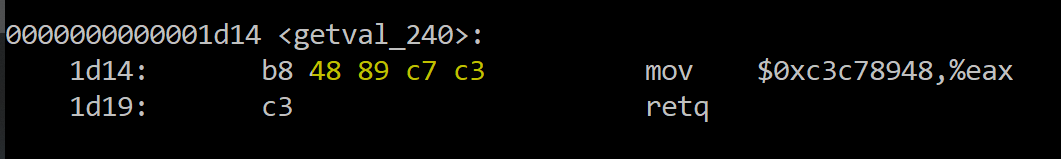
**Phase 4:**

1. In the pdf it tells us “It’s highly unlikely that a compiled function would have popq %rdi as its last instruction before ret.” From Figure 3B, the byte representation of popq %rdi is 5f. Since we don't have 5f byte in the dump file, we will look for a substitute:

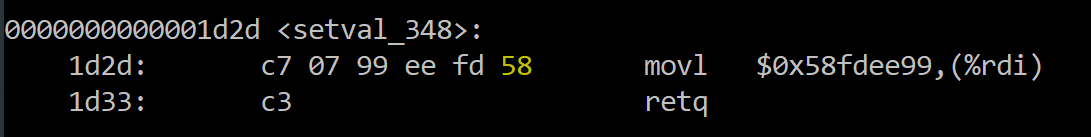
Gadget 1: movq %rax %rdi  → find 48 89 c7 (from Figure 3A)

Gadget 2: pop %rax → find 58 (from Figure 3B)

1. Then we need to find the gadgets in the assembly code of rtarget between start\_farm and end\_farm. To generate the assembly code of rtarget, use the command “objdump -d rtarget > rtarget\_dump.txt” and open the file “rtarget\_dump.txt”.
2. From start\_farm and end\_farm, we can find the two gadgets below:

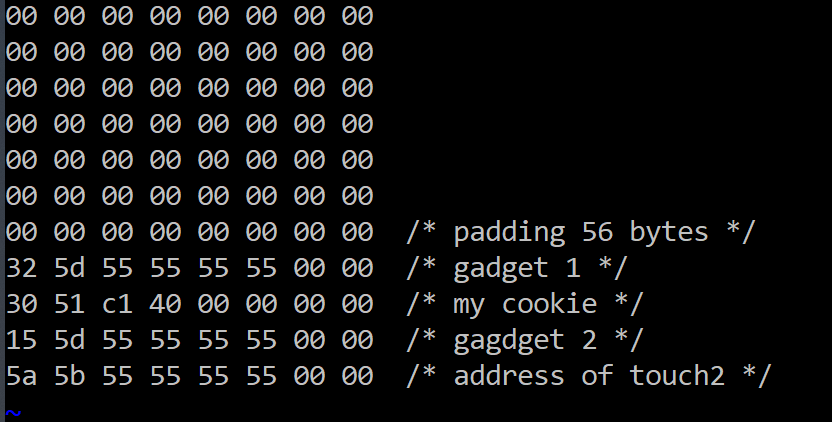


Address of the first gadget: 0x1d14+1=0x1d15

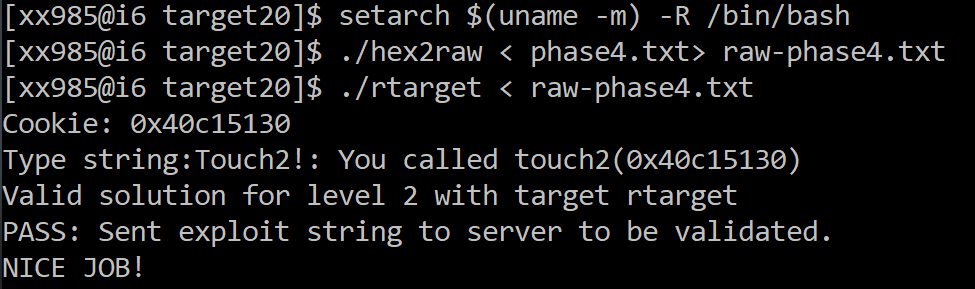


Address of the second gadget: 0x1d2d+5=0x1d32

1. Now we have 2 gadgets and can exploit the rtarget program. The next step is constructing our string, the format is padding for the buffer size, gadget 1 address, our cookie, gadget 2 address, return address and finally touch2 address.
2. Create a file named phase4.txt:



1. Last step is to generate the raw exploit string using the hex2raw program and run the raw file:



**Phase 5:**

1. The logic of the code is like this:
2. padding 56 bytes
3. save %rsp to %rdi (find code from Figure 3A and 3B)

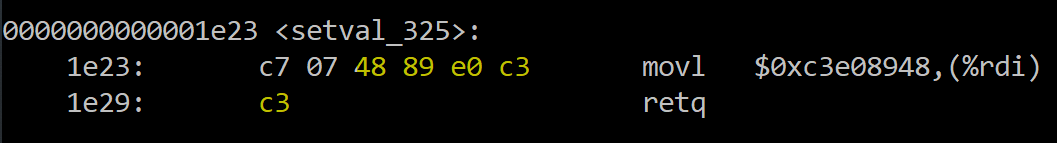
mov %rsp, %rax → find 48 89 e0  
mov %rax, %rdi → find 48 89 c7  
pop %rax → find 58

1. gap from gadget1 to cookie: offset 0x48
2. save the sum of the above to value to a register (find code from Figure 3C and 3D)  
   mov %eax, %edx → find 89 c2  
   mov %edx, %ecx → find 89 d1

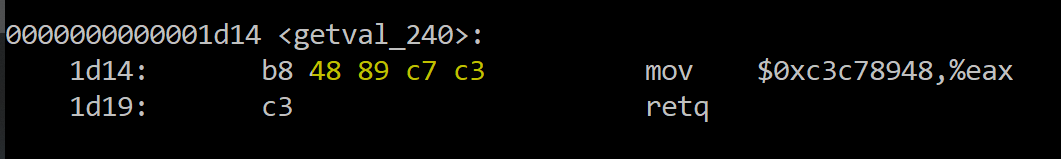
mov %ecx, %esi → find 89 ce  
lea (%rdi, %rsi,1), %rax → find function add\_xy  
mov %rax, %rdi → find 48 89 c7 (from Figure 3A)

1. the address of touch 3
2. cookie as string
3. Then we need to open the file rtarget\_dump.txt and look for functions between start\_farm and end\_farm.

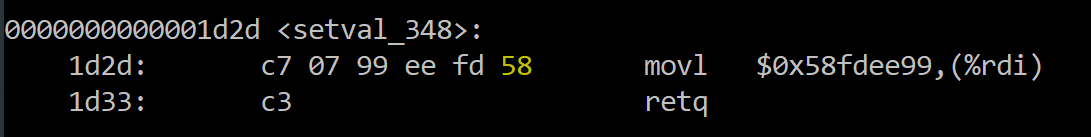
mov %rsp, %rax: 0x1e23+2=0x1e25



mov %rax, %rdi: 0x1d14+1=0x1d15

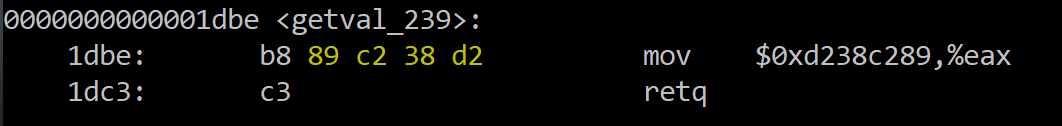


pop %rax: 0x1d2d+5=0x1d32

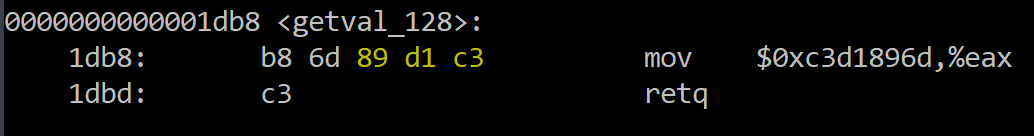


mov %eax, %edx: 0x1dbe+1=0x1dbf

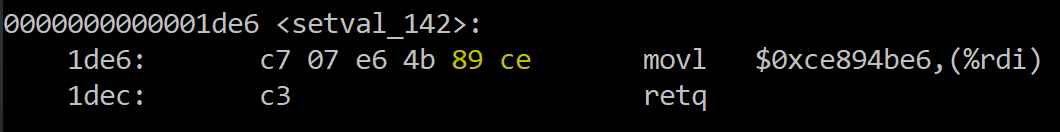
(From Figure 3D, 38 d2 is one of the functional nops)



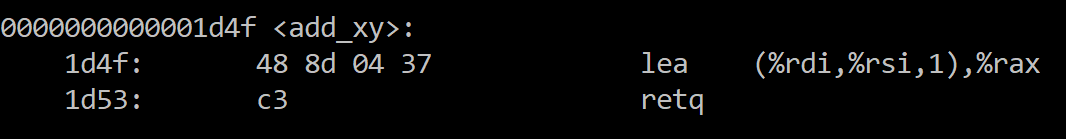
mov %edx, %ecx: 0x1db8+2=0x1dba



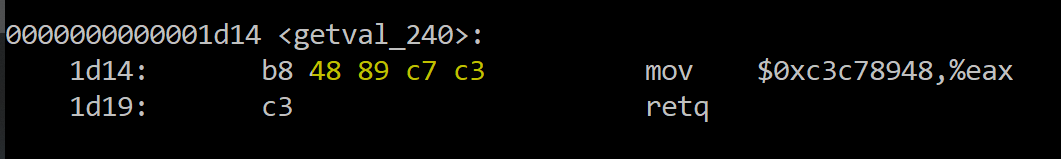
mov %ecx, %esi: 0x1de6+4=0x1dea



add\_xy: 0x1d4f

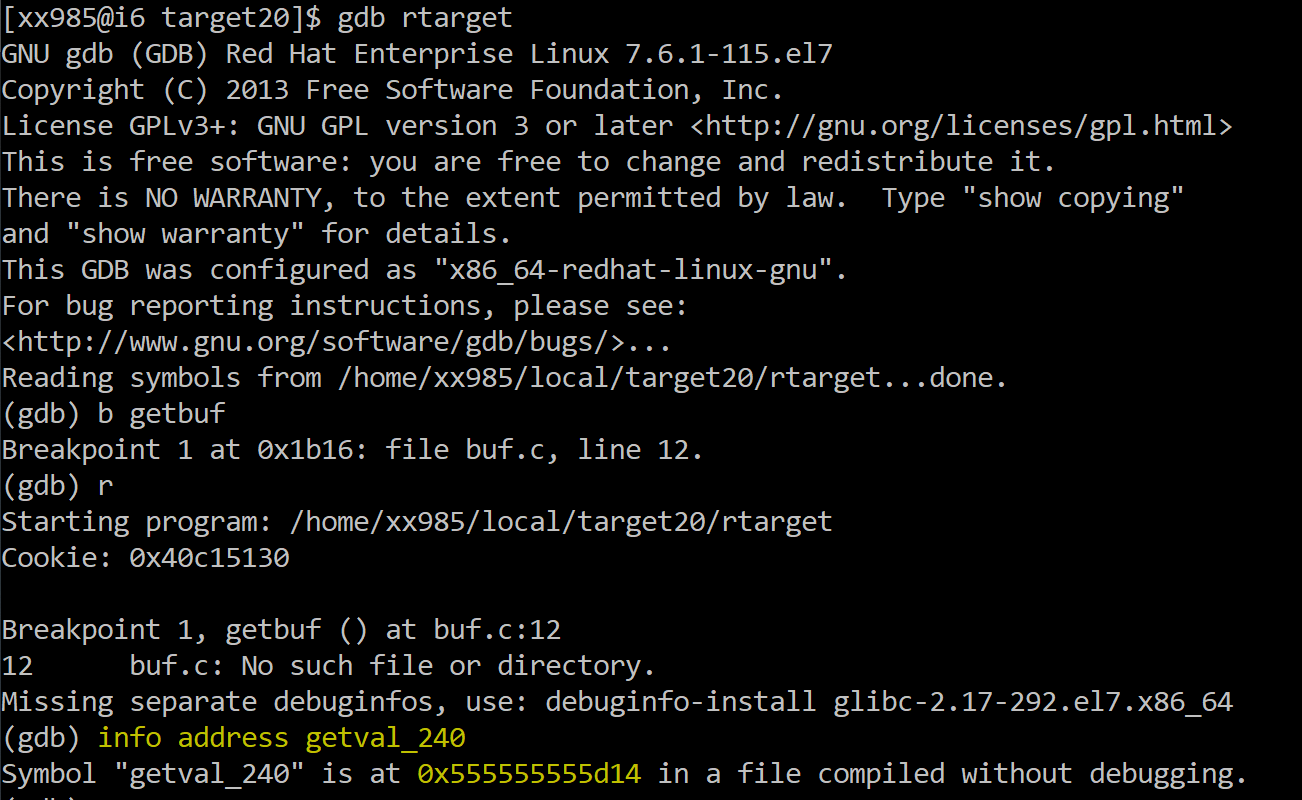


mov %rax, %rdi: 0x1d14+1=0x1d15

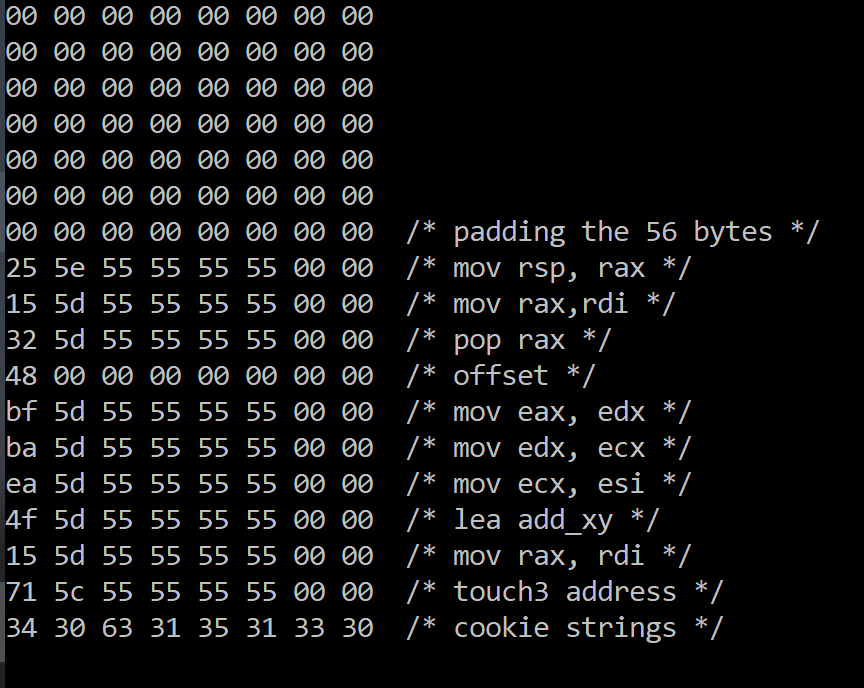


1. Find the run-time address of these functions by GDB:

Example:



1. Create a file called “phaseLast.txt” with the exploit string:



1. Generate the raw exploit string using the hex2raw program and run the raw file:

