CS201
REPORT: BOMB LAB

1551020 - Vo Tran Thanh Luong

December 1, 2016

Contents

1 Phase 1

Phase 1 : First we need to disas ctarget to assembly language file to see what it is doing inside. Because our exploiting technique needs to go through the getbuf function, we then search in the getbuf function. We can see that the command sub 0x28 %rsp indicates that the buffer is 40bytes long, so we must input the 40 bytes (in hexa of course) to get through buf, then 8 more bytes to penetrate the address of touch1 (0x401833). It will overwrite the address of the next command and make the program call touch1, not jump to test.

2 Phase2

Phase2 : We found out that at line 211 in ctarget.asm there is a line 400ecd: 48 c7 c7 b5 11 40 00 mov $0x4011b5,%rdi . This lines is to move a value to rdi. 48 c7 c7 is therefore machine code of a move command. Applying the same logic, we move our cookie to rdi. In our solution file for level 2 we need to replace the first six bits with 48 c7 c7, followed by the address of our cookie, which is d5 c2 6f 2d, and c3 to return. Then we use GDB to get the value of rdi. which we use to pass parameters to touch2. After all, we pass the touch2 function address in and then it's done.

3 Phase3

Your mission is to get CTARGET to execute the code for touch3. We must pass a string as its argument using our cookie. First we need to convert the cookie to string(a string is represented in C as a sequence of bytes followed by a byte with value 0). 0x2d6fc2d5 = 32 64 36 66 63 32 64 35. The address of rdi is a constant (the same to phase2) 58 bd 66 55. Now, on the first line, we do the same as phase 2. Instead of moving cookie to rdi using its value, it's now a pointer so we have to locate where the cookie is (in other words, we use its address). When functions hexmatch and strncmp are called, they push data onto the stack, overwriting portions of memory that held the buffer used by getbuf. As a result, the injected code should set register %rdi to the address of this string.

4 Phase4

Phase4 : We only needs 2 gadget to pass phase 4 . First we populate the first 56 bytes to fill the buffer. Then we need to pop rax (Read value at address given by STACK DIAGRAM (growing down) address of touch2() = 0x401949 gadget 2 = mov %rax to %rdi (cookie value now in %rdi) : hex =48 89 c7–>address: 0x401b01 value of cookie = 0x2d6fc2d5 gadget 1 = popq %rax (get the value of the cookie into %rax) : hex = 58 –>address: 0x401b24 padded buffer (completely filled with garbage)

5 Phase5

Phase 5 : We cannot move the address of cookie into rdi directly as we used to do in level3, hence we need to think of a different way to compute the address of cookie. After reading the asm code, we can see that it can be achieved by the function add xy (add the address in rdi and rsi and move the value to rax). First we need to populate the first 40 bytes of buf as usual. Then we move the address of rsp to rax and rax to rdi. Since we have the address of something in rdi, we need to pop the rax and take in the gap between the address of cookie and the address in rdi. From the moment we move the address of rsp to rax till the address of cookie, there are 9 addresses which are 8 gadgets and the address of touch3, exclude the value read in to rax and each address has 8 bytes. Clearly, the gap must be 72 bytes, which is equal to the address of 0x48. After that, we move the value in rax through edx(we only have the hex code of move eax to edx from start farm to end farm), ecx and then to esi. Then we try to use the function add xy to compute the address of cookie from 2 register rsi and rdi and move the value to rax, and then move to rdi before going to touch3 just like what we did in level3. When finding the address of hex code for each command we need to see if the code is followed only c3 or nops since from start farm to end farm, a command may appear many times.