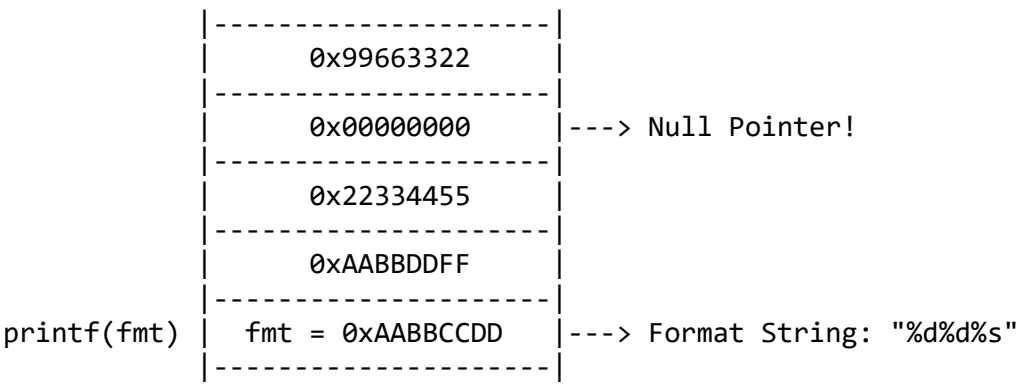
Problems

Question 6.4

When printf(fmt) is executed, the stack (from low address to high address) contains the following values (4 bytes each), where the first number is the content of the variable fmt, which is a pointer pointing to a format string. If you can decide the content of the format string, what is the smallest number of format specifiers that you can use crash the program with a 100 percent probability?

0xAABBCCDD, 0xAABBDDFF, 0x22334455, 0x00000000, 0x99663322

Answer



Two solutions --

(1) Most "predictable and guaranteed" format string: "%d%d%s"

We use %s on the third parameter which is zero. This would be interpreted as a null pointer. Dereferencing a null pointer will result in a segmentation fault with 100 percent probability.

(2) More aggressive "semi-unpredictable but guaranteed" format string: "%s%s%s"

This "more aggressive" solution will still guarantee a crash once reaching the third parameter. But, since we use %s in the first two parameters as well, we *may* crash sooner than that. No guarantees for the first two parameters, but it will definitely crash on the thrid parameter.

Either solution, using %s for the third parameter will guarantee a crash with 100 percent probability.

Mudit Vats
mpvats@syr.edu

Question 6.5.

A server program takes an input from a remote user, saves the input in a buffer allocated on the stack (Region 2 in Figure 6.9). The address of this buffer is then stored in the local variable fmt, which is used in the following statement in the server program:

printf(fmt);

When the above statement is executed, the current stack layout is depicted in Figure 6.9. If you are a malicious attacker, can you construct the input, so when the input is fed into the server program, you can get the server program to execute your code? Please write down the actual content of the input (you do not need to provide the exact content of the code; just put “malicious code” in your answer, but you need to put it in the correct location).

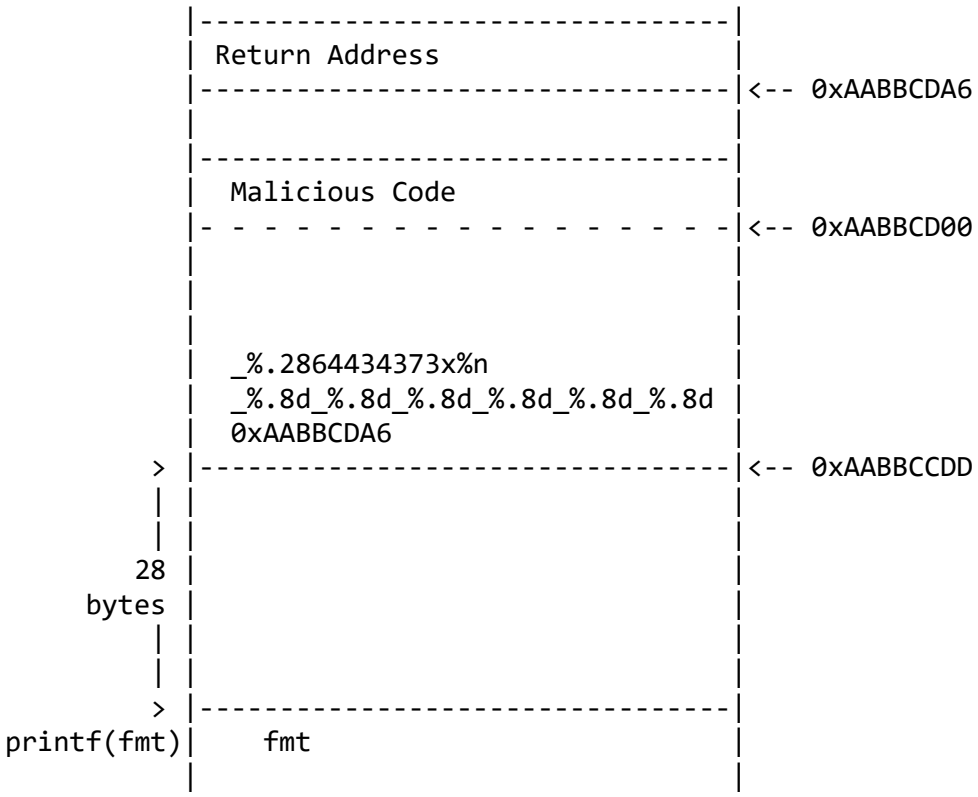
Answer

The diagram below shows the stack layout. Malicious code is the chunk of code that we wish to jump to. 0xAABBCD00 is the start of this code (arbitrarily chosen for this exercise at an offset > 0xAABBCCDD to fit inside the data region). The idea is to replace the contents of the return address at 0xAABBCDA6 with the address of the malicious code , 0xAABBCD00.

To accomplish this, we create a format string which places the return address as the first parameter, followed by the format specifier and finally the value to place into the return address. The parts are broken down as follows :

1. First Parameter – This value is the return address. This is where the %n writes the character count.
2. Format Specifiers – These values advance the argument pointer 24 bytes. Note, the distance is 28 bytes, but we stop at 24 bytes so that the following %x can argument can be processed.
3. %x Format Specifier – This specifier along with the precision value will pad whatever value that is printed with the precision value specified. In this case, we pad with a very large number (2,864,434,373) so that the following %n can calculate the correct Malicious Code "jump-to" address, 0xAABBCD00.
4. %n Format Specifier – This specifier is the sum of all printed characters, cumulative. It should equal the "jump-to" address, 0xAABBCD00.

Mudit Vats
mpvats@syr.edu




```
Printed Byte Count:
 _%.2864434373x%n              0xAABBCCC5+1 => 0xAABBCCC6 bytes
 _%.8d_%.8d_%.8d_%.8d_%.8d_%.8d        48+6 => 0x00000036 bytes
 0xAABBCDA6                               4 => 0x00000004 bytes
-----------------------------------------------------------------
malicious Code "jump to" address                0xAABBCD00 bytes
```

Final Raw Format String:
"0xAABBCDA6_%.8d_%.8d_%.8d_%.8d_%.8d_%.8d_%.2864434373x%n"

This format string will write address of the malicious code into the contents of the Return Address. When the function which calls printf returns, it will jump to the malicious code address.

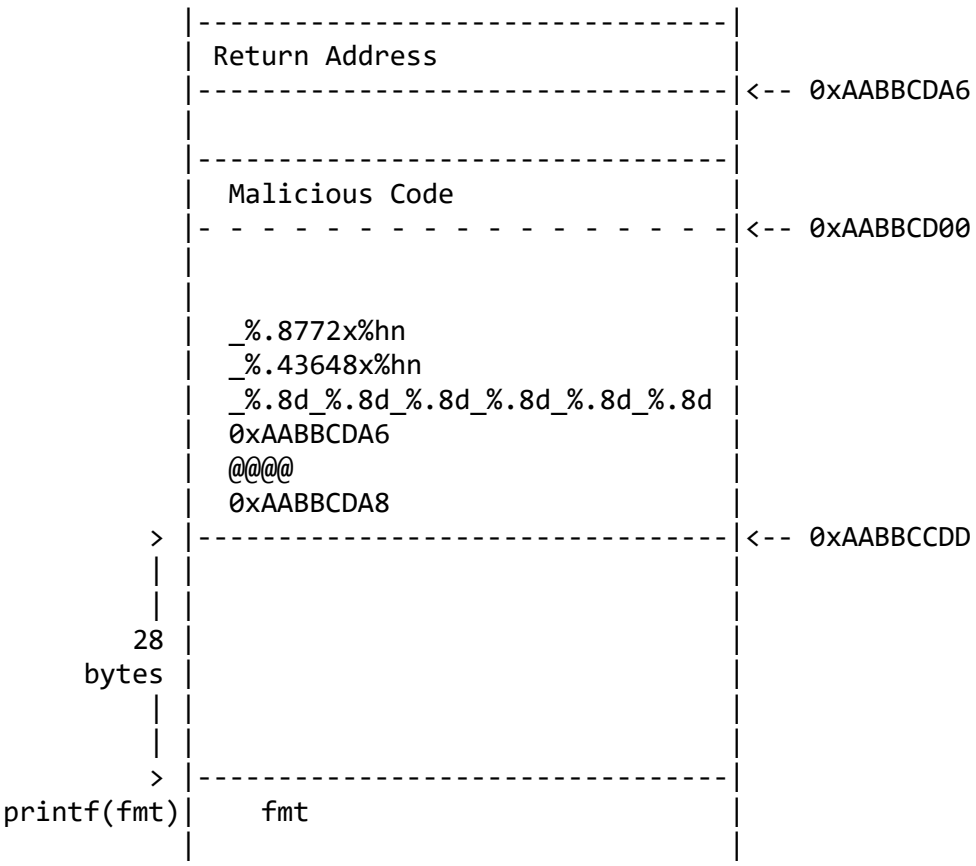
Mudit Vats
mpvats@syr.edu

Question 6.6.

If your answer to Problem 6.5. causes the server to print out more than a billion characters, it may take a while for your attack to succeed. Please revise your answer, so the total number of characters printed out is less than 60,000.

Answer

This answer is similar to the previous question with the difference that we split the Malicious Code "jump-to" address into high-order and low-order bytes (2 bytes each) and then copy them to the high-order and low-order sections of the return address. Slightly more complex, but much better performant since the number of characters printed is significantly less; I.e. less than 60 thousand vs greater than 2 billion!



Mudit Vats
mpvats@syr.edu

```
Printed Byte Count:
Low Order Bytes
 _%.8772x%n  (low order bytes)     0x2244+1 => 0x2245 bytes
    equals. 0xCD00 – 0xAABB - 1
 Previous Printed High Order Bytes             0xAABB bytes
-------------------------------------------------------------------
Malicious Code "jump to" low order bytes       0xCD00 bytes

High Order Bytes:
 _%.43648x%n (high order bytes)    0xAA80+1 => 0xAA81 bytes
    equals. 0xAABB – 0x3B

 _%.8d_%.8d_%.8d_%.8d_%.8d_%.8d        48+6 => 0x0036 bytes
 0xAABBCDA6                               4 => 0x0004 bytes
-------------------------------------------------------------------
Malicious Code "jump to" high order bytes      0xAABB bytes
```

Final Raw Format String:

"0xAABBCDA8@@@@0xAABBCDA6_%.8d_%.8d_%.8d_%.8d_%.8d_%.8d_%.43648x%hn_%.8772x%hn"

This format string will write address of the malicious code into the contents of the Return Address. In this case, the malicious code address is split into high-order and low-order addresses whose who's values are independently written to the high-order and low-order bytes (two bytes each) of the return address. When the function which calls printf returns, it will jump to the malicious code address.

Mudit Vats
mpvats@syr.edu