Format String Vulnerability

S6.1. Please write a function that takes a variable number of strings as its arguments, and prints out their total length.

S6.2. Both buffer-overflow and format-string vulnerabilities can lead to the modification of the return address field, but the ways how the field is modified are different in these two attacks. Please describe their difference, and comment on which one is less restricted.

S6.3. Can we use the StackGuard idea to protected against format-string attacks?

S6.4. 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
```

S6.5. A server program takes an input from a remote user, saves the input in a buffer allocated on the stack (Region ❷ in Figure 1). 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 this statement is executed, the current stack layout is depicted in Figure 1. 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).

S6.6. If your answer to Problem S6.5. 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 `80,000`.

S6.7. The problem is based on Problem S6.5. What if the distance between Region ❷ and Region ❸ in Figure 1 is not 28 bytes, but instead it is 26 bytes? This number is not divisible by 4, but each `%x` format string specifier does move the format string pointer by four bytes. Please construct your format string.

S6.8. ★
The problem is based on Problem S6.5. The difference is that we do not know exactly where the return address is stored. We only know that it is stored in one of the following addresses: `0xAABBCDA0`, `0xAABBCDA4`, `0xAABBCDA8`, or `0xAABBCDAC`.

1. Please construct one format string to exploit the vulnerability, such that your attack will be successful regardless of which of the address is the right one. The total number of characters printed out must be less than 80,000.

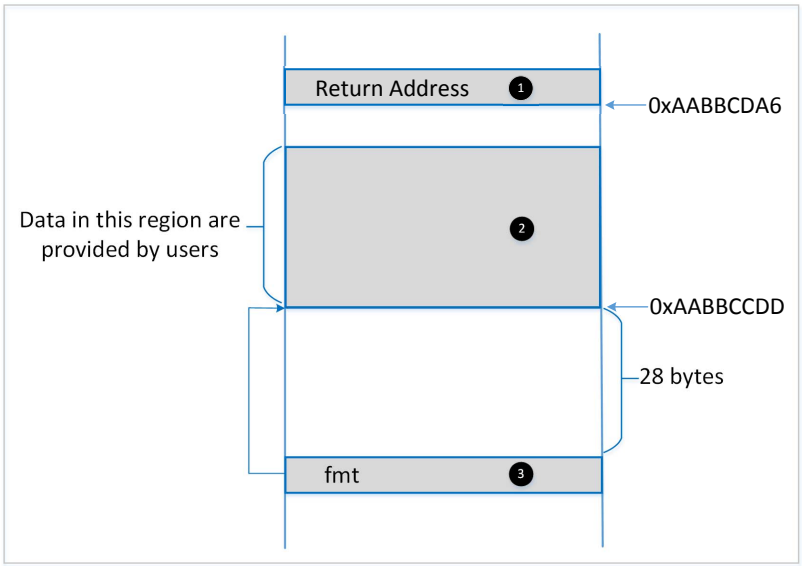

Figure 1: Stack Layout for Problem S6.5.

2. What is the shortest format string that you can come up with (the total number of characters printed out must be less than 80,000).

S6.9. Compilers can give a warning if it detects that the number of arguments in `printf()` does not match with the number of format specifiers. Please comment on the limitation of this countermeasure.

S6.10. Since `printf()` does not require any privilege, we can temporarily disable the program's privilege before we execute this statement; this way, even if the format-string vulnerability is exploited, attackers will not be able to gain much privilege. Please comment on this idea.

S6.11. If we make the stack non-executable, can we still exploit the format string vulnerability to get the victim program to spawn a shell?

S6.12. We are going to exploit a format-string vulnerability using the return-to-libc technique. Please describe in details what part of the stack needs to be modified and how you can achieve that by exploiting a format-string vulnerability. Please use Figure 1 when describing your solution.

S6.13. What if we want to write a small number such as 0 to a target address using a format-string vulnerability, but due to the `%x`'s that we have to place in the format string, the number of characters printed out by `printf()` is already nonzero before the `va_list` pointer reaches the target address. Is it still possible to write 0 to the target address?