CSE 484 Exam 1: Software Security

Question 1. (25 points). 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 the above 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). You are not allowed to use `%n`, but you can use `%hn` or `%hhn`. Here are some number conversions that you might need: `0x1000` = `4096` and `0x2000` = `8192`.

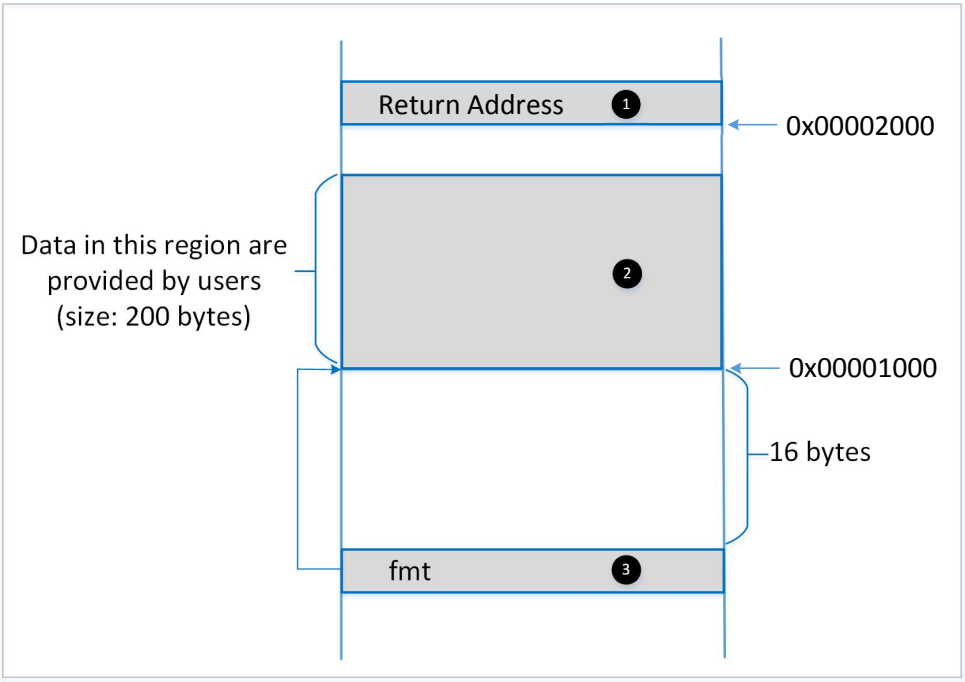

Figure 1: Stack Layout for Question 1.

Question 2. (20 points). The following function is called in a privileged program. The argument `str` points to a string that is entirely provided by users (the size of the string is up to 300 bytes). When this function is invoked, the address of the `buffer` array is `0xAABB0010`, while the return address is stored in `0xAABB0050`. Please write down the string that you would feed into the program, so when this string is copied to `buffer` and when the `bof()` function returns, the privileged program will run your code. In your answer, you don't need to write down the injected code, but the offsets of the key elements in your string need to be correct.

```
int bof(char *str)
{
   char buffer[24];
   strcpy(buffer,str);
   return 1;
}
```

Question 3. (10 points). Please use the StackGuard idea to rewrite the following function, making it immune to buffer-overflow attacks.

```
void func (char *str)
{

        char buffer[12];
        strcpy (buffer, str);

        return;
}
```

Question 4. (20 points). When a process maps a file into memory using the `MAP_PRIVATE` mode, the memory mapping is depicted in Figure 2. (1) Please describe what is going to happen when this process writes data to address `0x5100`. (2) Where was the Dirty COW race condition problem (before it was fixed)? (3) How can this race condition vulnerability be exploited?

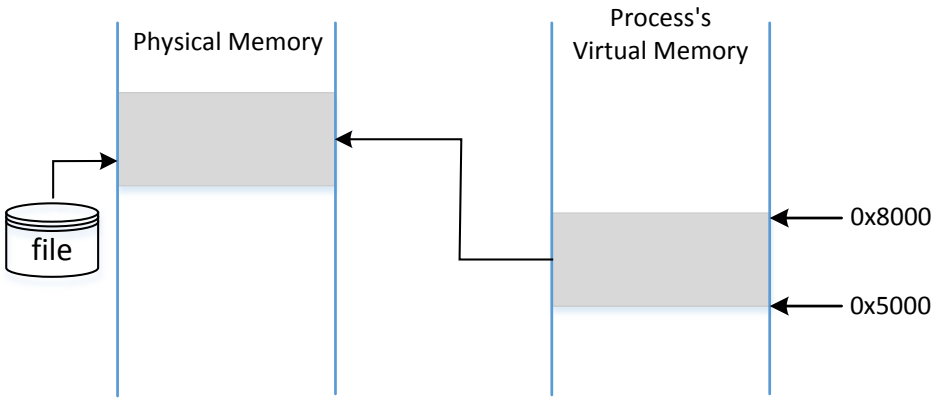

Figure 2: Figure for Question 4.

Question 5. (25 points). Sam found a very useful web page, which contains links to many interesting papers. He wants to download those papers. Instead of clicking on each of the links, he wrote a program that parses a HTML web page, get the papers' URLs from the web page, and then use a program called `wget` to fetch each identified URL. The following is the code snippet:

```
char command[100];
char* line, URL;

line = getNextLine(file);// Read in one line from the HTML file.
while (line != NULL) {
  URL = parseURL (line); // Parse the line and returns a URL string.
  if (URL != NULL){
    sprintf(command, "%s %s", "wget", URL); // construct a command
    system(command); // execute the wget command to download the paper
  }
  line = GetNextLine(file);
}
```

The function `sprintf` is quite similar to `printf`, except that `sprintf` puts the output in a buffer pointed by the first argument, while `printf` sends the output to the display. Please be noted that the functions `getNextLine()` and `parseURL()` are also implemented by Sam (their code is not displayed here). The program `wget` is a command-line program in Unix that can be used to download web files from a given URL.

The owner of the web page knows what Sam is doing with his page; he wants to attack Sam's program. He knows the code above, but he doesn't know how Sam implements `GetNextLine()` or `ParseURL()`, but he suspects that Sam may make some mistakes there. If you are the attacker, please describe how you plan to attack.