Worksheet 5 Solutions

1. What do I need to do if I want to access a function through buffer overflow (what needs to be done to the stack)? The function's address is 0x500142.

The function Gets is similar to the standard library function gets—it reads a string from standard input (terminated by '\n' or end-of-file) and stores it (along with a null terminator) at the specified destination (such as a char array previously declared). Functions Gets() and gets() have no way to determine whether their destination buffers are large enough to store the string they read.

```
Dump of assembler code for function getbuf:
=> 0x0000000000601748 <+0>:     push   %rax
   0x000000000060174c <+4>:     sub    $0x40,%rsp
   0x000000000060174f <+7>:     mov    %rsp,%rdi
   0x0000000000601754 <+12>:    callq  0x40198a <Gets>
   0x0000000000601759 <+17>:    add    $0x40,%rsp
   0x000000000060175d <+21>:    pop    %rax
   0x000000000060175f <+23>:    retq
```

Have 64 bytes of padding for the sub and 8 bytes to account for the push (draw the stack). When the function returns, we want the address popped into %rip to be the address of the function we want to run. Thus, we need 72 bytes of padding, and we place the address after that. This is the hex input we would pass in to a magical function like hex2raw which would give us the raw string that when fed into getbuf and placed onto the stack, would convert to the hex input.

```
PP AA DD DD II NN GG 01
PP AA DD DD II NN GG 02
PP AA DD DD II NN GG 03
PP AA DD DD II NN GG 04
PP AA DD DD II NN GG 05
PP AA DD DD II NN GG 06
PP AA DD DD II NN GG 07
PP AA DD DD II NN GG 08
PP AA DD DD II NN GG 09
42 01 50 00 00 00 00 00
```

What do I need to do if I want some instructions to be executed before the function is accessed? (Assume the value of %rsp right before getbuf is called is 0xabcd0000)

First set the return address to a location on the stack, where the instructions will be executed. At that location, write the instructions that you want to be executed. Then include a return statement. We then want to set the return address to the location of the function.

```
PP AA DD DD II NN GG 01
PP AA DD DD II NN GG 02
PP AA DD DD II NN GG 03
PP AA DD DD II NN GG 04
PP AA DD DD II NN GG 05
PP AA DD DD II NN GG 06
PP AA DD DD II NN GG 07
PP AA DD DD II NN GG 08
PP AA DD DD II NN GG 09
10 00 cd ab 00 00 00 00       // address of our instructions
42 01 50 00 00 00 00 00       // address of function to call
IN ST RU CT IO NS SS SS       // instructions to execute
IN ST RU CT IO NS SS SS
IN ST RU CT IO NS SS SS
...
IN ST RU CT IO NS SS SS
IN ST RU CT IO NS SS SS
c3                            // note: c3 is the bytecode for ret
```

2. What are some optimizations that can be made to the following function?

```
void cs33fun(char* Midterm, char* Grade, int* Final, int n) {

      for (int i = 0; i < (strlen(Midterm)); i++) {
            strcat(Grade, Midterm);

            for (int j = 0; j < n; j++)
                  for (int k = 0; k < i; k++)
                        Final[j] += strlen(Grade);
      }
}
```

There are many ways this function can be optimized, including but not limited to:

- The innermost loop can be replaced with the statement:
  `Final[j] += i * strlen(Grade);`
- Move `strlen(Midterm)` outside of the loop

There are several things you SHOULDN'T do:

1. Based on "Procedure calls" - Move strcat out of the loop
   - Strcat is required for the logic of the function
2. Based on "Procedure calls" - Move `strlen(Grade)` outside of the outermost loop (and nothing else)
   - The string `Grade` changes over each iteration of the outermost for loop
   - BUT can be moved outside of the middle loop
     - Since strlen(Grade) increments by strlen(Midterm) during each outermost iteration, can actually be moved outside the outermost loop if handled correctly