# ECE 568 – Computer Security

**The Edward S. Rogers Sr. Department of Electrical and Computer Engineering**

**Midterm Examination, Part 1, October 2021**

| | |
|---|---|
| **Name** | |
| **Student #** | |

Answer all questions. Write your answers on the exam paper. Show your work.
Each question has a different assigned value, as indicated.

Permitted: one 8.5 x 11”, two-sided page of notes.
No other printed or written material. No calculator.
NO PHOTOCOPIED MATERIAL
Total time: 50 minutes
Total marks available: 50 (roughly one mark per minute)
Verify that your exam has all the pages.
**Only exams written in ink will be eligible for re-marking**.

| 1 /25 | 2 /25 | Total |
|---|---|---|
| | | |

# Question 1: Buffer overflows [25 marks]

**Program:**

```
1:  int foo(char *arg) {
2:    char    buf[64];
3:    int p, j, min, len;
4:
5:    p = 138;
6:    min = (strlen(arg) > p) ? p : strlen(arg);
7:    len = min;
8:
9:    for (j = 0; j <= len; j++)
10:     buf[j] = arg[j+10];
11:
12:   return 0;
13: }
14:
15: int main(int argc, char *argv[]) {
16:   char string[20] = "abc";
17:
18:   str = &string[1];
19:   /* the arguments for snprintf are
          int snprintf(char *target_buffer, size_t len,  char * fmt_str, ...) */
20:   snprintf(str, 20, argv[1]);
21:   foo(argv[2]);
22:
23:   return 0;
24: }
```

**Registers:**

```
rbp            0x7ffdba279940
rsp            0x7ffdba2798e0
```

**Stack: (output of x/52x &buf)**

```
0x7ffdba2798f0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7ffdba279900: 0x00000000      0x00000000      0x00000000      0x00000000
0x7ffdba279910: 0x00000000      0x00000000      0x00000003      0x00000000
0x7ffdba279920: 0xba279a68      0x00007ffd      0xba279a88      0x00007ffd
0x7ffdba279930: 0x00000000      0x00000000      0x0000008a      0x00007f2f
0x7ffdba279940: 0xba279980      0x00007ffd      0x00400697      0x00000000
0x7ffdba279950: 0xba279a68      0x00007ffd      0x00000000      0x00000003
0x7ffdba279960: 0x00400061      0x00000000      0x004004f0      0x00000000
0x7ffdba279970: 0xba279a60      0x00007ffd      0x00000000      0x00000000
0x7ffdba279980: 0x004006a0      0x00000000      0x796aa493      0x00007f2f
0x7ffdba279990: 0xba279a68      0x00007ffd      0xba279a68      0x00007ffd
0x7ffdba2799a0: 0x7980d548      0x00000003      0x00400654      0x00000000
0x7ffdba2799b0: 0x00000000      0x00000000      0xe9d7f02d      0x66c14e70
```

**Other addresses:**

```
&buf:  0x7ffdba2798f0
&len:  0x7ffdba279930
&j:    0x7ffdba279934
&p:    0x7ffdba279938
&min:  0x7ffdba27993c
```

A program with a number of possible buffer overflow vulnerabilities is given above. The program is executed with an input passed in at the command line from the attacker. The state of the registers and stack when the program reaches line 6 is given. Note that all addresses are 64-bit addresses. Answer the following questions (next page):

1) Would an attacker be able to use either buffer `buf` or `string` to corrupt memory in the program? Please give a range of what memory can be corrupted in both cases [6 marks]:
   a) `buf`

   b) `string`

2) At what addresses on the stack are the return address of `main` and `foo` located and what are the values of those return addresses? Explain your answer [8 marks]:
   a) Return address of `foo`

   b) Return address of `main`

3) Please draw a diagram of the attack buffer needs to inject into the program to exploit the loop writing to `buf` in the function `foo`. Please give size of the nop sled, shellcode, return address and other elements in the buffer. For all values other than nops and shellcode, please give the values to be written. Assume the shellcode is 46 bytes in size. You can assume you are able to inject as many null characters as you need with environment variables [8 marks]

4) Suppose the line 9 and the following loop is changed to:

```
9:  arg = arg + 10;
10: for (j = 0; j <= len; j++)
11:   *buf++ = *arg++;
```

Does this affect the need to use environment variables? Why or why not? [3 marks]

## Question 2: Defenses and ROP attacks [25 marks]

1) Please fill in the following table. For the performance column, please indicate whether the performance is considered "better" (with a "+") or "worse" (with a "-") than most other defenses.

   For the "Stack smashing", "Format String", "Double Free" and "ROP" columns, indicate whether the defense makes harder/impossible to achieve the attack's objective (with a "+") or does not make the attack harder at all (with a "-"). You can consider the objective of "Stack smashing", "Format String", and "Double Free" to be to corrupt a memory location (could be data, a pointer or code), while "ROP" is to execute code of the attacker's choosing. [18 marks]

| Defenses | Performance | Stack smashing | Format String | Double Free | ROP |
|---|---|---|---|---|---|
| Stack Canaries | | | | | |
| NX pages | | | | | |
| ASLR | | | | | |
| Type-safe language (i.e. JAVA) | | | | | |
| CFI | | | | | |
| kBouncer/ROPecker | | | | | |

You may add any explanations you have for your answers below:

2) Suppose an attacker wants to mount a ROP attack against the vulnerability in the function `foo` from question 1 (on page 2). Assuming that gadgets the only way gadgets can affect rsp is to return or pop values off the stack, what region of the stack would they want to overwrite with the gadgets and fake arguments in their ROP attack? For argument's sake, you can assume that the top address of the stack region is `0x7ffdba27a000`. Please give the starting and ending address. [4 marks]

3) Given the assumptions in the previous question, what is the maximum number of gadgets the attacker could invoke in their ROP attack? You may give your answer in hex if you wish. Please justify your answer [3 marks]

# ECE 568 – Computer Security

**The Edward S. Rogers Sr. Department of Electrical and Computer Engineering**

**Mid-term Examination, Part 2, October 2021**

| | |
|---|---|
| **Name** | |
| **Student #** | |

Answer all questions. Write your answers on the exam paper. Show your work.
Each question has a different assigned value, as indicated.

Permitted: one 8.5 x 11", two-sided page of notes.
No other printed or written material. No calculator.
NO PHOTOCOPIED MATERIAL
Total time: 50 minutes
Total marks available: 50 (roughly one mark per minute)
Verify that your exam has all the pages.
**Only exams written in ink will be eligible for re-marking.**

| 3 /25 | 4 /25 | Total |
|---|---|---|
| | | |

# Question 3: Format String Vulnerability [25 marks]

Suppose you find the following vulnerable sequence in a program as follows (note a double "%%" in the format string tells the format string function to print out a single "%" instead of interpreting a command):

```
sprintf(log_msg, "[%%s:%%u]: %s\n", attack_str);
snprintf(buf, 256, log_msg, filename, line);
```

As an attacker who can control the contents of `attack_str`, your goal is to **write the value `0x01234567`** to the **target location `0xbffffe30`**. You should assume the following:

- There are an additional **8 bytes** between the last argument of the vulnerable format string function and the location of `buf` on the stack.
- Variable `filename` points to the string "`foo.c`" (without the quotes)
- Variable `line` contains `35`.
- Any shellcode you write is 46 bytes in length.
- We are on a 32-bit x86 machine

1) How do you overcome the 8 byte gap between the vulnerable format string function arguments and the location of `buf` on the stack? Be as precise as possible. [5 marks]

2) If you chose to write the target location in one byte at a time, how many writes do you need? To what addresses and in what order should those writes be made? [5 marks]

3) Please draw the contents you would need to put in `attack_str` to achieve the stated goal above. [15 marks]

## Question 4: Cryptography [25 marks]

After watching the cryptography lectures, Crypto Cathy decides to invent her own cipher CCath, which operates as follows:

- The user picks the name of her favorite school mascot and makes a *n*-byte string to use as the key, where *n* is the length of the character's name
- For each character in the plaintext, she converts the corresponding letter in the key to a number (i.e. a=0, b=1, …, z=25, A=26, B=27 … etc…) and adds it to the plaintext character mod 52 to get a new character. For example, a plaintext "d" with a corresponding "c" in the key would yield 'd' + 2 = 'f'
- If she runs out of characters (i.e. the length of the plaintext is greater than *n*) she reverses the key and continues using letters from the reversed key for the next *n* characters. If she runs out of characters, she reverses the key again to get the original key and continues. She alternately uses characters of the key and its reversed key until every character in the plaintext has been substituted.

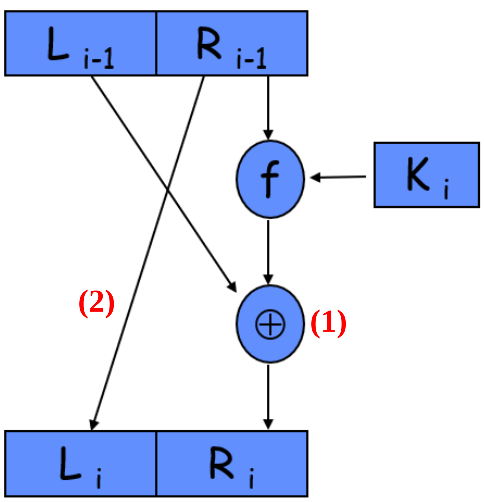
As an example, if the user had selected the name "Godiva", then the sequence of characters she would use to add to the plaintext with would be "GodivaavidoGGodivaavidoG….". The first letter of the plaintext would have 'G' (32) added to it. Answer the following questions about this cipher:

1) What cipher covered in class is this cipher closest to? Please explain. [5 marks]

2) What is the number of keys an attacker would have to try in a brute-force attack? If you cannot estimate it exactly, state any assumptions you need to make and try to be as precise as you can be. [5 marks]

3) What problem(s) do you see in the way the keys are created for this algorithm? [5 marks]

4)




Above is a picture of a stage in a Feistel network. Suppose the XOR at (1) is change to an AND. What would be the consequence of this change? Please explain. [5 marks]

5) Suppose you place a function g() at (2) such that $L_i = g(R_{i-1})$ . What properties should $g$ have ideally? Please explain. [5 marks]