Name: _____

ECE568S: Midterm Exam
Examiner: C. Gibson

DURATION: 110 Minutes

1. Please use a **pen** to complete all of your answers to the midterm.

2. Do not turn this page over until you have received the signal to start.

3. This exam is closed book and closed notes. One 8.5"x11" double-sided aid sheet containing your own personally-created notes is permitted. Use of calculators, computing and/or communicating devices is not permitted. Work independently.

4. Do not remove any sheets from this test book. Answer all questions in the space provided, unless otherwise indicated. No additional sheets are permitted.

5. The value of each question or part-question is noted next to the question. The total of all marks is shown below.

6. Please write your name and student number in the space below, and write your name on the top of each sheet.

First Name: _____

Last Name: _____

Student Number: _____

Q1. _____ / 35 Q3. _____ / 15

Q2. _____ / 20 Q4. _____ / 30

Total: _____ / 100

Name: _____

1. Short-Answer Questions (35 marks)

Please *briefly* answer the following questions, in the space provided. If you require more space, please indicate that you are continuing your answer on the back of the sheet, and put the question number next to the continuation of your answer.

a) If you are running a program under gdb and stopped at a breakpoint, what gdb command would allow you to determine the address of the local array named *buffer*? **(3 marks)**

b) What gdb command would let you print out 80 bytes of memory as a string (*i.e.*, display the contents as a sequence of ASCII characters), starting at address 0x123456? **(4 marks)**

c) You are asked to create a sample exploit that demonstrates a *buffer overflow* vulnerability, similar to your work in Labs 1 and 2. The buffer you are writing into (on the stack) starts at 0x8A269700, on a 32-bit little-endian machine. Explain both the difficulty in overwriting the Return Address with that value (using the buffer overflow technique we used in Labs 1 and 2), and how you would work around that problem. **(4 marks)**

Difficulty:

Work-around:

d) You are given the following shellcode, essentially identical to that used in Labs 1 and 2 (the only difference being that the trap to "exit" has been removed, to simplify this question):

```
jmp 0x1c
popl %esi
movl %esi,0x8(%esi)
movb $0x0,0x7(%esi)
movl $0x0,0xc(%esi)
movl $0xb,%eax
movl %esi,%ebx
leal 0x8(%esi),%ecx
movl 0xc(%esi),%edx
int $0x80
call -0x1a
.string "/bin/sh"
```

You need to modify this code for use on a new system, where the interrupt call is changed to "int $0x00". (This means your shellcode needs to have a "0x00" byte as part of the "int" instruction.) Explain how you could change the shellcode to make this work with a standard buffer overflow, by having the code modify itself during run-time. (You do not need to write the assembly code; just explain the change you would make in words / pseudocode.) **(4 marks)**

e) *Continuing from Part (d), above...* Explain how you could make this work *without* having the code modify itself during run-time. (You do not need to write the assembly code; just explain the change in words or pseudocode, and use a diagram if needed.) **(10 marks)**

f) Alice wishes to send a *confidential* and *signed* message to Bob, using public key cryptography. Bob and Alice each have their own public and private keys. Explain which public/private key Alice uses to sign the message to Bob, and which public/private key Alice uses to encrypt the message. **(4 marks)**

Signing:

Encrypting:

g) Consider a basic SP network, as discussed in class. Explain the purpose of the S-box, the P-box and the key. Do not explain how they are implemented: explain their purpose for existing in the design. **(6 marks)**

S-box:

P-box:

Key:

Name: _____

2. Input Vulnerabilities (20 marks)

a) **(15 marks)** Consider the following program:

```
01 #include <stdio.h>
02 #include <string.h>
03
04 #define BUFFER_LEN 128
05
06 void reverseString(char * src, char * dest)
07 {
08     size_t srcLen = strlen(src);
09     if ( srcLen > BUFFER_LEN ) srcLen = BUFFER_LEN;
10
11     dest[srcLen] = '\0';
12     for ( int i = 0; src[i] != '\0'; i++)
13           dest[srcLen - i - 1] = src[i];
14 }
15
16 int main(int argc, char * argv[])
17 {
18     char buffer[BUFFER_LEN + 1];
19
20     reverseString(argv[0], buffer);
21     printf(buffer);
22     return(0);
23 }
```

When executed, the program prints the contents of argv[0] backwards. The code compiles and runs correctly, but it contains at least two security vulnerabilities. Pick two vulnerabilities and that you could exploit with injected shellcode, and fill in the table below (continues onto the next page):

	Vulnerability #1	Vulnerability #2
Type of vulnerability		
Line number(s) where it appears		
Which function's RA would you attempt to overwrite?		
When would your shellcode execute?		

Name: _____

	Vulnerability #1	Vulnerability #2
How would you rewrite the line(s) to correct the problem?		
Describe, briefly, how you would structure your attack string? (Assume you are using shellcode similar to what you used in Labs 1 and 2.)		
How would you arrange to provide your attack string to the program?		

Name: _____

3. Hashes, MACs and Digital Signatures (15 marks)

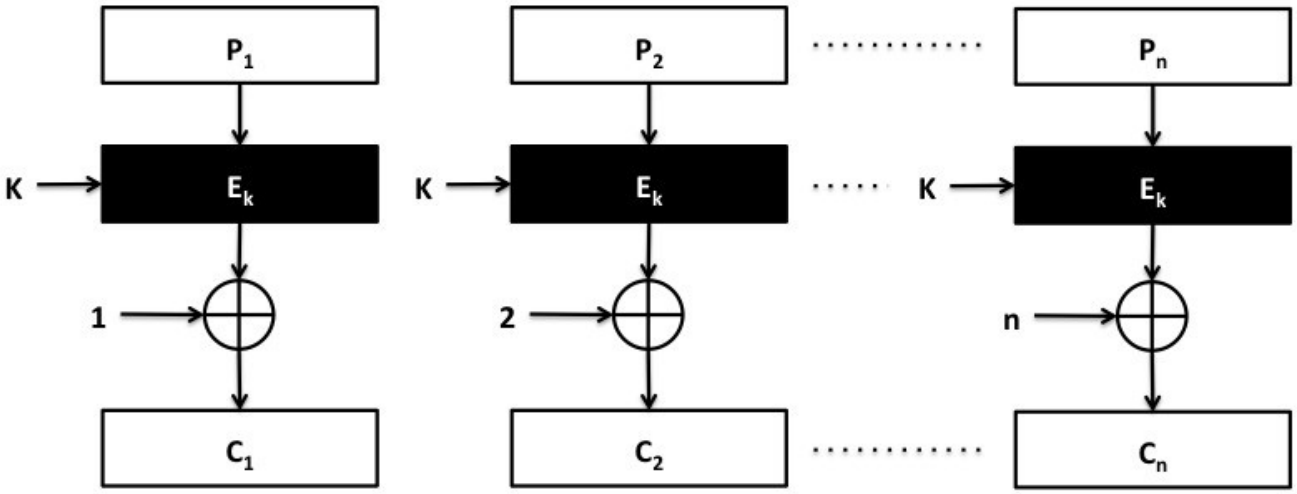
a) **(5 marks)** What special precautions should you take in sending an MDC to a recipient, and why?

b) **(10 marks)** Given a key K and a message M, how is an HMAC computed, and what vulnerability does it solve (*e.g.*, versus a CBC-MAC)?

Name: _____

4. Encryption (30 marks)

a) **(15 marks)** The following block encryption mode is proposed: the output of each encryption round is XOR'ed with a sequentially-increasing integer (n) in order to produce the ciphertext blocks C_n. This is shown below:



If an attacker notices that two encrypted blocks, C_i and C_j, are identical, and is also somehow able to find one corresponding plaintext P_i, can the attacker find the plaintext P_j? If so, explain how. If not, explain how the attacker could find other plaintext blocks that match P_i. (To be clear, the attacker does *not* have the key, K.)

b) **(10 marks)** Alice and Bob have established a secure communication channel using symmetric encryption, and a shared key **K**. Alice sends messages to Bob, and Bob confirms that he has received them by sending them back to Alice. There are no nonces or sequence numbers, which means that an attacker could simply replay the messages back to Alice, and she would think that Bob has received them. Without modifying the messages (*i.e.*, no nonces or sequence numbers) how could you prevent this sort of reflected replay attack while still using symmetric encryption?

c) **(5 marks)** Bob signs his messages with RSA digital signatures. Suppose that Alice can trick Bob into signing messages $\mathbf{m_1}$ and $\mathbf{m_2}$, such that $\mathbf{m = (m_1 \cdot m_2) \bmod n_{Bob}}$. Show that Alice can now forge Bob's signature on the message **m**.