Name: _____

ECE568S: Midterm Exam
Examiner: C. Gibson

DURATION: 90 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: _**Answer Key**_____

Last Name: _____

Student Number: _____

Q1. _____ / 25 Q3. _____ / 16

Q2. _____ / 14

Total: _____ / ~~55~~ **50**

Name: _____

1. Short-Answer Questions (25 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) Most *double-free* vulnerabilities could be eliminated if `free()` simply cleared the contents of the pointer after it freed the allocated memory. Why does `free()` not do this? **(4 marks)**

In "free(p);", the pointer "p" is passed by reference; as a result, it cannot be set to NULL. (If it could set the pointer values to NULL, then it would go a long way to eliminating double-free vulnerabilities.)

b) The first message in the *Needham-Schroeder* exchange is $A \rightarrow T: \{A,B,N_A\}$. From the standpoint of confidentiality, what is the problem with this message? **(2 marks)**

Anyone in the middle can obtain the metadata about the conversation (*e.g.*, "A is talking to B").

c) (*continued...)* Briefly explain how you would change this message to fix the problem. Then show your new message, using the same notation as above (*e.g.*, $A \rightarrow T: \{\_\_\_\_\_\}$). **(6 marks)**

Something like:

$A \rightarrow T: \{A, \{A,B\}_A, N_A\}$

Two key points: (i) T needs to know who it's talking to; and, (ii) the {A,B} portion should be encrypted with a unique key that's been pre-shared between T and A.

d) You are given the following shellcode, essentially identical to that used in Lab #1 (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)**

General idea: leave the "int $0x80" instruction as it is (or with any value other that "$0x00"), and have the code change itself by writing a value of $0x00 into the memory occupied by the instruction.

e) One of your colleagues has developed a messaging protocol for a new product. In order to protect against *reordering* and *replay* attacks, a sequence number is attached to each message:

Seq #: 1	"First message"
Seq #: 2	"Second message"
Seq #: 3	"Third message"
...	...

For each session between the sender and receiver, the sequence number starts at 1 and increments each time a message is sent. The sequence number is a 64-bit number: the sender and receiver will, in practice, never run out of sequence numbers. Does this protocol protect against both types of attack? Explain briefly. **(3 marks)**

The sequence number always starts at 1, so it's known – and the same at the start of every conversation; this does nothing to stop replay attacks. Having a sequence number does at least stop reordering attacks *within the same message*.

f) The `snprintf()` function allows developers to limit the amount of data written to buffer:

```
int snprintf(char * buffer, size_t size, const char * format, ...);
```

Briefly explain whether specifying a proper "size" limit will prevent vulnerabilities involving *information leakage*, *buffer overflows* and *format string attacks*. **(3 x 2 marks = 6 marks)**

Information leakage: No. An attacker can provide more "%___" commands in for format string than parameters.

Buffer overflows: Yes. If used properly, the "size" parameter will prevent "buffer" from being overflowed.

Format string attacks: No. Even with "size" set to zero, the entire format string will still be evaluated.

2. Input Vulnerabilities (14 marks)

a) Your company asks you to conduct a penetration test on their new virtual-server environment. You quickly discover a buffer overflow vulnerability, but the stack is in a memory region that is marked NX (Non-eXecutable), so you cannot inject any shellcode. However, you discover another vulnerability: there are virtual machine images sitting in memory in an encrypted state, which effectively gives you many, many gigabytes of random data... and, due to a bug, that memory is marked as executable. Assuming your ideal shellcode is very small (similar to Lab #1), what approach *might* allow you to construct an attack, using only a buffer overflow and this random data? (*Hint: consider both your work in Lab #1 and Assignment #1*.)
(5 marks)

General idea: The virtual machine images (essentially a large bulk of random data) will have sequences that happen to match portions of the valid instructions you are looking for (in the same way as Return-Oriented Programming). Use the same technique as ROP to construct a series of "return" calls that will execute your desired program.

b) Consider a *Double-Free* attack (as discussed in class) on a 32-bit application whose memory manager uses these tags:

tag.previous	tag.metadata	tag.next
32-bits	32-bits	32-bits

Assume that "q" is the pointer that is accidentally freed a second time. When "`free(q);`" is called during the attack, the memory manager executes:

```
q->prev->next = q->next;
q->next->prev = q->prev;
```

Briefly explain how you would construct your fake tag (be specific about what "previous" and "next" should point to). Also explain what you would need to do in the start of your shellcode, in order for this to work. **(9 marks)**

1) You need to create a tag where "tag.previous" points to your shellcode and "tag.next" points to your saved Return Address.

2) Some bytes near the start of your shellcode (corresponding to where "prev" would be located) will be overwritten. You will need to start your shellcode with a relative "jump" that will skip over the overwritten values.

Name: _____

3. Hashes, MACs and Public Key (16 marks)

a) Consider the following two ways of using hashes to protect a message ("M"). For each case, assume that "buffer" will be sent over an insecure channel, and comment briefly on how well the approach will address *confidentiality*, *integrity* and *authentication*. (Assume that "+" will perform string concatenation, and that "key" has been previously shared via a secure means.) **(2 x 3 marks = 6 marks)**

	buffer = M + SHA1(M);	**buffer = M + SHA1(key + M);**
Confidentiality?	No: message is not encrypted.	No: message is not encrypted.
Integrity?	Yes (except appending attacks)	Yes (except appending attacks)
Authentication?	No: anyone, including an attacker, can calculate a new hash value if they wish to modify the message.	Yes: only the sender and receiver know "key" -- and, thus, they are the only ones who can generate a valid hash.

b) Using the same notation as above, indicate how you would construct an HMAC. Include the definitions for any additional values/variables you need. **(5 marks)**

You need to propose some key scheduling proposal, to create (k_1, k_2) from your key. The general form of the HMAC is then something similar to:

$$SHA1(SHA1(M + k_1) + k_2)$$

c) Consider a Diffe-Hellman public key exchange, where Alice calculates $P = g^x \bmod n$ and Bob calculates $Q = g^y \bmod n$. Assuming that Alice starts the exchange, what values does Alice send Bob? What values does Bob send Alice? What value is their shared secret at the end of the exchange? **(5 marks)**

- **Alice -> Bob:** P, g, n
- **Bob -> Alice:** Q
- **Shared:** g^{xy} mod n (or g^{yx} mod n)