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. 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 70.

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. _____ / 13 Q3. _____ / 8

Q2. _____ / 35 Q4. _____ / 14

Total: _____ / 70

Name: _____

1. Short-Answer Questions (13 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) What `gdb` command will report information on the current stack frame (*e.g.*, the location of the saved Return IP, etc.)? **(3 marks)**

b) List two advantages of stream ciphers over block ciphers. **(2 marks)**

c) Briefly explain the function of a *stack canary*, and identify a type of exploit discussed in class that stack canaries would **not** protect against. **(2 marks)**

d) Briefly explain the purpose of the PLT/GOT and how an attacker might exploit them. **(2 marks)**

e) The shellcode that we are using in Lab #1 and Lab #2 starts by finding the address of its variables (*e.g.*, the location of the string "/bin/sh"). Briefly describe how the shellcode accomplishes this. (A small diagram may help make your answer shorter.) **(4 marks)**

Name: _____

2. Code Vulnerabilities (35 marks)

The following programs each run as described, but each has one or more security vulnerabilities that can be exploited using techniques we've discussed in class. All programs compile and run as described.

a) (10 marks) Consider the following program:

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int
05  main ( int argc , char * argv[] )
06  {
07       // Print the program name
08       printf ( "Program name: " );
09       printf ( argv[0] );
10
11       // Print the current date and time
12       printf ( "\nCurrent date is: " );
13       fflush ( stdout );
14       system ( "date" );
15
16       return (0);
17  }
```

When executed, the program prints its name and the current date/time:

```
$ question2a
Program name: question2a
Current date is: Mon  5 Mar 2012 10:10:00 EST
$
```

Briefly describe the security vulnerability (or vulnerabilities) in the above code.

Program Line(s)	What approach would an attacker use to exploit it?

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

```
01 #include <stdio.h>
02 #include <string.h>
03 #include <ctype.h>
04
05 #define BUFFER_LEN 10
06
07 // A function to convert a string to uppercase letters
08
09 int
10 upshift ( const char * src , char * dest , const short max_dest_len )
11 {
12       int   i;
13       short src_len;
14
15       // How long is our source string?
16       src_len = strlen ( src );
17       src_len += 1;  // Don't forget the '\0'
18
19       // Make sure we won't overflow dest
20       if ( src_len > max_dest_len ) return (-1);
21
22       // Copy src into dest, upshifting all of the chars
23       for ( i = 0 ; i <= src_len ; i++ )
24             dest[i] = toupper ( src[i] );
25
26       return (0);
27 }
28
29 int
30 main ( int argc , char * argv[] )
31 {
32       char  buffer[BUFFER_LEN];
33
34       // Make sure the user provided a command-line argument
35
36       if ( argc == 2 ) {
37
38             int rc;
39
40             // Upshift the command-line argument and print it
41
42             rc = upshift ( argv[1] , buffer , BUFFER_LEN );
43             if ( rc == 0 ) printf ( "%s\n", buffer );
44       }
45
46       return (0);
47 }
```

When executed, the program upshifts and prints the command-line argument, if it's not too long:

```
$ question2b hello
HELLO
$ question2b stringThatIsTooLong
$
```

Briefly describe the security vulnerability (or vulnerabilities) in the above code.

Program Line(s)	Type of Vulnerability	What approach would an attacker use to exploit it?

c) **(10 marks)** The following program implements a basic password-checking function. It takes a username and password as command-line arguments, and checks to see if they're valid:

```
$ question2c basicuser basicPass
The username and password are valid

$ question2c root X34j9Lm0
The username and password are valid
```

If an *invalid* username or password is provided, it reports this to the user:

```
$ question2c ece568 invalidPassword
You have entered an invalid username or password
```

As a special case, though, it will return a specific error message if you try to guess the "root" account password:

```
$ question2c root invalidPassword
ALERT: YOU'RE TRYING TO HACK THE SYSTEM!
```

Consider the following program code, which implements this functionality:

Name: _____

```
01  #include <stdio.h>
02  #include <string.h>
03
04  #define MAX_USERS          3
05  #define MAX_USERNAME_LEN   9
06  #define MAX_PASSWORD_LEN   9
07
08  struct {
09
10        char   password[MAX_PASSWORD_LEN];
11        char   username[MAX_USERNAME_LEN];
12
13  } users[MAX_USERS];
14
15  void initialize ()
16  {
17        // Create some user records
18
19        strncpy ( users[0].username, "ece568", MAX_USERNAME_LEN );
20        strncpy ( users[0].password, "8IlM2Q", MAX_PASSWORD_LEN );
21
22        strncpy ( users[1].username, "basicuser", MAX_USERNAME_LEN );
23        strncpy ( users[1].password, "basicPass", MAX_PASSWORD_LEN );
24
25        strncpy ( users[2].username, "root", MAX_USERNAME_LEN );
26        strncpy ( users[2].password, "X34j9Lm0", MAX_PASSWORD_LEN );
27  }
28
29  void checkPassword ( char * username , char * password )
30  {
31        int    u;
32
33        // Check all of the user records, to see if the username/password match
34
35        for ( u = 0 ; u < MAX_USERS ; u++ )
36        {
37              int valid_username =
38                    !strncmp ( username, users[u].username, strlen(username) );
39              int valid_password =
40                    !strncmp ( password, users[u].password, strlen(password) );
41
42              if ( valid_username && valid_password ) {
43
44                    printf ( "The username and password are valid\n" );
45                    return;
46              }
47        }
48
49        // If we've arrived here, then we didn't find a match
50
51        if ( !strcmp ( username, "root" ) )
52
53              printf ( "ALERT: YOU'RE TRYING TO HACK THE SYSTEM!\n" );
54        else
55              printf ( "You have entered an invalid username or password\n");
56  }
57
58  int main ( int argc , char * argv[] )
59  {
60        initialize ();
61
62        if ( argc == 3 ) checkPassword ( argv[1], argv[2] );
63
64        return (0);
65  }
```

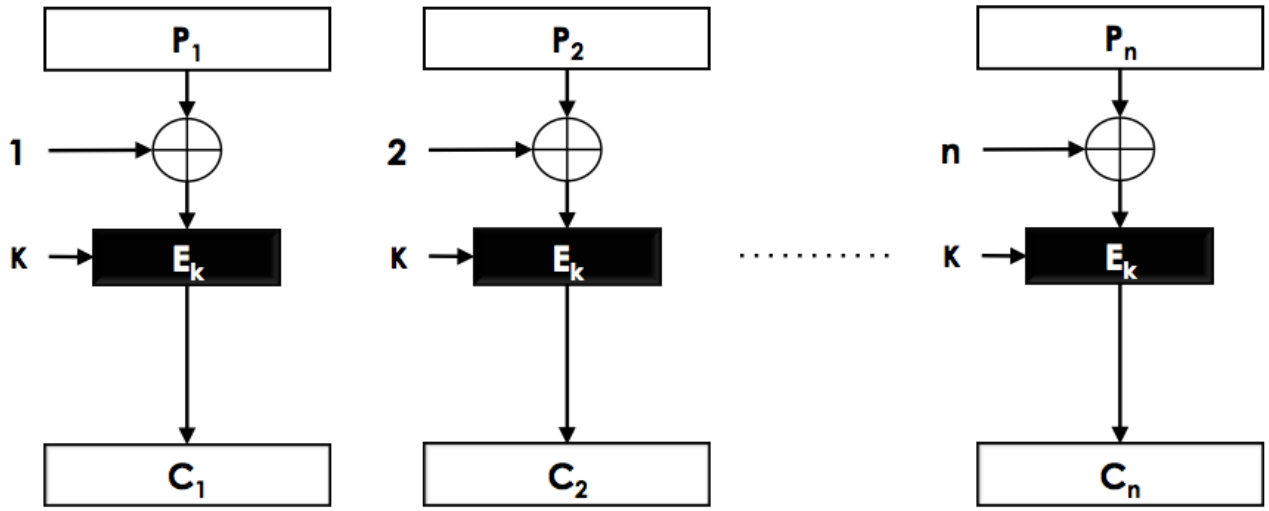
Briefly describe how an attacker could run this program repeatedly, and eventually discover the password to the "root" user, without ever causing the "ALERT: YOU'RE TRYING TO HACK THE SYSTEM!" message to appear. (Note: to save you time analyzing the code, the solution does not involve stack smashing or code injection.)

Name: _____

3. Symmetric Encryption (8 marks)

a) **(2 marks)** Claude Shannon's paper, *Communication Theory of Secrecy Systems*, proposes two desirable goals for a good cryptosystem: **confusion** and **diffusion**. Briefly explain the purpose of the **confusion** goal.

b) **(6 marks)** The following encryption mode is proposed: each plain text block (P_n) is XOR'ed with a sequentially-increasing integer (n), before being encrypted with a standard block cipher. This is shown below:



If an attacker notices that two encrypted blocks, C_i and C_j, are identical, and is also able to find the corresponding plaintext P_i, can the attacker find the plaintext P_j? If so, explain how. If not, explain why not.

Name: _____

4. Public Key Encryption (14 marks)

a) **(3 marks)** Describe the steps, in point form, that are required to generate a **self-signed** X.509 public-key certificate.

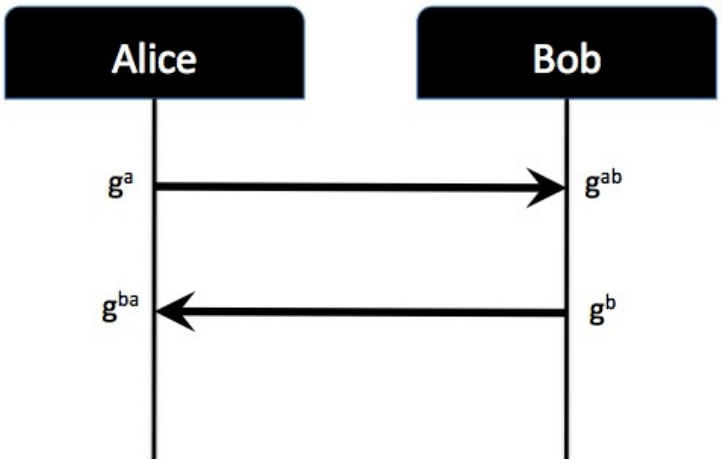
b) **(3 marks)** During the *Handshake* phase of the SSL protocol, the client and server exchange three pieces of data that are used to compute a shared secret key. Explain what those three values are and how they are exchanged, by completing the table below:

Description of value	Who sends this value?	Is it encrypted?	If so, how is it encrypted?

c) Recall that a Diffie-Hellman key exchange between **two parties** works as follows:

- *Alice* picks a random number, **a**
- *Bob* picks a random number, **b**
- *Alice* sends the value $\mathbf{g^a}$ to *Bob* (who can then compute $\mathbf{g^{ab}}$)
- *Bob* and sends the value to $\mathbf{g^b}$ to *Alice* (who can then compute $\mathbf{g^{ba}}$)

This exchange is shown in the diagram, below:



Both parties then share the secret value:

$$\mathbf{g^{ab} = g^{ba}}$$

which they can use to encrypt their conversation. (To simplify things, assume that both parties have already exchanged the generator, **g**, and all math is done modulo **n**.)

(continued on next page...)

Now, consider the case where **three parties** (*Alice*, *Bob* and *Charlie*) wish to have an encrypted conversation (*e.g.*, all three share the same secret key, so that if any of them sends a message, *Alice*, *Bob* and *Charlie* can all decrypt it). The Diffie-Hellman key exchange can be extended to allow all three parties to choose one secret key, without an attacker being able to determine its value. As above, assume the values **g** and **n** are already shared, and the exchange starts with:

- *Alice* picking a random number, **a**
- *Bob* picking a random number, **b**
- *Charlie* picking a random number, **c**

(2 marks) What is the shared secret value that *Alice*, *Bob* and *Charlie* will each have computed at the end of the exchange?

(6 marks) Using the same notation as on the previous page, complete the diagram below to show the messages, in order, that need to be exchanged in order for *Alice, Bob* and *Charlie* to all compute this value. (Reminder: you can't send the secret value directly, or an attacker will be able to read it!)