Question 1: Multiple Choice . . . . *12 points*

For each question, circle *all* that apply. Some questions may have more than one correct answer. For every wrong answer, you will lose 1 point, but the lowest score you can get for any part (e.g., 1a, 1b) is 0.

(a) (2 points) Alice clicks a link on a malicious website that was able to successfully initiate a money transfer from Alice's bank. What defense(s) could the bank have employed to prevent this from happening?

- A. Require that a valid session token be present in the URL of every transaction
- B. Use prepared statements for SQL queries
- C. Reauthenticate the user each time they want to transfer money
- D. Adhering to the Same Origin Policy is sufficient

(b) (2 points) What assurance(s) does the Same Origin Policy provide?

- A. Attackers are prevented from installing malware on a user's machine
- B. Websites are prevented from from using third-party widgets on their pages
- C. Users are prevented from downloading malicious video codecs
- D. Websites are prevented from reading the information from other sites

(c) (2 points) When Professor Kumar was preparing these midterm sample questions, he encrypted the exam files before saving them to disk. What aspect of "Thinking Like a Defender" is this an example of?

- A. Security Policy
- B. Threat Model
- C. Risk Assessment
- D. Countermeasure

(d) (2 points) Which of the following functions are secure against buffer overflow attacks? Assume that the function was used with the correct parameters.

- A. strcpy
- B. strncpy
- C. printf
- D. scanf
- E. memcpy

(e) (2 points) Which of the following techniques could potentially prevent (or mitigate) a Return Oriented Programming attack?

- A. Prepared Statements
- B. Data Execution Prevention
- C. Stack Canary
- D. Signature-based Anti-Virus Software
- E. ASLR

(f) (2 points) Say you are calling the following function in x86 (signature shown):

```
void add(int a, int b, int c);
```

Where is parameter `a` located on the stack **immediately** after the `call` to `add`?

- A. ebp + 4
- B. ebp + 8
- C. ebp + 16
- D. esp + 4
- E. esp + 8
- F. esp + 16

Question 2: Short Answer . . . . *8 points*

(a) (3 points) List two forms of code injection. Pick one and describe a defense for it.

(b) (3 points) Explain what a **stack canary** is and how it prevents a buffer overflow attack.

(c) (2 points) Explain what **ASLR** is and why it is useful.

Question 3: AppSec MP Question . . . . *8 points*

(a) Consider the following vulnerable function, compiled and run without DEP or ASLR enabled:

```
void vuln(char *arg)
{
  ...
  int a;
  int *b;
  int c;
  char msg[32];

  strncpy(msg, arg, sizeof(msg) + 8);

  a = c + 4;
  *b = a;
  return;
}
```

i. (5 points) Assume that `arg` is a pointer to a single command line argument you will pass using a Python script. Assume the compiler does not add any padding, and that the following values apply:

- Original return address of `vuln`: `0x08041234`
- Original return address of `vuln` is stored on the stack at: `0xbfffef44`
- Address of the beginning of the `msg` buffer on the stack: `0xbfffeeec`
- Length of `shellcode`: 23 bytes

Fill in the blanks to construct an attack that exploits this function to open a shell.

```
from shellcode import shellcode
from struct import pack


print shellcode + 'A' * _____ + pack('<I', __________________ ) +

     pack('<I', __________________ )
```

ii. (3 points) Suppose that the address of the beginning of the `msg` buffer was `0xbfff0504` instead (and addresses of other variables all shift accordingly). Would you still be able to conduct an attack? If not, why not? If so, describe how you would have to change the attack structure.