Question 1 ***Hacked EvanBot*** ()

Hacked EvanBot is running code to violate students' privacy, and it's up to you to disable it before it's too late!

```
1  #include <stdio.h>
2
3  void spy_on_students(void) {
4      char buffer[16];
5      fread(buffer, 1, 24, stdin);
6  }
7
8  int main() {
9      spy_on_students();
10     return 0;
11 }
```

The shutdown code for Hacked EvanBot is located at address `0xdeadbeef`, but there's just one problem—Bot has learned a new memory safety defense. Before returning from a function, it will check that its saved return address (rip) is not `0xdeadbeef`, and throw an error if the rip is `0xdeadbeef`.

Clarification during exam: Assume little-endian x86 for all questions.

Assume all x86 instructions are 8 bytes long. Assume all compiler optimizations and buffer overflow defenses are disabled.

The address of `buffer` is `0xbffff110`.

Q1.1 (3 points) In the next 3 subparts, you'll supply a malicious input to the `fread` call at line 5 that causes the program to execute instructions at `0xdeadbeef`, *without* overwriting the rip with the value `0xdeadbeef`.

The first part of your input should be a single assembly instruction. What is the instruction? x86 pseudocode or a brief description of what the instruction should do (5 words max) is fine.

Q1.2 (3 points) The second part of your input should be some garbage bytes. How many garbage bytes do you need to write?

◯ (G) 0 ◯ (H) 4 ◯ (I) 8 ◯ (J) 12 ◯ (K) 16 ◯ (L) ——

Q1.3 (3 points) What are the last 4 bytes of your input? Write your answer in Project 1 Python syntax, e.g. `\x12\x34\x56\x78`.

Q1.4 (3 points) When does your exploit start executing instructions at `0xdeadbeef`?

- ◯ (G) Immediately when the program starts
- ◯ (H) When the `main` function returns
- ◯ (I) When the `spy_on_students` function returns
- ◯ (J) When the `fread` function returns
- ◯ (K) ——
- ◯ (L) ——

Question 2 ***C Memory Defenses*** **()**

Mark the following statements as True or False and justify your solution. Please feel free to discuss with students around you.

1. Stack canaries completely prevent a buffer overflow from overwriting the return instruction pointer.

2. A format-string vulnerability can allow an attacker to overwrite values below the stack pointer

3. An attacker exploits a buffer overflow to redirect program execution to their input. This attack no longer works if the data execution prevention/executable space protection/NX bit is set.

4. If you have a non-executable stack and heap, buffer overflows are no longer exploitable.

5. If you use a memory-safe language, some buffer overflow attacks are still possible.

6. ASLR, stack canaries, and NX bits all combined are insufficient to prevent exploitation of all buffer overflow attacks.

Short answer!

1. What vulnerability would arise if the canary was above the return address?

2. What vulnerability would arise if the stack canary was between the return address and the saved frame pointer?

3. Assume ASLR is enabled. What vulnerability would arise if the instruction **jmp *esp** exists in memory?

Question 3 ***Pointer Authentication Codes (PACs)*** **()**

Suppose we are on a 64-bit system, and we have an address space of 2^{50} bytes.

For each of the following questions, provide a short answer and justify your response. Please feel free to discuss with students around you.

1. How many unused bits are available for pointer authentication in each address?

Regardless of your answer to the previous part, for the remainder of the questions, assume that 10 bits are used for pointer authentication in each address and the attacker does not have the ability to create their own pointer authentication codes (PACs).

2. Assume that 64-bit stack canaries are enabled and that the first *two* bytes of the stack canary are always null. How many bits does the attacker have to guess correctly to guess the stack canary and the PAC?

Now assume that the attacker has a format string vulnerability that lets them read any part of memory while the program is running.

3. How many bits does the attacker have to guess correctly to guess the stack canary and the PAC?

4. Suppose the attacker is interacting with a remote system. Provide at least one defense that would make brute-force attacks infeasible for the attacker.