# CS161 Fall 2026 — Introduction to Computer Security — Exam Prep 10

## Q1 *Memory Safety: EvanBond, Double-O \x90* **(19 points)**

Consider the following vulnerable C code:

```
1  void q(int spectre) {
2      char goldfinger[8];
3      fread(goldfinger, 13, 1, stdin);
4      for (int i = 0; i < 8; i++) {
5          goldfinger[i] = 0x90;
6      }
7  }
8
9  void m() {
10     q(007);
11 }
```

**Stack at Line 2**

| |
|---|
| RIP of `m` |
| (1) |
| (2) |
| (3) |
| SFP of `q` |
| (4) |

Assumptions:

- All memory safety defenses are disabled.
- You run GDB and break before executing line 3. You find that the RIP of `q` has the value `0xffffd6c0`.
- You also find that `goldfinger` is located at address `0xffffd620`.
- Recall that the byte `0x90` is also known as the NOP (no-operation) instruction.
- Your goal is to execute the 4-byte-long SHELLCODE.

Q1.1 (2 points) What values go in blanks (1) through (4 ) in the stack diagram above?

- ◯ (1) SFP of `m` (2) RIP of `q` (3) `spectre` (4) `goldfinger`
- ◯ (1) SFP of `m` (2) `spectre` (3) RIP of `q` (4) `goldfinger`
- ◯ (1) RIP of `q` (2) `spectre` (3) SFP of `m` (4) `goldfinger`
- ◯ (1) RIP of `q` (2) `goldfinger` (3) `spectre` (4) SFP of `m`

Q1.2 (4 points) Which of these values does the exploit have to overwrite to execute `SHELLCODE`? Select all that apply.

- ☐ `goldfinger`
- ☐ SFP of `m`
- ☐ Least significant byte of the RIP of `q`
- ☐ `spectre`
- ☐ SFP of `q`
- ◯ None of the above



*This content is protected and may not be shared, uploaded, or distributed.*

Q1.3 (4 points) Provide an input to the `fread` on Line 3.

If a part of the input can be any non-zero value, use `'A' * n` to represent `n` bytes of garbage.

Q1.4 (2 points) Which memory safety defenses would cause the correct exploit from Q7.3 (without modifications) to fail? Consider each choice independently.

- ☐ Stack canaries
- ☐ Non-executable pages
- ◯ None of the above

Q1.5 (3 points) Which **values** of the RIP of `q` would cause the correct exploit from Q7.3 (without modifications) to fail? Select all that apply.

- ☐ `0xffffd640`
- ☐ `0xffffd630`
- ☐ `0xffffd620`
- ☐ `0xffffd610`
- ☐ `0xffffd600`
- ☐ `0xffffdbe0`
- ☐ `0xffffdbd0`
- ☐ `0xffffdbc0`
- ◯ None of the above

Q1.6 (4 points) When does the correct exploit from Q7.3 start executing instructions in `SHELLCODE`?

- ◯ When the function `fread` on line 3 returns
- ◯ When the `for` loop on line 4 completes
- ◯ When the function `q` on line 1 returns
- ◯ When the function `m` on line 8 returns



*This content is protected and may not be shared, uploaded, or distributed.*

## Q2 *Memory Safety: Are you being fr ?* **(17 points)**

Consider the following vulnerable C code:

```
1  void foo(void) {
2      char msg[8];
3      fgets(msg, 8, stdin);
4      printf(msg);
5      fread(msg, 20, 1, stdin);
6  }
7
8  void vulnerable() {
9      int i;
10     char buf[20];
11     char cpy[20];
12     fread(buf, 20, 1, stdin);
13     for(i = 19; i >= 0; i--) {
14         cpy[i] = buf[19-i];
15     }
16     foo();
17 }
```

**Stack at Line 2**

| |
|---|
| RIP of `vulnerable` |
| (1) |
| (2) |
| `i` |
| (3) |
| `cpy` |
| (4) |
| (5) |
| (6) |
| `msg` |

Assumptions:

- Stack canaries and non-executable pages are enabled, and all other memory safety defenses are disabled.
- You run GDB once and find that the library function `system` is located at the address `0xdeadbeef`.
- The RIP of `foo` is located at `0xffffdc90`. The RIP of `vulnerable` is located at `0xffffdcc8`.
- Your goal is to execute `system` with the 8-character string `"rm -rf /"` as the argument.

Q2.1 (1 point) What values go in blanks (1) through (3) in the stack diagram above?

- ◯ (1) canary (2) RIP of `foo` (3) SFP of `vulnerable`
- ◯ (1) canary (2) SFP of `vulnerable` (3) `buf`
- ◯ (1) SFP of `vulnerable` (2) canary (3) `buf`
- ◯ (1) SFP of `vulnerable` (2) `cpy` (3) canary

Q2.2 (1 point) What values go in blanks (4) through (6) in the stack diagram above?

- ◯ (4) canary (5) RIP of `foo` (6) SFP of `foo`
- ◯ (4) canary (5) SFP of `foo` (6) RIP of `foo`
- ◯ (4) RIP of `foo` (5) SFP of `foo` (6) canary
- ◯ (4) RIP of `foo` (5) `&msg` (6) SFP of `foo`



*This content is protected and may not be shared, uploaded, or distributed.*

Q2.3 (2 points) What type of vulnerability is present in this code? Select all that apply.

- ☐ Format string vulnerability
- ☐ Off-by-one
- ☐ ret2ret
- ☐ Buffer overflow
- ☐ Signed/unsigned
- ○ None of the above

Q2.4 (3 points) Which of these inputs to **`fgets`** on Line 3 will always leak the value of the stack canary in the **`foo`** stack frame? Select all that apply.

*Note: You are able to convert any numerical representation of the canary into a usable form.*
*Note: Stack canaries are four random bytes (no null byte).*

- ☐ `'%x' * 3`
- ☐ `2 * '%c' + '%p'`
- ☐ `'%n' * 3`
- ☐ `'\xc0\xdc\xff\xff' + '%s'`
- ☐ `'\x88\xdc\xff\xff' + '%s'`
- ○ None of the above

In the next two subparts, provide inputs that would cause the program to execute **`system("rm -rf /")`**. You may use **`CANARY`** to refer to the correct value of the stack canary, as leaked by **`printf`**.

Q2.5 (5 points) Input to **`fread`** at Line 12:

```
'A'*[          ] + '\x00' + '[                        ]' +
'[                                  ]' + 'A'*[          ]
```

Q2.6 (5 points) Input to **`fread`** at Line 5:

## Q3 *Cryptogrpahpy: Obliviously Garbage* **(11 points)**

Alice has two messages: $m_0$ and $m_1$. Bob wants to retrieve one of the two messages, without Alice finding out which message Bob chose to retrieve.

To do this, Alice and Bob follow the *blind retrival* protocol below:

**Setup:**

1. Alice generates an RSA key pair: public key $(N, e)$ and private key $d$. Alice sends $(N, e)$ to Bob.
2. Alice generates two random values $r_0$ and $r_1$ and sends them to Bob.
3. If Bob chooses $m_0$, he will define $r_b = r_0$. Otherwise, he will define $r_b = r_1$.

**Protocol Steps:**

4. Bob generates a random value $k$.
5. Bob computes $v \equiv r_b + k^e \bmod N$ and sends this value $v$ to Alice.



*This content is protected and may not be shared, uploaded, or distributed.*

6. Alice computes $k_0 \equiv$ _____ [Q3.1] and $k_1 \equiv$ _____ [Q3.2].

7. Alice sends $m_0' \equiv m_0 + k_0 \bmod N$ and $m_1' \equiv m_1 + k_1 \bmod N$ to Bob.

8. Bob recovers his desired message by computing $m_b \equiv$ _____ [Q3.3].

Q3.1 (1 point) Provide the value for $k_0$ in step 6.

- ◯ $k_0 \equiv (v + r_0)^d \bmod N$
- ◯ $k_0 \equiv (v \cdot r_0)^d \bmod N$
- ◯ $k_0 \equiv (v - r_0)^d \bmod N$
- ◯ $k_0 \equiv v^d - r_0 \bmod N$

Q3.2 (1 point) Provide the value for $k_1$ in step 6.

- ◯ $k_1 \equiv (v + r_1)^d \bmod N$
- ◯ $k_1 \equiv (v \cdot r_1)^d \bmod N$
- ◯ $k_1 \equiv (v - r_1)^d \bmod N$
- ◯ $k_1 \equiv v^d - r_1 \bmod N$

Q3.3 (1 point) Provide the value for $m_b$ in step 8.

- ◯ $m_b \equiv m_b{}' + k \bmod N$
- ◯ $m_b \equiv m_b{}' \oplus k$
- ◯ $m_b \equiv m_b{}' - k \bmod N$
- ◯ $m_b \equiv m_b{}' \cdot k^{-1} \bmod N$

Q3.4 (1 point) Why can Alice not determine which message Bob chose to retrieve? Select the best answer.

- ◯ Because the value $v = r_b + k^e \bmod N$ is masked by the term $k^e$.
- ◯ Because $r_0, r_1$ are both randomly generated and therefore evenly distributed mod $N$.
- ◯ Because Bob's private RSA exponent $d$ remains secret.
- ◯ Because Bob sends $v$ over an encrypted channel, so Alice cannot read it directly.



*This content is protected and may not be shared, uploaded, or distributed.*

**Subparts Q3.5 to Q3.8 are independent of earlier subparts.**

Consider this protocol:

- Alice and Bob each have a secret bit (Alice has $a$, Bob has $b$).
- They want to compute the bitwise AND of their secret bits, such that both parties learn $a \wedge b$.
- Alice and Bob should not learn each other's secret bit (except what can be inferred: see note below).
- *Note:* Sometimes you can infer the other person's bit from the $a \wedge b$ output, and it's okay if the protocol leaks this information. For example, if Bob picks $b = 1$ and sees $a \wedge b = 0$, he can infer that $a = 0$. However, if Bob picks $b = 0$ and sees $a \wedge b = 0$, he cannot infer $a$ (could be 0 or 1).

**Protocol**:

1. Alice generates four random symmetric keys: $K_{a=0}, \quad K_{a=1}, \quad K_{b=0}, \quad K_{b=1}$.
2. Alice uses the symmetric keys to compute four ciphertexts:

$$\mathsf{Enc}(K_{a=0},\ \mathsf{Enc}(K_{b=0}, 0))$$
$$\mathsf{Enc}(K_{a=0},\ \mathsf{Enc}(K_{b=1}, 0))$$
$$\mathsf{Enc}(K_{a=1},\ \mathsf{Enc}(K_{b=0}, 0))$$
$$\mathsf{Enc}(K_{a=1},\ \mathsf{Enc}(K_{b=1}, 1))$$

3. Alice sends all four ciphertexts to Bob.
4. Let $K_a$ be $K_{a=0}$ or $K_{a=1}$, depending on which bit Alice chose. Alice sends $K_a$ to Bob.
5. Let $K_b$ be $K_{b=0}$ or $K_{b=1}$, depending on which bit Bob chose. Bob retrieves $K_b$ from Alice.
6. For each of the four ciphertexts, Bob evaluates _____[Q3.5].

Three of the ciphertexts will decrypt to garbage. One of the ciphertexts will decrypt to either 0 or 1. The desired output $a \wedge b$ is the non-garbage value.

Q3.5 (1 point) Fill in the blank for step 6 above.

Your answer may include $\mathsf{Enc}$, $\mathsf{Dec}$, $K_a$, $K_b$, and $C$ (one of the four ciphertexts Alice sends).

Q3.6 (1 point) In step 3, should Alice send the four ciphertexts in a random order?

- ◯ Yes, to prevent Bob from using the ciphertext order to always deduce Alice's bit $a$.
- ◯ Yes, to ensure each ciphertext uses a different encryption key.
- ◯ No, because Bob can already decrypt the non-garbage ciphertext.
- ◯ No, because encryption alone prevents Bob from always deducing Alice's bit $a$.



*This content is protected and may not be shared, uploaded, or distributed.*

**Protocol** (reprinted for your convenience):

1. Alice generates four random symmetric keys: $K_{a=0}$, $K_{a=1}$, $K_{b=0}$, $K_{b=1}$.
2. Alice uses the symmetric keys to compute four ciphertexts:

$$\mathsf{Enc}(K_{a=0},\ \mathsf{Enc}(K_{b=0}, 0))$$
$$\mathsf{Enc}(K_{a=0},\ \mathsf{Enc}(K_{b=1}, 0))$$
$$\mathsf{Enc}(K_{a=1},\ \mathsf{Enc}(K_{b=0}, 0))$$
$$\mathsf{Enc}(K_{a=1},\ \mathsf{Enc}(K_{b=1}, 1))$$

3. Alice sends all four ciphertexts to Bob.
4. Let $K_a$ be $K_{a=0}$ or $K_{a=1}$, depending on which bit Alice chose. Alice sends $K_a$ to Bob.
5. Let $K_b$ be $K_{b=0}$ or $K_{b=1}$, depending on which bit Bob chose. Bob retrieves $K_b$ from Alice.
6. For each of the four ciphertexts, Bob evaluates _____[Q3.5].

Three of the ciphertexts will decrypt to garbage. One of the ciphertexts will decrypt to either 0 or 1. The desired output $a \wedge b$ is the non-garbage value.

Q3.7 (1 point) Suppose that in Step 5, Bob retrieves $K_b$ by telling Alice: "I want $K_{b=0}$" or "I want $K_{b=1}$". Why is this a bad idea?

- ◯ Because that would reveal Bob's bit $b$ to Alice.
- ◯ Because Bob would not have the key required to decrypt the ciphertexts.
- ◯ Because $K_{b=0}$ and $K_{b=1}$ are both generated as a function of Bob's bit $b$.
- ◯ Because that would reveal Alice's bit $a$ to Bob.

Q3.8 (2 points) Suppose that in Step 5, Bob retrieves $K_b$ by asking for both keys. This is a bad idea because Bob can now reveal Alice's bit $a$.

Which expression, when evaluated on each ciphertext $C$, will reveal Alice's bit $a$?

*Note: The version of this subpart that appeared on the exam erroneously flipped the order of decryption keys. This was clarified, and has been corrected for this version of the exam.*

- ◯ $\mathsf{Dec}(K_b,\ \mathsf{Dec}(K_{a=0}, C))$
- ◯ $\mathsf{Dec}(K_b,\ \mathsf{Dec}(K_{a=1}, C))$
- ◯ $\mathsf{Dec}(K_{b=0},\ \mathsf{Dec}(K_a, C))$
- ◯ $\mathsf{Dec}(K_{b=1},\ \mathsf{Dec}(K_a, C))$



*This content is protected and may not be shared, uploaded, or distributed.*

Q3.9 (2 points) Suppose that in Step 5, Bob uses the *blind retrieval* protocol (from earlier in this question) to retrieve either $K_{b=0}$ or $K_{b=1}$ from Alice, without Alice knowing which one Bob chose to retrieve.

After the blind retrieval in step 5, which values are known to Bob? Select all that apply.

- ☐ $K_a$, the key $K_{a=0}$ or $K_{a=1}$ corresponding to the bit Alice chose.
- ☐ $K_b$, the key $K_{b=0}$ or $K_{b=1}$ corresponding to the bit Bob chose.
- ☐ The key $K_{a=0}$ or $K_{a=1}$ corresponding to the bit Alice did *not* choose.
- ☐ The key $K_{b=0}$ or $K_{b=1}$ corresponding to the bit Bob did *not* choose.
- ☐ Alice's bit $a$.
- ◯ None of the above



*This content is protected and may not be shared, uploaded, or distributed.*