CS161 Fall 2025 — Introduction to Computer Security — Midterm

Name: _____

Student ID: _____

This exam is 110 minutes long. There are 8 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	8	Total
Points:	0	15	18	13	13	12	19	10	100

For questions with **circular bubbles**, you may select only one choice.

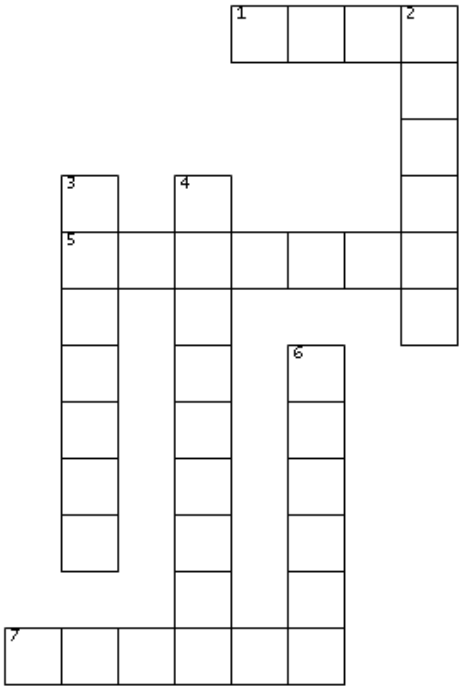
- ○ Unselected option (Completely unfilled)
- ◎ Don't do this (it will be graded as incorrect)
- ● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

- ■ You can select
- ■ multiple squares (completely filled).
- ☑ (Don't do this)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we may grade the worst interpretation.

Pre-Exam Activity: Crossword (0 points):



ACROSS:
1. What goes up
5. Cutest mascot
7. Professor _____

DOWN:
2. Don't forget to format
3. _____ in depth
4. Tastiest one-way function
6. Always overflowing

Q1 *Honor Code* 📜 (0 points)

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam.

Read the honor code above and sign your name: _____

Q2 *Potpourri* **(15 points)**

Q2.1 (1 point) True or False: The Trusted Computing Base (TCB) is another term for the data that we're trying to protect from an attack.

◯ True ◯ False

Q2.2 (1 point) A web app lets logged-in users access files by specifying a filename in the URL, but it does not restrict access, so any user can view all other users' files. Which security principle is violated?

◯ Shannon's Maxim

◯ Fail-safe defaults

◯ Detect if you can't prevent

◯ Ensure complete mediation

◯ None of the above

Q2.3 (1 point) True or False: When storing a word `0xDEADBEEF` in memory on a *big*-endian system, the bytes at increasing addresses are `\xEF`, `\xBE`, `\xAD`, `\xDE`.

◯ True ◯ False

Q2.4 (1 point) Which of the following registers stores the address of the top of the current stack frame?

◯ `eax` ◯ `eip`

◯ `ebp` ◯ None of the above

Q2.5 (1 point) True or False: `gets` is a memory-safe function because it will always automatically append a null terminator at the end.

◯ True ◯ False

Q2.6 (1 point) True or False: Buffer overflows cannot be exploited if stack canaries are in use, because then the return address cannot be overwritten.

◯ True ◯ False

Q2.7 (3 points) An attacker is exploiting a buffer overflow in a program with ASLR and non-executable pages enabled. Which of the following attacks is most feasible?

◯ Inject shellcode onto the stack and jump to it.

◯ Find some other vulnerability that reveals the address of a `libc` function, then overwrite the return address with the `libc` function address (return-to-`libc`).

◯ Overwrite the address of the saved frame pointer (SFP) on the stack.

◯ Overwrite the least significant byte of the saved frame pointer (SFP) on the stack with a null byte, so when the current function returns, the caller uses a fake stack frame and jumps to shellcode stored in the fake RIP (off-by-one).

Q2.8 (1 point) True or False: In hybrid encryption, the public key is used to encrypt the message, and the symmetric key is used only to sign.

◯ True ◯ False

Q2.9 (1 point) True or False: A major drawback of a Trusted Directory is that it is not scalable.

◯ True ◯ False

Q2.10 (4 points) Suppose Alice and Bob share a secret key K used with a secure MAC algorithm. Alice sends (M, T), where $T = \text{MAC}(K, M)$, and a man-in-the-middle attacker intercepts it.

The attacker wants to replace Alice's message with $(M \parallel 1, T')$, with T' chosen so that Bob will accept this modified message as valid. Is it feasible for an attacker to do this, without knowing the key K?

◯ Yes, because the attacker knows M and anyone can compute the hash of $M \parallel 1$

◯ Yes, because MACs provide confidentiality, not integrity

◯ Yes, because MACs are deterministic

◯ No, because MACs are unforgeable

◯ No, because MACs are collision resistant

◯ No, because MACs are deterministic

◯ None of the above

Q3 *Memory Safety: Betelgeuse* **(18 points)**

For each C code snippet below, answer the two subsequent questions regarding the vulnerability present and the best way to mitigate it.

```
1 void get_user_greeting() {
2     char user_name[32];
3     printf("Please enter your name:\n");
4     gets(user_name);
5     printf("Hello, %s!\n", user_name);
6 }
```

Q3.1 (2 points) Which of the following memory safety vulnerabilities are present in the code snippet above? Select all that apply.

- ☐ Stack Smashing / Buffer Overflow
- ☐ Format String Vulnerability
- ☐ Integer Conversion Vulnerability
- ☐ Time-of-Check/Time-of-Use
- ☐ Off-by-One Vulnerability
- ◯ None of the above

Q3.2 (4 points) Which of the following changes would effectively mitigate the vulnerabilities identified in the previous subpart? Select all that apply.

- ☐ Replace `gets(user_name);` with `fgets(user_name, 32, stdin);`.
- ☐ Insert `if (strlen(user_name) >= 32) return;` after line 4 and before line 5.
- ☐ Enable Stack Canaries.
- ☐ Replace `char user_name[32];` with `char username[16];`.
- ◯ None of the above

```
1 void log_status(char *status) {
2     char msg[24];
3     fread(msg, 1, 24, stdin);
4     printf(status);
5     printf(msg);
6 }
```

Q3.3 (2 points) Which of the following memory safety vulnerabilities are present in the code snippet above? Select all that apply.

- ☐ Stack Smashing / Buffer Overflow
- ☐ Format String Vulnerability
- ☐ Integer Conversion Vulnerability
- ☐ Off-by-One Vulnerability
- ☐ Time-of-Check/Time-of-Use
- ◯ None of the above

Q3.4 (4 points) Which of the following changes would effectively mitigate the vulnerabilities identified in the previous subpart? Select all that apply.

- ☐ Replace `fread(msg, 1, 24, stdin)` with `fgets(msg, 24, stdin)`.
- ☐ Replace `printf(status)` with `printf("%s", status)` and replace `printf(msg)` with `printf("%s", msg)`.
- ☐ Allocate `status` and `msg` on the heap instead of the stack.
- ☐ Enable Address Space Layout Randomization.
- ◯ None of the above

```
1 void process_message(char *msg, int len) {
2     char buffer[128];
3
4     if (len > 128) {
5         printf("Error: Message length exceeds buffer size.\n");
6         return;
7     }
8     memcpy(buffer, msg, len);
9 }
```

Q3.5 (2 points) Which of the following memory safety vulnerabilities are present in the code snippet above? Select all that apply.

- ☐ Stack Smashing / Buffer Overflow
- ☐ Heap Overflow
- ☐ Signed/Unsigned Integer Vulnerability
- ☐ Time-of-check/Time-of-Use
- ☐ Format String Vulnerability
- ◯ None of the above

Q3.6 (4 points) Which of the following changes would effectively mitigate the vulnerabilities identified in the previous subpart? Select all that apply.

- ☐ Use `strncpy(buffer, msg, len);` instead of `memcpy(buffer, msg, len);`
- ☐ Add a check to ensure `len` is not negative before the size comparison.
- ☐ Replace `int len` with `size_t len`
- ◯ None of the above

Q4 *Cryptography: One Question After Another* **(13 points)**

Q4.1 (3 points) Is a deterministic encryption scheme (where the same plaintext always produces the same ciphertext for a given key) considered IND-CPA secure?

- ◯ Yes, because IND-CPA security is guaranteed as long as the underlying block cipher is a one-way function.
- ◯ Yes, because the security relies on the secrecy of the key, not on randomizing the ciphertext output.
- ◯ No, because an attacker can detect if the same message is sent twice and thereby win the IND-CPA game.
- ◯ No, because determinism makes the scheme vulnerable to brute-force attacks on the key space.

Q4.2 (3 points) In a message encrypted with CBC mode, ciphertext block C_5 is corrupted, but C_6 and C_7 are received intact. Will the corresponding plaintext block P_7 be decrypted correctly?

- ◯ Yes, because the decryption of P_7 does not depend on the corrupted C_5.
- ◯ Yes, because P_7 is only dependent on C_7 and the original IV.
- ◯ No, because the "chaining" in CBC mode means an error in one block corrupts all subsequent plaintext blocks.
- ◯ No, because decrypting P_7 requires a correctly decrypted P_6, which is impossible since C_5 was corrupted.

Q4.3 (4 points) Alice attempts to provide message integrity by sending a message M concatenated with its hash, as $M \| H(M)$, where H is a secure cryptographic hash function. Does this scheme protect against an active adversary who can modify the message in transit?

- ◯ Yes, because the one-way property of the hash prevents an attacker from creating a new message M' that hashes to the same value as the original hash $H(M)$.
- ◯ Yes, because Bob's verification step, where he re-computes the hash of the received M, would detect any tampering.
- ◯ No, because this scheme fails to provide confidentiality, and integrity is not possible without it.
- ◯ No, because the hash is unkeyed, allowing an attacker to modify the message and simply recompute a valid hash for the new message.

Q4.4 (3 points) Alice and Bob use the Diffie–Hellman key exchange to derive a shared secret g^{ab}. After each session, they erase both their private exponents (a, b) and the derived value g^{ab} before starting a new one. Does this provide forward secrecy if an attacker later learns the values of g, a and b?

- ◯ Yes, because forward secrecy concerns the ability to predict future session keys after the compromise, not before.
- ◯ Yes, because a compromise of future secrets would not reveal past session keys, as they were derived from discarded ephemeral secrets.
- ◯ No, because the attacker can perform a man-in-the-middle attack during the initial exchange, which retroactively breaks forward secrecy.
- ◯ No, because if the public parameters g and p are ever revealed, all past sessions become insecure.

Q5 *Cryptography: Slack DMs* **(13 points)**

Alice wishes to send Bob a message m over an insecure channel, and is deliberating over what scheme $F(m)$ to employ. A secure scheme should provide:

- **Confidentiality.** From the value $F(m)$ alone, no adversary should be able to efficiently recover the value of message m. Assume that dictionary attacks on m are not possible (there are too many possible values of m for the attacker to enumerate all of them).
- **Integrity.** It should be computationally infeasible to find two distinct messages $m \neq m'$ such that $F(m) = F(m')$ (a collision).

For each scheme $F(m)$ below, determine whether it provides **confidentiality**, **integrity**, **both**, or **neither**. In all questions, $\|$ denotes concatenation, and H denotes a secure cryptographic hash function.

Q5.1 (4 points) Alice sends:

$$F(m) = H(m)$$

◯ Confidentiality only ◯ Integrity only ◯ Neither ◯ Both

Q5.2 (3 points) Let p be a publicly known, large prime number. Alice sends:

$$F(m) = 1^m \bmod p$$

◯ Confidentiality only ◯ Integrity only ◯ Neither ◯ Both

For the next two subparts, Alice wishes to send a two-part message composed of m_0 and m_1 where $m_0 \neq m_1$. For each scheme $F(m)$ below, determine whether it provides **integrity**.

Note:
- $\mathsf{len}(m_0 \parallel m_1)$ returns the length of $m_0 \parallel m_1$,
- $\oplus$ denotes the XOR function,
- $(m_0, m_1) = (m_0', m_1')$ is true if and only if both $m_0 = m_0'$ and $m_1 = m_1'$.

Q5.3 (3 points) Alice sends:

$$F(m_1) = H(m_0 \parallel m_1 \parallel \mathsf{len}(m_0 \parallel m_1))$$

Does this scheme provide integrity?
In other words: Is it computationally infeasible to find two distinct pairs $(m_0, m_1) \neq (m_0', m_1')$ such that $F(m_0 \parallel m_1 \parallel \mathsf{len}(m_0 \parallel m_1)) = F(m_0' \parallel m_1' \parallel \mathsf{len}(m_0' \parallel m_1'))$.

◯ Yes ◯ No

Q5.4 (3 points) Alice sends:

$$F(m) = H(m_0 \oplus m_1)$$

Does this scheme provide integrity?
In other words: Is it computationally infeasible to find two distinct pairs $(m_0, m_1) \neq (m_0', m_1')$ such that $F(m_0 \oplus m_1) = F(m_0' \oplus m_1')$.

◯ Yes ◯ No

Q6 *Cryptography: Is This Random?* **(12 points)**

A pseudorandom number generator (PRNG) uses an initial **seed** of truly random bits to generate a long sequence of outputs. The seed serves as the starting point, or **initial state**, denoted s_0. From any given state s_i, the PRNG produces an output r_{i+1} and computes the next internal state s_{i+1}.

A secure PRNG should have the following properties:

- **Deterministic** – Given the same seed, a PRNG must always produce the same sequence of outputs. This ensures reproducibility (anyone who knows the seed and algorithm can regenerate the exact sequence).
- **Pseudorandom** – The output sequence is *computationally indistinguishable* from true randomness for any efficient adversary who does not know the seed or internal state, i.e., an adversary cannot tell the difference between output from the PRNG vs truly random bits. In practice, this means past outputs should not help an attacker predict the next output.
- **Rollback Resistant** – Even if an attacker learns the *current* internal state, they cannot reconstruct *earlier* outputs or states.

For each of the three functions below, select **all** the properties that the function satisfies. If a function has none of these properties, select "None of the above." H denotes SHA256 (which is a secure cryptographic hash function).

Q6.1 (4 points) Select all of the characteristics that this PRNG satisfies:

$$r_{i+1} = H(s_i \parallel 0), \quad s_{i+1} = H(s_i \parallel 1)$$

☐ Deterministic ☐ Pseudorandom ☐ Rollback Resistant ◯ None of the above

Q6.2 (4 points) Select all of the characteristics that this PRNG satisfies:

$$r_{i+1} = H(i), \quad s_{i+1} = i + 1$$

☐ Deterministic ☐ Pseudorandom ☐ Rollback Resistant ◯ None of the above

Q6.3 (4 points) Select all of the characteristics that this PRNG satisfies:

$$r_{i+1} = H(s_i), \quad s_{i+1} = H(r_{i+1})$$

☐ Deterministic ☐ Pseudorandom ☐ Rollback Resistant ◯ None of the above

Q7 *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.

Q7.1 (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`

Q7.2 (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

Q7.3 (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.

Q7.4 (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

Q7.5 (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.

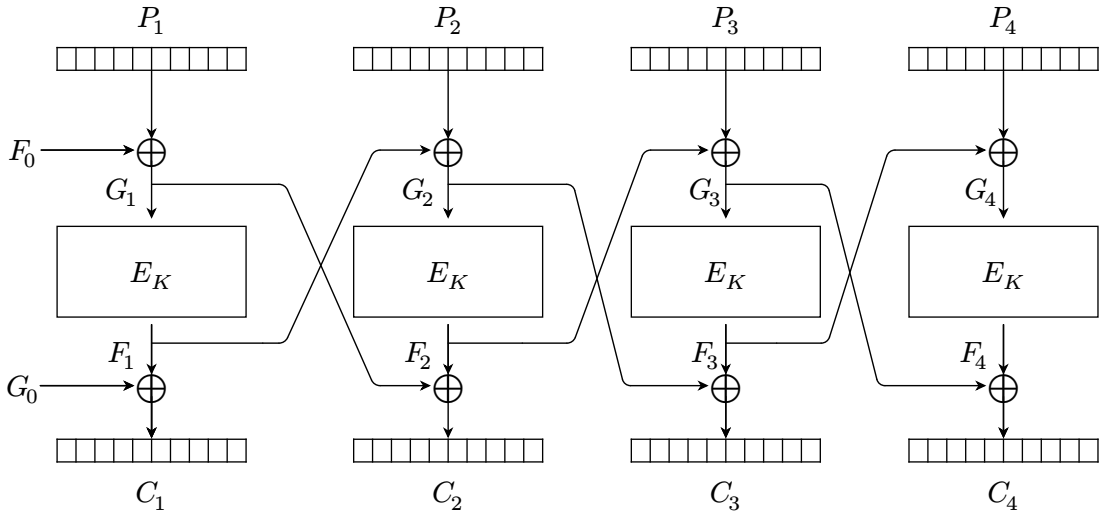
- ☐ `0xffffd640`
- ☐ `0xffffd630`
- ☐ `0xffffd620`
- ☐ `0xffffd610`
- ☐ `0xffffd600`
- ☐ `0xffffdbe0`
- ☐ `0xffffdbd0`
- ☐ `0xffffdbc0`
- ◯ None of the above

Q7.6 (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

Q8 *Cryptography: The Mumbling-Jumbling Block Cipher* **(10 points)**

EvanBot creates a new block cipher mode of operation. The encryption formula requires using two IVs: $G_0 = IV_0$ and $F_0 = IV_1$. Given a message $M = (P_1, P_2, P_3)$, the sender appends one more block $P_4 = 0$, which is used for integrity checking, then encryption proceeds as shown below:



$$G_i = P_i \oplus F_{i-1} \qquad F_i = E_K(G_i) \qquad C_i = F_i \oplus G_{i-1}$$

In this entire question, assume that all IVs are independently randomly generated.

Q8.1 (3 points) What formulas should we use to decrypt the ciphertexts?

- ◯ $F_i = C_{i-1} \oplus G_{i-1}$ $\quad G_i = D_K(F_i)$ $\quad P_i = G_i \oplus F_{i+1}$
- ◯ $F_i = C_{i+1} \oplus G_{i-1}$ $\quad G_i = D_K(F_{i+1})$ $\quad P_i = G_{i+1} \oplus F_i$
- ◯ $F_i = C_i \oplus G_{i-1}$ $\quad G_i = D_K(F_i)$ $\quad P_i = G_i \oplus F_{i-1}$

Q8.2 (3 points) Is this scheme IND-CPA-secure?

- ◯ Yes, no attacker can win the IND-CPA game with a probability greater than $\frac{1}{2}$.
- ◯ Yes, because no attacker can forge the ciphertexts without being detected.
- ◯ No, the last block of the plaintext, P_4, allows for attackers to distinguish between M_1 and M_2.
- ◯ No, because the IVs are non-deterministic.

When receiving a ciphertext, the recipient decrypts it (using the correct method from Q8.1), then verifies integrity by checking that $P_4 = 0$. If $P_4 \neq 0$, the recipient ignores the ciphertext (i.e. it is invalid). Otherwise, the recipient accepts the decrypted message.

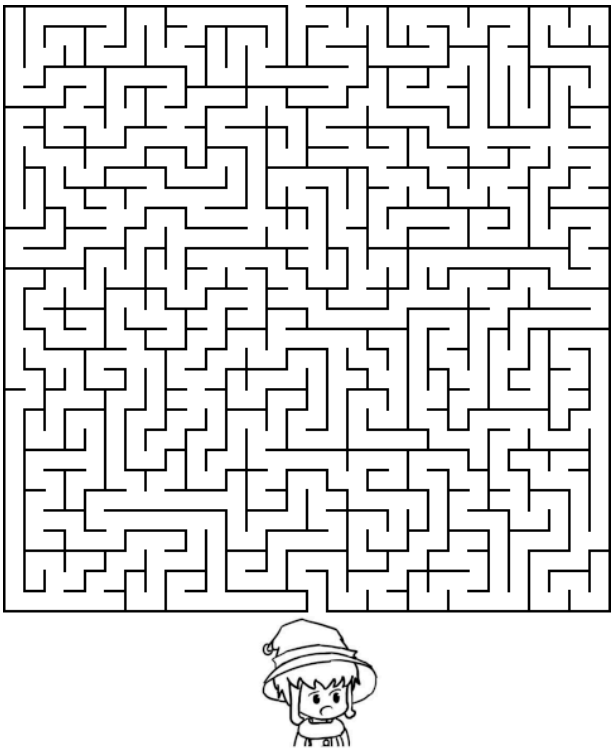
Q8.3 (4 points) Does this scheme provide authenticity?

- ◯ Yes, this is exactly Encrypt-then-MAC with CBC mode as the encryption scheme.
- ◯ Yes, an attacker who observes the encryption of either M_1 or M_2 cannot guess which was encrypted.
- ◯ No, because an attacker can always recover the secret key k from a ciphertext and reconstruct a new ciphertext that way.
- ◯ No, if a message has a 0 block, an attacker could truncate the ciphertext after that without being detected.
- ◯ No, if Alice encrypts $M = (P_1)$ with $P_1 = 0$ and the attacker observes the corresponding ciphertext $C = (C_1, C_2)$, then sending the ciphertext $C' = (C_1, C_2, C_1, C_2)$ will decrypt to the message $M' = (0, 0, 0)$ and the recipient will accept the decrypted message.

(Question 8 continued...)

Post-Exam Activity: The Bot and the Maze

Help Evanbot get through the maze!



Comment Box

Congratulations for making it to the end of the exam! Feel free to leave any final thoughts, comments, feedback, or doodles here: