CS 161 Fall 2025 — Introduction to Computer Security — Final

Name: ________________________________

Student ID: ________________________________

This exam is 170 minutes long. There are 11 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	8	9	10	11	Total
Points:	0	8	8	8	9	7	9	16	11	12	12	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: Caesar Cipher (0 points):

EvanBot wishes to share the following message, but doesn't want it to fall into the wrong hands. Decipher away!

Hint: Shift +3

J	→	
R	→	
R	→	
G	→	
O	→	
X	→	
F	→	
N	→	

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* **(8 points)**

Q2.1 (0.5 points) True or False: The principle of "Least Privilege" dictates that a program or user should only be granted the specific permissions required to perform their intended task and nothing more.

◯ True ◯ False

Q2.2 (0.5 points) True or False: Using GDB, if you want to inspect the **contents** of the buffer `buf` in the code to the right, you can break at line 7 and then run the command `x/16x buf`.

◯ True ◯ False

C code:

```
1 void func(){
2   char buf[16];
3   ...
4 }
5
6 void vulnerable(){
7   ...
8   func();
9 }
```

You run the code in GDB, break at line 7, and run `info frame`. You receive the output to the right.

GDB Output:

```
(gdb) info frame
...
Saved registers:
ebp at 0xffffdc58,
eip at 0xffffdc5c
```

Q2.3 (0.5 points) True or False: The RIP of `vulnerable()` is at the address `0xffffdc5c`.

◯ True ◯ False

Q2.4 (0.5 points) True or False: The value of the SFP of `func()` is `0xffffdc58`.

◯ True ◯ False

Q2.5 (0.5 points) True or False: Format string vulnerabilities allow an attacker to read data from the stack, but they cannot be exploited to write to memory or execute shellcode.

◯ True ◯ False

Q2.6 (0.5 points) True or False: Bear Systems modifies ASLR: instead of randomly generating the memory offset when the program starts, they randomly generate the memory offset when the program is compiled and hardcode this offset into the binary. Compared to standard randomized ASLR, Bear Systems' modification improves security against memory safety exploits.

◯ True ◯ False

Q2.7 (0.5 points) True or False: The HMAC algorithm is specifically designed to be secure against length extension attacks (where an attacker can compute $H(M \parallel M')$ for some M' given only $H(M)$, the length of M, and M').

◯ True ◯ False

Q2.8 (0.5 points) True or False: In AES-CBC mode, encryption cannot be parallelized because the encryption of block C_i depends on the ciphertext of the previous block C_{i-1}, but decryption *can* be parallelized.

◯ True ◯ False

Q2.9 (0.5 points) True or False: If an attacker intercepts a Diffie-Hellman key exchange where Alice sends $A = g^a \bmod p$ and Bob sends $B = g^b \bmod p$, the attacker can easily compute the shared secret S if they can solve the Discrete Logarithm Problem.

◯ True ◯ False

Q2.10 (0.5 points) True or False: El Gamal encryption is deterministic; if you encrypt the same message M twice with the same public key, you will always generate the exact same ciphertext (C_1, C_2).

◯ True ◯ False

Q2.11 (0.5 points) True or False: The "Same-Origin Policy" allows a script loaded by **`http://example.com`** to read the properties of a document from **`https://example.com`** because they share the same domain name and the protocol doesn't need to match.

◯ True ◯ False

Q2.12 (0.5 points) True or False: If a website sets a cookie with **`Domain=.example.com`**, that cookie will be sent by the browser in requests to both **`www.example.com`** and **`secure.example.com`**.

◯ True ◯ False

Q2.13 (0.5 points) True or False: In a TCP handshake, if the Initial Sequence Number (ISN) is generated randomly with a cryptographically secure pseudorandom number generator, this will prevent off-path attackers from easily injecting packets into the connection by guessing the sequence number.

◯ True ◯ False

Q2.14 (0.5 points) True or False: DNSSEC with NSEC prevents "zone walking" (enumerating all valid domain names in a zone) by using NSEC records that return a cryptographic hash of the queried domain name rather than the name itself.

◯ True ◯ False

Q2.15 (0.5 points) True or False: The BGP (Border Gateway Protocol) includes built-in cryptographic verification to ensure that an Autonomous System (AS) actually owns the IP prefixes it advertises, preventing prefix hijacking attacks by default.

◯ True ◯ False

Q2.16 (0.5 points) True or False: The Kaminsky DNS attack allows an attacker to poison the DNS cache of a recursive resolver by flooding it with spoofed responses for non-existent subdomains (e.g., **`1.google.com`**, **`2.google.com`**), aiming to overwrite the authority records for the target domain (e.g., **`google.com`**).

◯ True ◯ False

Q3 *Memory Safety: Tomayto* 🍅 **(8 points)**

Consider the following vulnerable C code:

```
 1  void tomayto(int count, char *input) {
 2      char buffer[32];
 3      if (count > 32) { return; }
 4      memcpy(buffer, input, count);
 5  }
 6
 7  void main() {
 8      int user_int = 0;
 9      char user_string[40];
10      fread(user_string, 4, 11, stdin);
11      tomayto(user_int, user_str);
12  }
```

Stack at Line 5

RIP of `main`
SFP of `main`
(1)
(2)
(3)
(4)
RIP of `tomayto`
SFP of `tomayto`
`buffer`

- All memory safety defenses are disabled.
- You run GDB, set a breakpoint at line 4, run and find that `buffer` starts at address `0x50ffd100`.
- You run GDB, set a breakpoint at line 4, run and find that the RIP of `tomayto` has the value `0x08048999`.
- Your goal is to execute the 32-byte long `SHELLCODE`.

(0.5 points each) What goes in the blanks in the stack diagram above?

Q3.1 Blank (1): (A) `count` (B) `user_string` (C) `user_int` (D) `input`

Q3.2 Blank (2): (A) `count` (B) `user_string` (C) `user_int` (D) `input`

Q3.3 Blank (3): (A) `count` (B) `user_string` (C) `user_int` (D) `input`

Q3.4 Blank (4): (A) `count` (B) `user_string` (C) `user_int` (D) `input`

Q3.5 (1 point) Which of the following memory safety vulnerabilities are present in the above code?

[A] Format String Vulnerability [C] Stack Buffer Overflow [E] Heap Overflow

[B] Signed/Unsigned [D] ret2ret (F) None of the above

Q3.6 (4 points) Provide an input to `fread` on Line 10 that would cause the program to execute shellcode. If a part of the input can be any non-zero value, use `'A' * n` to represent `n` bytes of garbage.

Don't worry about segfaults that could possibly occur during the `memcpy` (all memory is mapped in). If you weren't worried about that, please ignore this remark.

Q3.7 (1 point) Which changes need to be made to make this code memory-safe?

[A] Line 1: Change `int count` to `size_t count;`

[B] Line 3: Change `count > 32` to `count >= 32`

[C] Line 4: Add `input[31] = '\0';` before the `memcpy` on line 5

[D] Line 8: Change `int user_int = 0;` to `int user_int = -1;`

Q4 *Memory Safety: I've played these games before...* 🍅 **(8 points)**

Consider the following vulnerable C code:

```
1  void tomahto() {
2      char cage[12];
3      fgets(cage, 12, stdin);
4      printf(cage);
5      gets(cage);
6  }
7
8  void main() {
9      tomahto();
10 }
```

Stack at Line 6
RIP of `main`
SFP of `main`
(1)
RIP of `tomahto`
(2)
(3)
(4)

- Stack canaries are enabled. All other memory safety mitigations are disabled.
- You run GDB, set a breakpoint at line 5, run and find that `cage` starts at address `0xffffd100` and that there is a copy of `SHELLCODE` at address `0xffffd204`.
- Through trial and error, you discover that the stack canary for `tomahto` is the 4th value `printf` reads from the stack when it looks for arguments (i.e., it is offset 4 words away from the stack pointer `printf` uses).

(0.25 points each) What goes in the blanks in the stack diagram above?

Q4.1 Blank (1): Ⓐ `cage` Ⓑ SFP of `tomahto` Ⓒ canary of `tomahto` Ⓓ canary of `main`

Q4.2 Blank (2): Ⓐ `cage` Ⓑ SFP of `tomahto` Ⓒ canary of `tomahto` Ⓓ canary of `main`

Q4.3 Blank (3): Ⓐ `cage` Ⓑ SFP of `tomahto` Ⓒ canary of `tomahto` Ⓓ canary of `main`

Q4.4 Blank (4): Ⓐ `cage` Ⓑ SFP of `tomahto` Ⓒ canary of `tomahto` Ⓓ canary of `main`

Q4.5 (1 point) Which of the following memory safety vulnerabilities are present in the above code?

[A] Format String Vulnerability [C] Stack Buffer Overflow [E] Heap Overflow

[B] Signed/Unsigned [D] ret2ret Ⓕ None of the above

Q4.6 (2 points) Which of these inputs to `fgets` on Line 3 will always leak the value of the stack canary in the `tomahto` stack frame? Select all that apply.

Note: Stack canaries are four random bytes and do not contain a null byte.

[A] `'%x' * 4` [C] `('%c' * 3) + '%x'` [E] `'%n' * 4`

[B] `'%x' * 3` [D] `'%x' + ('%s' * 3)` Ⓕ None of the above

In the next part, provide an input to `gets` on line 5 that would cause the program to execute `SHELLCODE`, assuming the correct input has been provided to `fgets` on line 3. You may use `CANARY` to refer to the correct 4-byte string value of the stack canary, as leaked by `printf`.

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

Q4.7 (4 points) Input to `gets` on line 5:

Q5 *Cryptography: Secret Santa* 🎅 **(9 points)**

Annabella and Fred want to establish a secure communication channel. They require a protocol that supports asynchronous communication (Annabella can send a message even if Fred is offline, i.e., even if Fred is not connected to the Internet at that moment), mutual authentication, and forward secrecy.

Q5.1 (2 points) Why is a basic, unauthenticated Diffie-Hellman exchange vulnerable to Man-in-the-Middle (MITM) attacks?

Ⓐ The discrete logarithm problem is easier to solve when values are intercepted.

Ⓑ Diffie-Hellman keys are too short to resist brute-force attacks.

Ⓒ The public keys exchanged are not cryptographically bound to the users' identities.

Ⓓ Servers cannot store Diffie-Hellman public values.

Q5.2 (1 point) Annabella and Fred decide to simply publish static (long-term) Diffie-Hellman (DH) public keys to a single, central, trusted server. They use these same keys to derive a shared secret for every message they ever send.

Why does this approach fail to provide forward secrecy?

Ⓐ Static DH outputs are deterministic and therefore predictable by random guessing.

Ⓑ The server must regenerate the group parameters for each session.

Ⓒ If a long-term private key is stolen later, the attacker can decrypt all past recorded traffic.

Ⓓ Public keys expire too quickly to be useful.

The Protocol

To solve these issues, Annabella and Fred adopt a new scheme:

1. Fred uploads keys: Fred generates the following and uploads the **public** parts to a central trusted server:

- Identity Key (IK_F): Long-term static key pair.
- Signed Pre-Key (SPK_F): A medium-term key pair signed by IK_F.
- One-Time Pre-Keys ($\mathrm{OPK}_F[i]$): A batch of key pairs intended to be used once and deleted. Not signed.

2. Annabella fetches keys: If Annabella wants to message Fred, she fetches Fred's IK_F, SPK_F, and a single $\mathrm{OPK}_F[i]$ from the server; verifies the signature, and generates a fresh Ephemeral Key pair (EK_A).

3. Key derivation: Annabella computes the shared secret SK by combining four Diffie-Hellman (DH) calculations:

1. $K_1 = \mathrm{DH}(\mathrm{IK}_A, \mathrm{SPK}_F)$
   (Binds Annabella's Identity to Fred's Signed Key)
2. $K_2 = \mathrm{DH}(\mathrm{EK}_A, \mathrm{IK}_F)$
   (Binds Session to Fred's Identity)
3. $K_3 = \mathrm{DH}(\mathrm{EK}_A, \mathrm{SPK}_F)$
   (Binds Session to Fred's Signed Key)
4. $K_4 = \mathrm{DH}(\mathrm{EK}_A, \mathrm{OPK}_F[i])$
   (Provides Strong Forward Secrecy)

$$\mathrm{SK} = \mathsf{H}(K_1 \parallel K_2 \parallel K_3 \parallel K_4)$$

Q5.3 (2 points) The calculation includes the term $\mathrm{DH}(\mathrm{IK}_A, \mathrm{SPK}_F)$. Why does this term specifically provide assurance to Annabella that the recipient is actually Fred?

Ⓐ Because Annabella verified the signature on SPK_F, she knows only the holder of Fred's private Identity Key could have authorized it.

Ⓑ Because Annabella's Identity Key (IK_A) is included, Fred automatically knows who sent the message.

Ⓒ Because DH values are universally unique, no one else could generate this specific integer.

Ⓓ Because the server performs a Zero-Knowledge Proof to validate the Pre-Key before storage.

Q5.4 (1 point) Which architectural feature specifically enables asynchronous communication (Annabella sending a message while Fred is offline)?

Ⓐ The use of symmetric Key Derivation Functions (KDF).

Ⓑ The requirement for Annabella to sign her own messages.

Ⓒ The inclusion of One-Time Pre-Keys for forward secrecy.

Ⓓ The use of a server to store Fred's pre-published public keys.

Q5.5 (1 point) Why couldn't Annabella and Fred just use a standard, ephemeral Diffie-Hellman handshake to achieve asynchronous messaging?

Ⓐ Fred's computer cannot generate DH keys when it is not connected to the Internet.

Ⓑ Ephemeral DH requires both parties to be online simultaneously to exchange values.

Ⓒ Servers are technically incapable of storing DH integers.

Ⓓ Standard DH is too computationally expensive for mobile devices.

Q5.6 (1 point) Consider a simplified protocol that **only** computes $\mathrm{SK} = \mathrm{DH}(\mathrm{EK}_A, \mathrm{IK}_F)$.

Which security properties are **missing** from this specific exchange? Select all that apply.

☐ A Confidentiality against passive eavesdroppers.

☐ B Authentication of Annabella (to Fred).

☐ C Authentication of Fred (to Annabella).

☐ D Forward Secrecy.

Ⓔ None of the above

Q5.7 (1 point) Fred modifies the scheme to publish only an identity key IK_F and a batch of one-time pre-keys OPK_F, but not SPK_F (no signed pre-key is published). The secret key is computed as $\mathrm{SK} = \mathsf{H}(K_2 \parallel K_4)$.

Select all the security guarantees/properties this modified scheme provides.

☐ A Authentication of Annabella (Fred can verify he is speaking with Annabella)

☐ B Authentication of Fred (Annabella can verify she is speaking with Fred)

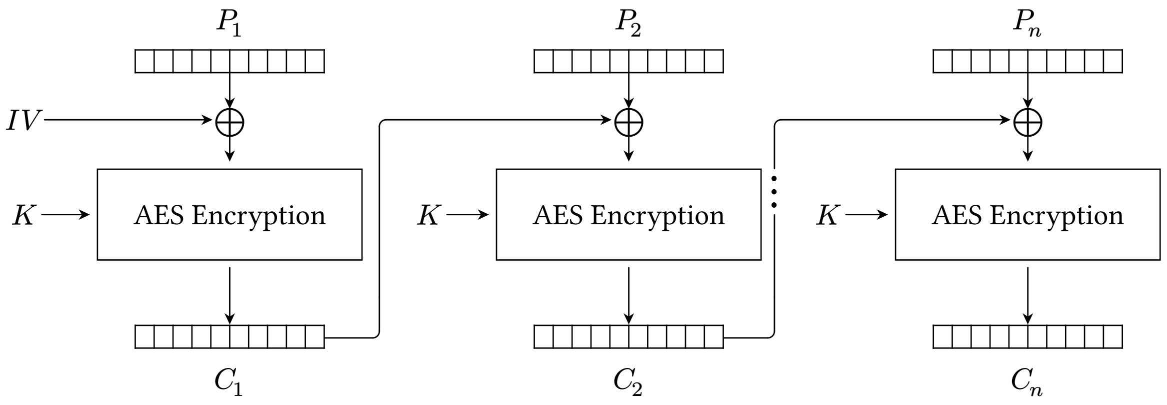
☐ C Forward secrecy (as long as unused one-time pre-keys remain)

☐ D Asynchronous communication (only while unused one-time pre-keys remain)

Ⓔ None of the above

Q6 *Cryptography: Double Dipping* 🍯 **(7 points)**

Recall the implementation of AES-CBC encryption:



- Alice uses AES-CBC encryption to encrypt the plaintext $P = P_1 \parallel P_2 \parallel P_3$. She sends the corresponding ciphertext $C = C_1 \parallel C_2 \parallel C_3$ to Bob.
- Alice and Bob use a key K_1.

Q6.1 (2 points) Under AES-CBC, which of the following are the correct value for C_3? Select all that apply.

- [A] $C_3 = E_{K_1}(C_2)$
- [B] $C_3 = E_{K_1}(C_2 \oplus P_3)$
- [C] $C_3 = E_{K_1}(P_3 \oplus E_{K_1}\left(P_2 \oplus E_{K_1}(P_1 \oplus IV)\right)$
- [D] $C_3 = E_{K_1}(P_1 \oplus P_2 \oplus P_3 \oplus IV)$
- [E] $C_3 = E_{K_1}(P_1 \oplus P_2 \oplus P_3)$
- [F] $C_3 = E_{K_1}(C_2 \oplus P_3) \oplus IV$

Q6.2 (1.5 points) Mallory wants to manipulate the message. She flips the **first bit** of the IV. She leaves all ciphertext blocks (C_1, C_2, C_3) unchanged.

What happens to the decrypted plaintext P'?

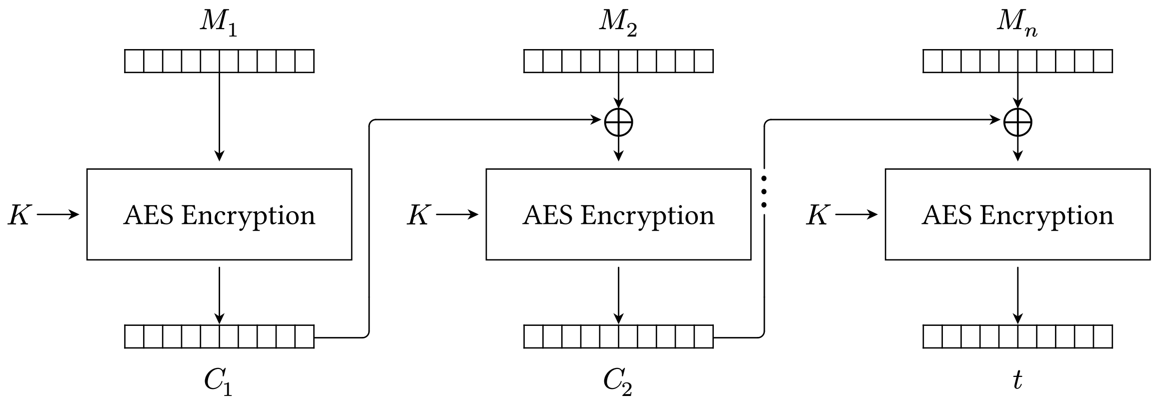
- (A) The first bit of P_1' is flipped; all other blocks are correct.
- (B) The whole block P_1' is garbled (randomized); all other blocks are correct.
- (C) The first bit of P_1' is flipped, and P_2' is completely garbled.
- (D) The decryption fails completely due to padding errors.

Q6.3 (1.5 points) Alternatively, suppose Mallory flips the **last bit** of ciphertext block C_1. She leaves IV, C_2, C_3 unchanged.

What is the specific effect on the decrypted plaintext blocks P_1' and P_2'?

- (A) Both P_1' and P_2' have their last bits flipped.
- (B) P_1' has its last bit flipped; P_2' is completely garbled.
- (C) P_1' is completely garbled; P_2' has its last bit flipped.
- (D) P_1' is completely garbled; P_2' is unaffected.

Consider the CBC-MAC scheme, which takes an input message $M = (M_1, M_2, ..., M_n)$ and key K, and outputs a tag t. The same key is used for all CBC-MAC computations in this question.



Suppose for the next two subparts:

- Recall that Alice encrypted P with K_1 and sent it to Bob. Bob will also decrypt with K_1.
- Assume that Alice used $IV = 0$ when encrypting P.
- Before Bob can decrypt, Mallory (an on-path attacker) tampers with the first 2 ciphertext blocks so that Bob receives $C' = C_1' \parallel C_2' \parallel C_3$. Note that C_3 and IV remain unchanged.
- Alice also computes a CBC-MAC tag t on the plaintext P, using key K_2, and sends it to Bob.
- Bob decides to compute the CBC-MAC tag t' on his tampered P' (the decryption of the tampered C') to verify the integrity of the plaintext.
- Alice and Bob use K_2 as the key for CBC-MAC, and K_1 for AES-CBC encryption.

Q6.4 (1 point) Bob decrypts C' to get P'. Which blocks of P' will be guaranteed to be the same as the corresponding blocks from P? Select all that apply.

- A P_1'
- B P_2'
- C P_3'
- D None of the above

Q6.5 (1 point) Is it possible for $t' = t$ (i.e., the MAC remains valid despite the tampering)?

- A Yes, if $K_1 = K_2$.
- B No, because C_2' has been tampered with.
- C No, because the plaintext P' is different from P.
- D No, because the attacker does not know the key K_2.

Q7 *Networking: The Fate of Streamify* 🎶 **(9 points)**

Nazar is live-streaming video to Zoir over Streamify, their new streaming service. To send the video, Nazar's computer sends UDP packets to Zoir's computer.

Suppose there are no security provisions: Zoir's computer accepts the UDP packet if and only if it has the correct UDP destination port. In UDP, port numbers are 16 bits.

Q7.1 (2 points) What is the probability that a single spoofed UDP packet from an off-path attacker is accepted, if it uses the correct destination IP address and if the destination port is chosen randomly?

Ⓐ 1 Ⓑ $1/2^8$ Ⓒ $1/2^{16}$ Ⓓ $1/2^{24}$ Ⓔ $1/2^{32}$ Ⓕ 0

Q7.2 (2 points) Now suppose an **on-path** attacker observes a few packets sent by Nazar, and then wants to send a new spoofed packet to Zoir. What is the best probability that the on-path attacker can have their spoofed UDP packet be accepted by Zoir?

Ⓐ 1 Ⓑ $1/2^8$ Ⓒ $1/2^{16}$ Ⓓ $1/2^{24}$ Ⓔ $1/2^{32}$ Ⓕ 0

Nazar and Zoir want stronger security, so they add a 16-bit integrity tag to each packet:
$t = \mathsf{SHA256\text{-}HMAC}(K,\ \texttt{packet})$, where `packet` is the rest of the packet, and K is an 8-bit secret key shared securely between Nazar and Zoir. A packet is accepted if and only if has both a correct 16-bit destination port and correct 16-bit integrity tag t.

Q7.3 (2 points) An **off-path** attacker wants to send a spoofed packet to Zoir. What is the best probability that the off-path attacker can have their spoofed UDP packet be accepted by Zoir, in this new design? Assume the attacker doesn't know the UDP port and has to guess it randomly.

Ⓐ 1 Ⓑ $1/2^8$ Ⓒ $1/2^{16}$ Ⓓ $1/2^{24}$ Ⓔ $1/2^{32}$ Ⓕ 0

Q7.4 (3 points) Now suppose an **on-path** attacker observes a few packets sent by Nazar, and then wants to send a new spoofed packet to Zoir. What is the best probability that the on-path attacker can have their spoofed UDP packet be accepted by Zoir, in this new design? Assume the attacker can do any reasonable amount of computation before constructing the spoofed packet.

Ⓐ 1 Ⓑ $1/2^8$ Ⓒ $1/2^{16}$ Ⓓ $1/2^{24}$ Ⓔ $1/2^{32}$ Ⓕ 0

Q8 *Web Security: Super Query League* ⚽ **(16 points)**

EvanBot has created a fantasy football app called FantasyLeague. Each user is able to name their team, draft players, and manage lineups. The app relies on the following SQL tables:

Account	
`username`	`string`
`password`	`string`
`team_name`	`string`

Team	
`team_name`	`string`
`draft_order`	`uint32_t`
`completed`	`boolean`

Player	
`team_name`	`string`
`player_name`	`string`
`in_lineup`	`boolean`

When users create an account, they select a `team_name` that is used to represent their team. In each round of drafting, the `draft_order` determines the order in which teams get to select players. Once a team has run out of people to draft, `completed` is set to `true` and they can start playing. Each week, they set the `in_lineup` variable for some of the players on their team, and score based on the people who are in their lineup.

Q8.1 (3 points) To search for players, the app executes the following query:

```
SELECT player_name, team_name FROM Player WHERE player_name = '$search';
```

Which of the following payloads, when injected into the `$search` parameter, will allow the attacker to learn the `username` and `password` of every user in the `Account` table, if the attacker can see the results of the query?

Select all that apply.

[A] `' OR 1=1; --`

[B] `' UNION SELECT username, password FROM Account --`

[C] `' UNION SELECT username, password, team_name FROM Account --`

[D] `' UNION SELECT password, username FROM Account --`

[E] `' UNION SELECT * FROM Account --`

[F] `' UNION SELECT player_name, team_name FROM Player --`

Q8.2 (3 points) Jonah's jealousy knows no bounds. He decides to ban his rival, `Fred`, from the platform entirely. To login, the app executes the following query:

```
SELECT * FROM Account WHERE username = '$username';
```

When injected into the `$username` parameter of the login query, which of the following payloads will delete the row `Fred` from the `Account` table, and only delete that row?

Select all that apply.

Note: `DELETE FROM table WHERE condition` removes rows from `table` that satisfy `condition`.

- [A] `'; DELETE FROM Account WHERE username = 'Fred';`
- [B] `'; DELETE FROM Account WHERE username = 'Fred'; --`
- [C] `'; DELETE FROM Account WHERE username = 'Fred' AND '1' = '1`
- [D] `'; DELETE FROM Account WHERE username = 'Fred' AND '1' = '1';`
- [E] `'; DELETE FROM Account WHERE username = 'Fred' AND '1' = '1'; --`
- [F] `'; DELETE FROM Player WHERE player_name = 'Fred';`
- [G] `'; DELETE FROM Player WHERE player_name = 'Fred'; --`

Q8.3 (2 points) The login page has a "Check Username" feature to see if a user exists. It is vulnerable to SQL injection but does not display any database content. It simply prints "User Found" if the query returns any rows, and "User Not Found" otherwise.

```
SELECT * FROM Account WHERE username = '$user';
```

You suspect the `admin` user has a password starting with the letter 's'. Which payload(s) will let you infer whether your suspicion is correct, based on whether the program prints "User Found" or not in response to your payload?

Select all that apply.

Note: `substring(str, start, length)` extracts `length` characters from `str` beginning at index `start`.

- [A] `admin' --`
- [B] `admin' OR '1'='1' --`
- [C] `admin' AND substring(password, 1, 1) = 's' --`
- [D] `admin' AND '0'='1' UNION SELECT * FROM Account WHERE substring(password, 1, 1) = 's' --`

Q8.4 (3 points) You can ask the app to release a player from their team. The app will execute the following SQL command.

```
DELETE FROM Player
 WHERE player_name = '$player'
   AND team_name = '$my_team';
```

You are logged in as `guest` and your team name is `GuestTeam`. Assume `$my_team` is set to `GuestTeam` and cannot be modified. You control `$player` but not `$my_team`.

Provide a SQL payload for the `$player` input that will delete every player belonging to the rival team `Winners`, without deleting players from other teams.

Q8.5 (3 points) On signup, the application executes the following SQL query as a prepared statement. The user-provided inputs `user` and `team` are bound securely to the `?` placeholders:

```
  INSERT INTO Account (username, password, team_name)
  VALUES (?, 'dummy_pass', ?);
```

Later, when a user views their team page, the app executes the following SQL query (notice how there is no prepared statement here):

```
SELECT * FROM Team WHERE team_name = '$my_team';
```

Assume `$my_team` is retrieved securely from the `Account` table and matches the `team` provided at signup.

Provide a stored-injection payload for the `team` parameter during signup that will drop the entire `Team` table when the user visits their team page later.

Q8.6 (2 points) In 10 words or fewer, describe one other serious security flaw with the signup design in the prior part. Do **not** mention SQL injection; find some other flaw.

Q9 *Network Security: Caffè Strada* ☕ **(11 points)**

Consider a local area network (LAN) at a coffee shop.

- Alice is a legitimate user trying to connect to the internet and communicate with Bob.
- Mallory is a malicious attacker on the same LAN.
- The network uses standard ARP for address resolution and DHCP for configuration.

Q9.1 (1 point) Mallory is on the same local network as Alice. She wants to intercept Alice's traffic destined for the Internet by providing false information about the Gateway Router.

If Mallory uses **ARP spoofing**, what information must she substitute in her malicious replies?

Ⓐ Substitute the IP address normally provided by the gateway with Mallory's IP address

Ⓑ Substitute the MAC address normally provided by the gateway with Mallory's MAC address

Q9.2 (1 point) Mallory is on the same local network as Alice. She wants to intercept Alice's traffic destined for the Internet by providing false information about the Gateway Router.

If Mallory uses **DHCP spoofing**, what information must she substitute in her malicious replies?

Ⓐ Substitute the IP address normally provided by the gateway with Mallory's IP address

Ⓑ Substitute the MAC address normally provided by the gateway with Mallory's MAC address

Q9.3 (2 points) Alice restarts her computer and rejoins the network. She broadcasts a `Client Discover` message to get a network configuration via DHCP.

Mallory sends a malicious `DHCP Offer` to Alice. Which fields in this offer would Mallory alter to position herself as a Man-in-the-Middle? Select all that apply.

☐ A Gateway Router IP

☐ B Alice's IP Address

☐ C Alice's public key

☐ D WiFi WPA2 pre-shared key

Ⓔ None of the above

Q9.4 (1 point) The coffee shop owner wants to improve security. Which of the following defenses would effectively mitigate these vulnerabilities? Select all that apply.

☐ A Implementing WPA2-PSK will prevent Mallory from spoofing ARP packets even if she knows the WiFi password.

☐ B DHCP attacks are trivial to fix by having the router sign all DHCP offers with a hardcoded certificate.

Ⓒ None of the above

(Question 9 continued...)

EvanBot and CodaBot are working on a project at a coffee shop. Mallory is connected to the same local network (LAN) and wants to intercept their traffic.

Mallory is an on-path attacker who can guarantee that her packets arrive before any legitimate packets, 100% of the time.

Q9.5 (2 points) EvanBot connects to the network for the first time and broadcasts a DHCP `Client Discover` message.

To trick EvanBot into accepting her configuration, how many packets is Mallory required to send that contain a falsified Source IP or Source MAC address (i.e., pretending to be the legitimate server in Source IP address or Source MAC address of the packet header)?

Ⓐ 0 Ⓑ 1 Ⓒ 2 Ⓓ More than 2

Q9.6 (2 points) CodaBot is already connected to the network. He wants to send a packet to the real Gateway Router (IP `10.1.6.1`), but his ARP cache is empty.

CodaBot broadcasts: "Who has IP `10.1.6.1`?".

How many packets does Mallory need to send to poison CodaBot's ARP cache and force him to send his traffic to her instead of the real router?

Ⓐ 0 Ⓑ 1 Ⓒ 2 Ⓓ More than 2

Q9.7 (2 points) What kind of attackers can execute a DHCP spoofing attack? Select all that apply.

[A] Man-in-the-middle

[B] On-path

[C] Off-path

Ⓓ None of the above

Q10 *Web Security: Criss Cross Apple Sauce* 🍏 (12 points)

Q10.1 (2 points) Alice visits a search engine that displays her search terms back to her on the results page. For example, if she searches for "cats", the page displays: `You searched for: cats`.

Mallory notices that the website does not sanitize this output. She creates a link containing a malicious JavaScript payload in the search query and tricks Alice into clicking it. When Alice clicks the link, the script executes in her browser.

What specific type of attack is this?

Ⓐ Stored XSS

Ⓑ Reflected XSS

Ⓒ Cross-Site Request Forgery (CSRF)

Ⓓ SQL Injection

Q10.2 (1 point) Mallory creates a malicious website called `freemoney.com`. On this page, she places a "Claim Prize" button.

Directly on top of this button, she layers a transparent (invisible) `<iframe>` that loads Alice's bank account settings page, specifically positioning the "Delete Ac=count" button directly over her "Claim Prize" button.

Alice visits `freemoney.com` and clicks "Claim Prize," but she unknowingly clicks the "Delete Account" button on the invisible bank page.

What specific type of attack is this?

Ⓐ Phishing

Ⓑ Cross-Site Request Forgery (CSRF)

Ⓒ Clickjacking (UI Redressing)

Ⓓ Man-in-the-Middle (MITM)

Q10.3 (2 points) A web developer tries to prevent XSS by writing a filter that simply removes the text `<script>` and `</script>` from all user input.

Which of the following payloads would successfully bypass this specific filter and execute JavaScript?

Ⓐ `<script>alert(1)</script>`

Ⓑ `<img src=x onerror="alert(1)">`

Ⓒ `<bold>alert(1)</bold>`

Ⓓ `<a href="http://evil.com">Click me</a>`

Q10.4 (2 points) Mallory creates a malicious website containing a hidden HTML form. When Alice visits Mallory's site, a script automatically submits this form to `bank.com` to transfer money.

Why does `bank.com` accept this request and transfer the money, even though Alice did not intend to click the button?

Ⓐ Mallory guessed Alice's password and included it in the form.

Ⓑ Alice's browser automatically attached her `bank.com` session cookies to the request.

Ⓒ Mallory intercepted Alice's Wi-Fi traffic and modified her packets.

Ⓓ The bank's server allows any request that comes from the same IP address as Mallory.

Q10.5 (1 point) A server defends against CSRF by checking the `Referer` header. It rejects requests where the Referer is `evil.com`. However, to protect user privacy, the server accepts requests where the `Referer` header is missing (blank).

How can Mallory bypass this defense?

Ⓐ By encrypting the Referer header so the server cannot read it.

Ⓑ By spoofing the IP address to look like it came from the server.

Ⓒ By configuring her website (e.g., using `<meta name="referrer" content="no-referrer">`) to tell the browser not to send a Referer header.

Ⓓ It is impossible to bypass; browsers always send the Referer header.

Q10.6 (2 points) Alice wants to implement a strong defense against XSS that restricts where scripts can be loaded from. She configures her server to send a specific HTTP header that tells the browser: "Only load scripts from `https://mysite.com`. Block all other scripts."

What is the name of this defense?

Ⓐ Same-Origin Policy (SOP)

Ⓑ Content Security Policy (CSP)

Ⓒ Cross-Site Request Forgery (CSRF) Token

Ⓓ Referer Validation

Q10.7 (2 points) The most robust defense against CSRF is the use of a CSRF Token (a random, secret value included in forms).

Why does this prevent Mallory from successfully forging a request from `evil.com`?

Ⓐ The token encrypts the session cookie so Mallory cannot use it.

Ⓑ Mallory cannot read the token from the legitimate site due to the Same-Origin Policy, so she cannot include the correct token in her forged request.

Ⓒ The token verifies that the IP address of the request matches the IP address of the user.

Ⓓ The token prevents the browser from sending cookies to the server.

Q11 *Memory Safety: Jonah Dreams of Sheep* 🐑 **(12 points)**

Consider the following vulnerable C code:

```
 1  void sleep() {
 2    char dream[24];
 3    fread(dream, 24, 1, stdin);
 4    count_sheep(dream);
 5  }
 6
 7  void count_sheep(char* input) {
 8    size_t num_sheep;
 9    fread(&num_sheep, sizeof(size_t), 1, stdin);
10
11    if (num_sheep > 64) {
12      exit(1);
13    }
14    printf(input);
15  }
```

Stack at Line 9
RIP of `sleep`
(1)
(2)
(3)
RIP of `count_sheep`
SFP of `count_sheep`
`num_sheep`

Assumptions:
- All memory safety defenses are disabled.
- There is a copy of `SHELLCODE` at address `0xdeadbeef`.
- There is no compiler padding.
- RIP of `sleep` is located at address `0xffffd634`.

Q11.1 (0.5 points) What goes in blank (1) in the stack diagram above?

Ⓐ `dream` Ⓑ `input` Ⓒ `num_sheep` Ⓓ RIP of `sleep` Ⓔ SFP of `sleep`

Q11.2 (0.5 points) What goes in blank (2) in the stack diagram above?

Ⓐ `dream` Ⓑ `input` Ⓒ `num_sheep` Ⓓ RIP of `sleep` Ⓔ SFP of `sleep`

Q11.3 (0.5 points) What goes in blank (3) in the stack diagram above?

Ⓐ `dream` Ⓑ `input` Ⓒ `num_sheep` Ⓓ RIP of `sleep` Ⓔ SFP of `sleep`

Q11.4 (1.5 points) Which of the following memory safety vulnerabilities are present in the above code?

[A] Stack Buffer Overflow [C] Off-by-one Ⓔ None of the above

[B] Heap Buffer Overflow [D] Format String Vulnerability

Warning: Q11.5 and Q11.6 are very hard. Consider attempting all other questions before spending too much time on these two parts.

Q11.5 (5 points) Provide an input to the `fread` call on Line 3 that will help us execute `SHELLCODE`.

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

For the purposes of this question, the `%*u` specifier reads one argument from the stack (call it `m`), treats it as an integer, and prints `m` bytes. It then proceeds to read a second argument off the stack, but does nothing with it.

Hint: When processed by printf, `%hhn` writes one byte to the address given by the corresponding argument; the value written is the number of characters printed so far.

```
'[                    ]' + '%*u' + ('%c' * [    ]) +
[      ] + [                      ] + ('A' * [    ])
```

Q11.6 (2 points) Provide an input to the `fread` call on Line 9 that, together with your answer to the previous part, will execute `SHELLCODE`.

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

Q11.7 (1 point) Which memory safety defenses would cause the correct exploit (without modifications) to fail? Consider each choice independently.

☐ A ASLR ☐ B Non-Executable Pages ○ C None of the above

Q11.8 (1 point) Would the same exploit work if the call to `fread` on line 3 was replaced with `fgets` (with the same parameters)?

○ A Yes, because we can still write in the correct exploit input into `dream`.

○ B Yes, because `fgets` does the same thing as `fread` when given the same number of bytes to read.

○ C No, because `fgets` will overwrite the last byte of the SFP of `sleep` with a null terminator.

○ D No, because the amount of space in `dream` required to perform the exploit will no longer be sufficient.

Post-Exam Activity: The Meaning of it All.

EvanBot, in search of a greater truth that has eluded them for far too long, has fallen deep into their own mind, seemingly trapped indefinitely in their pensive state. Help EvanBot escape by answering life's great mystery:

Select your favorite topic from the cryptography portion of the course.

- Ⓐ Diffie-Hellman
- Ⓑ Encrypt-then-MAC
- Ⓒ PRNGs
- Ⓓ El Gamal
- Ⓔ MAC-then-Encrypt (You wouldn't dare)
- Ⓕ None of the above

Showcase how this topic illuminates the meaning of life (you may use words, math, drawings, etc.):

Comment Box

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