CS161
Summer 2025

Final

Name: ____________________

Student ID: ____________________

This exam is 110 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	9	14	11	10	11	5	14	5	12	9	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)

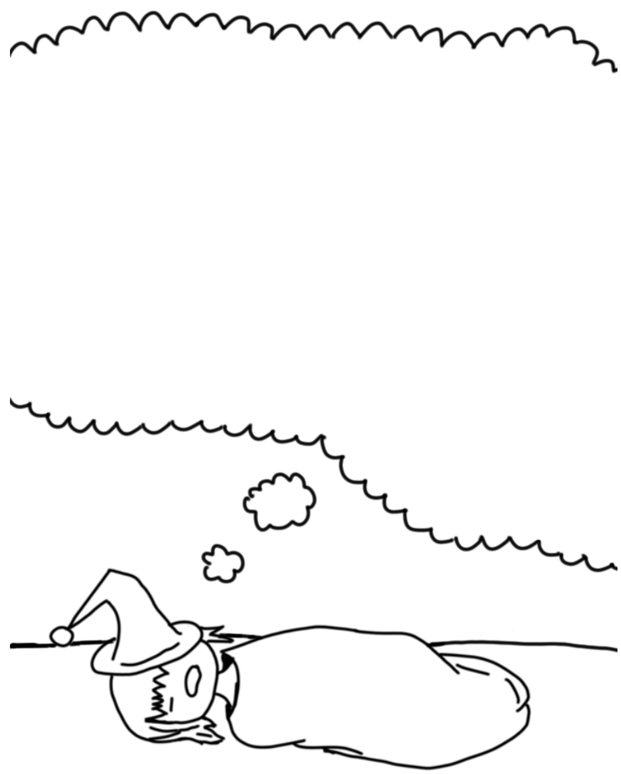
For questions with **square checkboxes**, you may select one or more choices.

- ■ 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 (0 points):

Instead of attending their final, EvanBot has chosen to sleep in. What is bot dreaming about?



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* 🍲 **(9 points)**

Each true/false is worth 0.5 points.

Q2.1 EvanBot protects their network by deploying a firewall, a NIDS, and a HIDS.

True or False: The relevant security principle is Defense-in-Depth.

○ True ○ False

Q2.2 True or False: In C, if `uint8_t x = 255` and we run `x += 1`, then `x` is now `256`.

○ True ○ False

Q2.3 True or False: A programming language that enforces type checks (e.g. strings cannot be assigned to ints) is guaranteed to be memory-safe.

○ True ○ False

Q2.4 True or False: In x86, the `call` instruction pushes the return address onto the stack and transfers control to the callee.

○ True ○ False

Q2.5 True or False: In x86, when executing a `push` instruction, the CPU increments `ESP` by 4 and writes the value to the stack.

○ True ○ False

Q2.6 True or False: Non-executable pages prevents attackers from overwriting function pointers.

○ True ○ False

Q2.7 True or False: ECB mode encryption is IND-CPA secure for single-block messages.

○ True ○ False

Q2.8 True or False: A successful HMAC verification alone is sufficient to establish both the integrity and the confidentiality of a file.

○ True ○ False

Q2.9 True or False: A fast cryptographic hash function like SHA-256 alone is sufficient for securely storing passwords.

○ True ○ False

Q2.10 True or False: Setting `HttpOnly=True` on a cookie prevents it from being sent in CSRF attacks.

○ True ○ False

Q2.11 True or False: CSRF tokens reliably mitigate CSRF on state-changing `POST` requests.

○ True ○ False

Q2.12 TRUE OR FALSE: Setting `Secure=True` on a cookie prevents it from ever being sent over `HTTPS`.

◯ TRUE ◯ FALSE

Q2.13 TRUE OR FALSE: Parametrized SQL will always prevent an SQL injection attack from succeeding.

◯ TRUE ◯ FALSE

Q2.14 TRUE OR FALSE: HTTP traffic runs over TLS, whereas HTTPS traffic runs directly over TCP.

◯ TRUE ◯ FALSE

Q2.15 TRUE OR FALSE: UDP uses sequence numbers to ensure correct packet ordering.

◯ TRUE ◯ FALSE

Q2.16 TRUE OR FALSE: TCP's three-way handshake provides built-in authentication of the communicating peers.

◯ TRUE ◯ FALSE

Q2.17 TRUE OR FALSE: SYN cookies mitigate TCP SYN flooding attacks.

◯ TRUE ◯ FALSE

Q2.18 TRUE OR FALSE: DNS over HTTPS ensures that the recursive resolver can never modify queries.

◯ TRUE ◯ FALSE

Q2.19 (0 points) TRUE OR FALSE: `EvanBot is a real bot?`

◯ TRUE ◯ FALSE

Q3 *Memory Safety: Elementary, my dear Watson* (14 points)

Consider the following vulnerable C code:

```
1  void sherlock() {
2      char buf[16];
3      int shell_ptr = 0xdeadbeef;
4      char user_input[4];
5      fgets(user_input, 4, stdin);
6
7      buf[16] -= user_input[2];
8  }
9
10 void watson(){
11     sherlock();
12 }
13
14 int main() {
15     watson();
16     return 0;
17 }
```

Stack at Line 4

RIP of `main`
SFP of `main`
RIP of `watson`
SFP of `watson`
(1)
(2)
(3)
(4)
`user_input`

Assumptions:

- The goal is to execute shellcode located at address `0xdeadbeef`.
- We run GDB once and find that the RIP of `sherlock` is at address `0xffffdc80`.
- All memory safety mitigations are disabled.

Q3.1 (1 point) What values go in blanks (1) through (4) in the stack diagram above?

- ○ (1) RIP of `sherlock` (2) SFP of `sherlock` (3) `shell_ptr` (4) `buf`
- ○ (1) `shell_ptr` (2) RIP of `sherlock` (3) SFP of `sherlock` (4) `buf`
- ○ (1) RIP of `sherlock` (2) SFP of `sherlock` (3) `buf` (4) `shell_ptr`

Q3.2 (1 point) What type of vulnerability is present in this code?

- ○ Off-by-one
- ○ Signed/unsigned
- ○ Format string vulnerability
- ○ `ret2ret`

Q3.3 (2 points) What is the **value** (not the address) of the SFP of `sherlock`?

- ○ `0xffffdc60`
- ○ `0xffffdc74`
- ○ `0xffffdc84`
- ○ `0xffffdc70`
- ○ `0xffffdc80`
- ○ `0xffffdc90`

Q3.4 (4 points) Provide an input to **`fgets`** on Line 5 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.

Q3.5 (2 points) Which memory safety defenses would cause the correct exploit (without modifications) to fail? Consider each choice independently.

Note: For the PACs option only, assume the system is 64-bit (the exploit remains unchanged).
Note: Assume the shellcode is in the code section of memory.

- ☐ Stack canaries
- ☐ Pointer authentication codes (PACs)
- ☐ Non-executable pages
- ◯ None of the above

Q3.6 (3 points) Which **values** of the SFP of **`sherlock`** would cause the correct exploit (without modifications) to fail? Select all that apply.

- ☐ `0xffffdc70`
- ☐ `0xffffdc40`
- ☐ `0xffffdc10`
- ☐ `0xffffdc60`
- ☐ `0xffffdc30`
- ☐ `0xffffdc00`
- ☐ `0xffffdc50`
- ☐ `0xffffdc20`
- ◯ None of the above

Q3.7 (1 point) Would the correct exploit (without modifications) fail if ASLR is enabled?

◯ Always ◯ Sometimes ◯ Never

Q4 *Memory Safety: Chained Together* (11 points)

Consider the following vulnerable C code:

```
1  void getting_over_it() {
2    char mountain[44];
3    fread(mountain, 44, 1, stdin);
4
5    int tether = ___Q4.2___ ;
6    char* jump_queen = &mountain[4];
7    char* jump_king = &mountain[0];
8
9    int idx = 3;
10   while (idx > 0) {
11     jump_queen = jump_king + tether;
12     jump_king = jump_queen + tether;
13     idx -= 1;
14   }
15
16   memcpy(jump_king, jump_queen, 4);
17 }
```

Stack at Line 16

RIP `getting_over_it`
SFP `getting_over_it`
`mountain`
(1)
(2)
(3)
`idx`

Assumptions:

- The goal is to execute shellcode located at address `0xdeadbeef`.
- We run GDB once and find that the address of `mountain` on the stack is `0xffffde50`.
- All memory safety mitigations are disabled.

Q4.1 (1 point) What values go in blanks (1) through (3) in the stack diagram above?

- ◯ (1) `tether` (2) `jump_queen` (3) `jump_king`
- ◯ (1) `jump_king` (2) `jump_queen` (3) `tether`
- ◯ (1) `jump_queen` (2) `tether` (3) `jump_king`

In the next two subparts, provide inputs that would cause shellcode to execute.

Q4.2 (2 points) What value should be assigned to `tether` (in the blank on Line 5)?

◯ 1 ◯ 2 ◯ 4 ◯ 6 ◯ 8 ◯ 12

Q4.3 (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.

Q4.4 (2 points) Which modifications would cause the correct exploit (without modifications) to fail? Consider each choice independently.

☐ Line 5: `idx = 3` → `idx = 2`

☐ Line 6: `char* jump_queen = &mountain[4]` → `char* jump_queen = &mountain[8]`

☐ Line 16: `memcpy(jump_king, jump_queen, 4)` → `memcpy(jump_king, jump_queen, 8)`

☐ Line 16: `memcpy(jump_king, jump_queen, 4)` → `memcpy(jump_queen, jump_king, 4)`

○ None of the above

Q4.5 (1 point) Would the correct exploit (without modifications) fail if we changed Line 3 from `fread(mountain, 44, 1, stdin)` to `fgets(mountain, 44, stdin)`?

○ Yes, because `fgets` only allows you to write 43 non-null bytes into `mountain`.

○ Yes, because `fgets` stops reading when it reads a null terminator.

○ No, because our exploit does not include null terminators.

○ No, because there are no stack canaries to detect tampering.

Q4.6 (1 point) Would the correct exploit (without modifications) fail if stack canaries are enabled?

○ Yes, because the stack canary is overwritten, causing the program to crash.

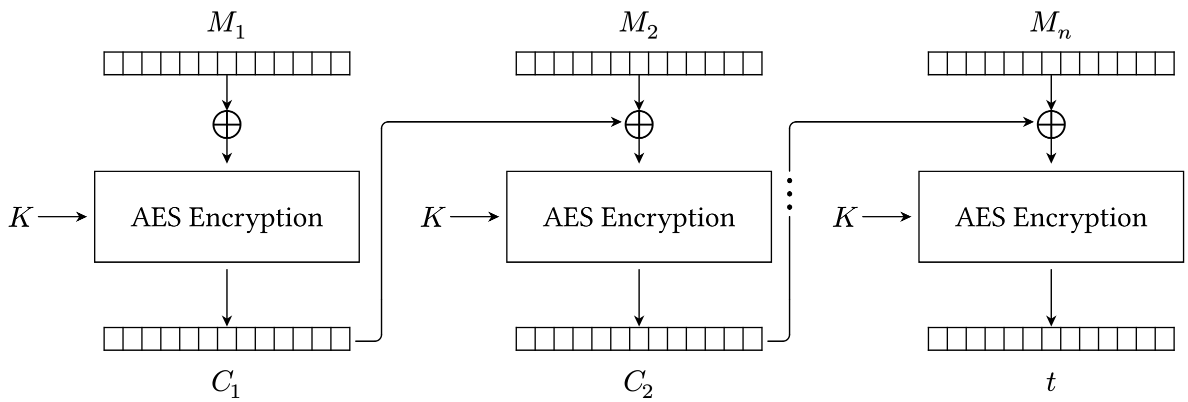
○ Yes, because the stack canary changes the number of bytes between `mountain` and the RIP.

○ No, because the stack canary is never modified.

○ No, because stack canary is overwritten but returned to its original value by the exploit.

Q5 *Cryptography: Fake It Until You MAC It* ⏰ **(10 points)**

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.



For the entire question, you may use mathematical operators, including $\oplus$, in the boxes.

Clarification During Exam: Typo in diagram, M1 is not XORed with anything.

Q5.1 (1 point) In CBC-MAC, what is the value of C_2 for a 3-block message (M_1, M_2, M_3)?

- ◯ $C_2 = E_K(M_2)$
- ◯ $C_2 = C_1 \oplus M_2$
- ◯ $C_2 = E_K(M_1 \oplus M_2)$
- ◯ $C_2 = E_K(C_1 \oplus M_2)$
- ◯ $C_2 = E_K(C_1 \oplus M_3)$
- ◯ $C_2 = C_1 \oplus E_K(M_2)$

Q5.2 (4 points) You know that message $M = (M_1, M_2)$ has tag t, and message $M' = (M_1')$ has tag t'. You do not know K.

Construct a three-block message M_{new} with the same tag as M (i.e. with the tag t).

Your answer can include $M_1,\ M_2,\ M_1',\ t,\ t'$.

$$M_{\text{new}} = \big(\boxed{}\ ,\ \boxed{}\ ,\ \boxed{} \big)$$

First block Second block Third block

Q5.3 (5 points) You know that message $M = (M_1, M_2, M_3)$ has tag t, and message $M' = (M'_1, M'_2, M'_3)$ has tag t'. You do not know K.

You want to forge a message M_{new} with the same tag as M (i.e. with the tag t).

To help with your forgery, you can query for the MAC of two messages before constructing M_{new}. In each blank, you may use: $M_1,\ M_2,\ M_3,\ M'_1,\ M'_2,\ M'_3,\ t,\ t'$.

Hint: In our solution, both messages are one block each.

What is the first message you query for?

[ ]

The MAC of the message in the box above is t_a.

What is the second message you query for?

[ ]

The MAC of the message in the box above is t_b.

Now, construct a three-block message M_{new} with the same tag as M (i.e. with the tag t):

- Your answer can include $M_1,\ M_2,\ M_3,\ M'_1,\ M'_2,\ M'_3,\ t,\ t',\ t_a,\ t_b$.
- Your answer cannot be exactly (M_1, M_2, M_3), (M'_1, M'_2, M'_3), or the queries in the boxes above.

$M_{\text{new}} = ($ [ ] , [ ] , [ ] $)$

First block | Second block | Third block

Q6 *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.
6. Alice computes $k_0 \equiv$ __________ (Q6.1) and $k_1 \equiv$ __________ (Q6.2).
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 $m_b \equiv$ __________ (Q6.3).

Q6.1 (1 point) Provide the value for k_0 in step 6.

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

Q6.2 (1 point) Provide the value for k_1 in step 6.

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

Q6.3 (1 point) Provide the value for m_b in step 8.

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

Q6.4 (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.

Subparts Q6.5 to Q6.8 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: $K_{a=0},\ K_{a=1},\ K_{b=0},\ K_{b=1}$.
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 ________________ (Q6.5).

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.

Q6.5 (1 point) Fill in the blank for step 6 above.

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

Q6.6 (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.

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 ____________________ (Q6.5).

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.

Q6.7 (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.

Q6.8 (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.

○ $\mathsf{Dec}(K_b,\ \mathsf{Dec}(K_{a=0}, C))$

○ $\mathsf{Dec}(K_b,\ \mathsf{Dec}(K_{a=1}, C))$

○ $\mathsf{Dec}(K_{b=0},\ \mathsf{Dec}(K_a, C))$

○ $\mathsf{Dec}(K_{b=1},\ \mathsf{Dec}(K_a, C))$

Q6.9 (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

Q7 *Web Security: Many links lead to EvanRome* 🏛️🏺 **(5 points)**

For each subpart, select the URL with the same origin as the given URL, according to Same Origin Policy.

Q7.1 (1 point) **`https://www.cs161.org:443/policies`**

- ◯ **`https://su25.cs161.org`**
- ◯ **`https://sp25.cs161.org:161`**
- ◯ **`http://evil.mallory.com`**
- ◯ None of the above

Q7.2 (1 point) **`http://sp25.cs161.org:161/policies`**

- ◯ **`https://su25.cs161.org`**
- ◯ **`https://sp25.cs161.org:161`**
- ◯ **`http://evil.mallory.com`**
- ◯ None of the above

Q7.3 (1 point) **`https://sp25.cs161.org:161/policies`**

- ◯ **`https://su25.cs161.org`**
- ◯ **`https://sp25.cs161.org:161`**
- ◯ **`http://evil.mallory.com`**
- ◯ None of the above

Q7.4 (1 point) **`http://evil.mallory.org:80/policies`**

- ◯ **`https://su25.cs161.org`**
- ◯ **`https://sp25.cs161.org:161`**
- ◯ **`http://evil.mallory.com`**
- ◯ None of the above

Q7.5 (1 point) **`http://su25.cs161.org:80/attack`**

- ◯ **`https://su25.cs161.org`**
- ◯ **`https://sp25.cs161.org:161`**
- ◯ **`http://evil.mallory.com`**
- ◯ None of the above

Q8 *Web Security: Mallory-PT* 🤖 **(14 points)**

A new trend is sweeping the nation — everyone is chatting away using ClosedAI's new product: GPTChat! When a user logs in at `gpt.chat`, they can communicate with a chat bot.

Assumptions:

- `gpt.chat` uses session-based authentication. Session tokens are stored as cookies with:
  `Name=token; Domain=gpt.chat; Path=/; HttpOnly=False; Secure=True`.
- `gpt.chat` hosts many chat bots. Users can select which bot to chat with, by setting a `bot` cookie with:
  `Name=bot;   Domain=gpt.chat; Path=/; HTTPOnly=False; Secure=True`.
  The `Value` is the URL of the selected bot, e.g. `Value=gpt.chat/evan` or `Value=gpt.chat/coda`.

Users logged into `gpt.chat` can access these paths:

Path	Method	Description
`/chat`	`GET`	Returns a chat HTML page containing: • The `CHAT_ID` for this chat. • A space where messages are displayed unsanitized. • A chat bar. When a user presses [Enter] a `POST` request is made to `/prompt`, and the bot's response is added to the space.
`/prompt`	`POST`	Forwards the body of the `POST` request to the URL in the `bot` cookie. Returns the response from that URL to the user as HTML.
`/share?id=CHAT_ID`	`GET`	Loads a read-only version of the chat with the given `CHAT_ID`. If the `CHAT_ID` is invalid, loads this unsanitized HTML, replacing `CHAT_ID` with the URL parameter: CHAT_ID invalid.
`/list`	`GET`	Returns a list of the user's chats. Each entry has a `CHAT_ID` and a link to the chat.

Mallory controls a server at `mallory.com` with these paths:

Path	Method	Description
`/store`	`GET/POST`	Mallory will record any data sent here.
`/post`	`POST`	Mallory can respond to the `POST` request with any data she wants.

JavaScript functions you can use in this question:

- `get(url)`: Executes a `GET` request to the provided URL.
- `post(url, body)`: Executes a `POST` request to the provided URL with the provided body.
- `updateCookie(name, value)`: Sets the value of the cookie with name `name` to `value`.
  Only works if JavaScript has access to the cookie in question. All other flags remain the same.

Q8.1 (1 point) Suppose `mallory.com/chat` returns JavaScript that makes a `POST` request to `gpt.chat/prompt`, with some malicious message.

When a user logged into GPTChat visits `mallory.com/chat`, will the `POST` request succeed?

- ◯ Yes, cookie policy considers the URL of the request, so the `session` cookie will be sent.
- ◯ Yes, session cookies are attached to all HTTP requests.
- ◯ No, cookie policy considers the origin of the request, so the `session` cookie will not be sent.
- ◯ No, the `Secure` flag on the `session` cookie will prevent it from being attached.

Q8.2 (1 point) For this subpart, Mallory is an on-path attacker between a logged-in user and GPTChat.

The user opens each of these URLs. Select all URLs that will leak their `session` token to Mallory.

- ☐ `https://gpt.chat`
- ☐ `http://gpt.chat`
- ☐ `https://mallory.com/store`
- ☐ `http://mallory.com/store`
- ☐ `https://fake.gpt.chat`
- ☐ `http://fake.gpt.chat`
- ◯ None of the above

Q8.3 (4 points) Construct a URL that, when clicked, sends all of a user's `CHAT_ID`s to Mallory.

Q8.4 (3 points) `/prompt` does not check the URL in the `bot` cookie before forwarding to that URL.

Mallory exploits this by designing an attack:

1. She writes some JavaScript: `<script>__________</script>`.
2. The user runs this script with GPTChat's origin.
3. The user opens `/chat`.
4. Now, Mallory can add responses to the `/chat` page as if she was the bot.

What goes in the blank to achieve Mallory's attack?

Q8.5 (2 points) Select all actions Mallory can do after executing the attack in Q8.4.

☐ Read messages that the user types in the chat bar.

☐ Add any HTML of Mallory's choosing on the `/list` page.

☐ Make the user run malicious JavaScript with the origin of `gpt.chat`.

☐ Make the user run malicious JavaScript with the origin of `bank.com` (a secure site).

☐ Learn information about any other tab that the user has open.

○ None of the above

Q8.6 (2 points) Mallory uses reflected XSS to make the user run her script in Q8.4.

Select all defenses that would prevent Mallory's attack in Q8.4.

☐ Origin/Referer checking

☐ Input sanitization

☐ CSRF tokens

☐ Content security policy

☐ Prepared statements

☐ Setting `HttpOnly=True` for all cookies

☐ Using the `SameSite` flag

○ None of the above

Q8.7 (1 point) Mallory now designs an attack to cause Alice to run malicious JavaScript:

1. Mallory runs the attack in Q8.4 on herself.
2. Mallory sends a response with malicious JavaScript to herself.
3. Mallory copies the `/share?id=CHAT_ID` link for the chat with malicious JavaScript.
4. Mallory sends the link to Alice, and Alice clicks the link.

Which type of attack is executed on Alice?

○ CSRF attack

○ Stored XSS

○ Buffer overflow

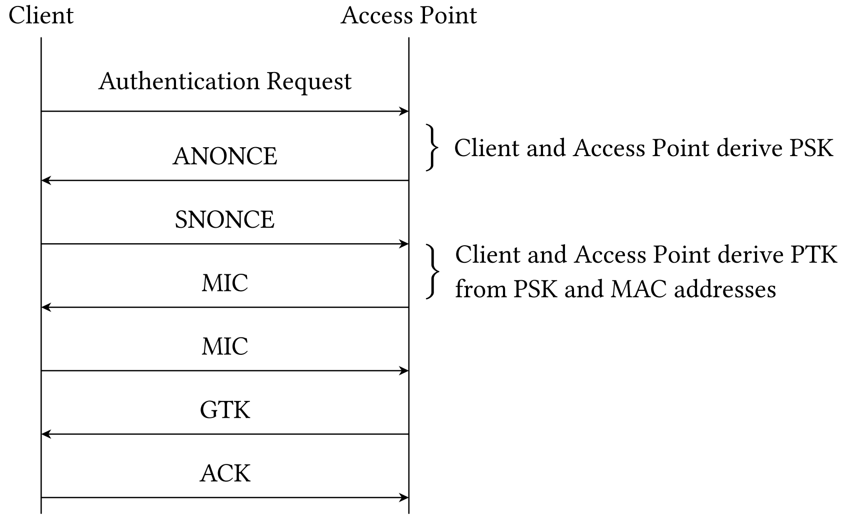
○ Reflected XSS

○ SQL injection

○ None of the above

Q9 *Networking: Hodgepodge* ✱ **(5 points)**

The WPA2-PSK scheme from lecture is shown below. Each subpart is independent.



Clarification During Exam: In the diagram, the PTK should be derived from the PSK, MAC addresses, and Nonces.

Q9.1 (2 points) An attacker records an entire session (WPA handshake and subsequent messages) between a client and access point. Later, the attacker learns the network's PSK. Select all true statements.

- ☐ The attacker can decrypt the messages in the recorded session.
- ☐ The attacker can derive the PTK used in the recorded session.
- ☐ The attacker can derive the GTK used in the recorded session.
- ☐ The attacker can decrypt future recorded sessions between other clients and the access point.
- ◯ None of the above

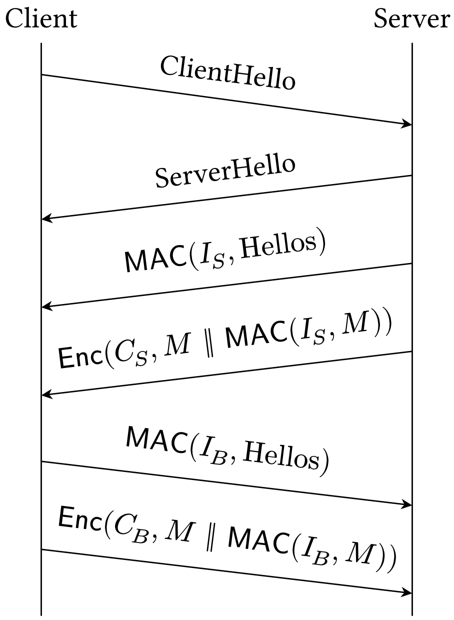
Q9.2 (1 point) What would happen if the client sent the same SNonce value in multiple handshakes with the same access point?

- ◯ A different PTK is derived in each handshake.
- ◯ The same PTK is derived in each handshake.
- ◯ A different PSK is derived in each handshake.
- ◯ A different GTK is derived in each handshake.

Q9.3 (2 points) Suppose the Wi-Fi password is changed once per hour. Select all true statements.

- ☐ Users joining at different hours will derive different PSKs.
- ☐ Users joining at different hours will derive different PTKs.
- ☐ Users joining at different hours will use different GTKs.
- ☐ Every hour, existing users' PTKs become invalid, and users must re-join the network.
- ◯ None of the above

Q10 *Networking: TLSplit* 🔨 **(12 points)**



Consider the modified TLS handshake shown in the diagram:

1. **ClientHello**: Client sends $g^a \bmod p$ (ephemeral Diffie-Hellman value) and R_B (random nonce).
2. **ServerHello**: Server sends $g^b \bmod p$ (ephemeral Diffie-Hellman value) and R_S (random nonce).
3. The Client and Server derive the symmetric keys (I_S, I_B, M_S, M_B) using the premaster secret $g^{ab} \bmod p$ and the random nonces R_B, and R_S.
4. The Server sends the MAC on both Hello messages.
5. The Server MACs and encrypts a message and sends it to the Client.
6. The Client sends the MACs on the Hello messages.
7. The Client MACs and encrypts a message and sends it to the Server.

Note: The version of this question featured in the exam contained answer choices that referred to variables from an older version of the question. This was clarified, and has been corrected for this version of the exam.

Q10.1 (1 point) True or False: This scheme ensures forward secrecy.

○ True ○ False

Q10.2 (1 point) True or False: This scheme guarantees that the client is talking to the legitimate server.

○ True ○ False

Suppose the Client and Server start a connection in the presence of Mallory. Mallory is a man-in-the-middle attacker who wants to send messages to the client after the TLS handshake is completed.

Q10.3 (1 point) After the ClientHello has been recieved, Mallory replaces the ServerHello. What values should Mallory send to the client?

○ $g^a \bmod p$ and R_m ○ $g^m \bmod p$ and R_m ○ $g^b \bmod p$ and R_m ○ R_m

Q10.4 (2 points) After Q10.3, what values will the **Client** use to derive the symmetric keys in Step 3?

☐ a ☐ $g^a \bmod p$ ☐ R_B

☐ m ☐ $g^m \bmod p$ ☐ R_m

☐ b ☐ $g^b \bmod p$ ☐ R_S

Q10.5 (2 points) After Q10.3, what values will the **Server** use to derive the symmetric keys in Step 3?

- ☐ a
- ☐ $g^a \bmod p$
- ☐ R_B
- ☐ m
- ☐ $g^m \bmod p$
- ☐ R_m
- ☐ b
- ☐ $g^b \bmod p$
- ☐ R_S

Q10.6 (2 points) After Q10.3, what values will **Mallory** use to derive the same symmetric keys as the Client in Step 3?

- ☐ a
- ☐ $g^a \bmod p$
- ☐ R_B
- ☐ m
- ☐ $g^m \bmod p$
- ☐ R_m
- ☐ b
- ☐ $g^b \bmod p$
- ☐ R_S

Q10.7 (1 point) After Step 4 of the TLS handshake is complete, what can Mallory do? Select all that apply.

- ☐ Pretend to be the Server and send the message in Step 5 to the Client.
- ☐ Pretend to be the Client and send the message in Step 7 to the Server.
- ○ None of the above

For Q10.8 and Q10.9, consider the standard TLS handshake from lecture (these subparts are **independent** from the modified scheme above).

Q10.8 (1 point) The Client and Server complete a standard TLS handshake. If an attacker compromises all routers between the Client and the Server, can they decrypt messages?

- ○ Yes, because the compromised routers can inspect and forward packets.
- ○ Yes, because the attacker can inject traffic to downgrade encryption and then decrypt.
- ○ No, because TLS is end-to-end secure.
- ○ No, because the underlying TCP session provides confidentiality.

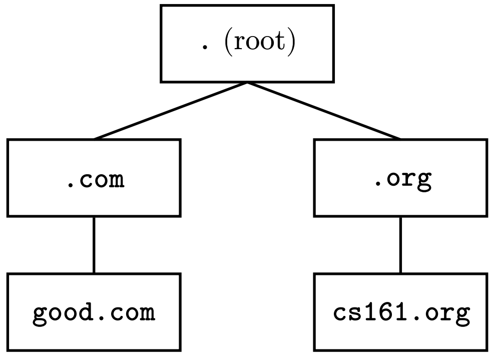
Q10.9 (1 point) Suppose the Client and Server change the length of the random nonce from 256 to 128 bits in the TLS handshake.

With this modification, what happens to the probability that a packet recorded from one connection can be replayed in another connection?

- ○ The probability increases, and the resulting probability of success is non-negligible.
- ○ The probability increases, and the resulting probability of success is negligible.
- ○ The probability decreases, and the resulting probability of success is non-negligible.
- ○ The probability decreases, and the resulting probability of success is negligible.

Q11 *Networking: GooDNS* 🫢 **(9 points)**

Consider this DNS hierarchy, where each box represents a name server:



EvanBot has the following records cached:

Record 1:	`org.`	`NS`	`a.org-servers.net`
Record 2:	`a.org-servers.net`	`A`	`192.7.14.21`
Record 3:	`com.`	`NS`	`a.com-servers.net`
Record 4:	`a.com-servers.net`	`A`	`192.6.16.161`
Record 5:	`evil.com`	`A`	`192.5.55.555`

In Q11.1 to Q11.3, each subpart continues on from previous subparts, i.e. records received in one subpart can be cached for later subparts.

Q11.1 (1 point) How many DNS requests does EvanBot need to make to learn the IP address of **`www.cs161.org`**?

○ 0 ○ 1 ○ 2 ○ 3

Q11.2 (1 point) Record 1 is expired and removed from the cache.

How many DNS requests does the EvanBot need to make to learn the IP address of **`www.cs161.org`**?

○ 0 ○ 1 ○ 2 ○ 3

Q11.3 (1 point) How many DNS requests does EvanBot need to make to learn the IP address of **`not.good.com`**?

○ 0 ○ 1 ○ 2 ○ 3

The rest of the question is independent of earlier subparts.

Q11.4 (1 point) Which of these best describes why an attacker would use the Kaminsky attack, instead of some other cache poisoning attack?

- ◯ The attacker is on-path; the Kaminsky attack only works for on-path attackers.
- ◯ Unlike other DNS attacks, the Kaminsky attack can poison a cache shared by many users.
- ◯ The attacker is off-path; the Kaminsky attack guarantees that the attacker will guess correctly.
- ◯ The attacker is off-path; the Kaminsky allows the attacker to make more guesses.

Q11.5 (2 points) Suppose the attacker can place HTML on a website that the victim will visit.

Which HTML snippets can help the attacker poison the cache for **`www.google.com`** (using the Kaminsky attack)? Select all that apply.

- ☐ **`<img src="false1.google.com">`**
- ☐ **`<img src="false1.google.com/image.png">`**
- ☐ **`<img src="false1.cs161.org">`**
- ☐ **`<img src="www.google.com">`**
- ◯ None of the above

Q11.6 (1 point) When source port randomization is enabled, what is the approximate probability that an off-path attacker successfully spoofs a DNS response?

◯ $1/2^{16}$ ◯ $1/2^{32}$ ◯ $1/2^{64}$ ◯ 1 ◯ 0

Q11.7 (1 point) When executing a Kaminsky attack, what should be the source IP in the attacker's spoofed DNS response?

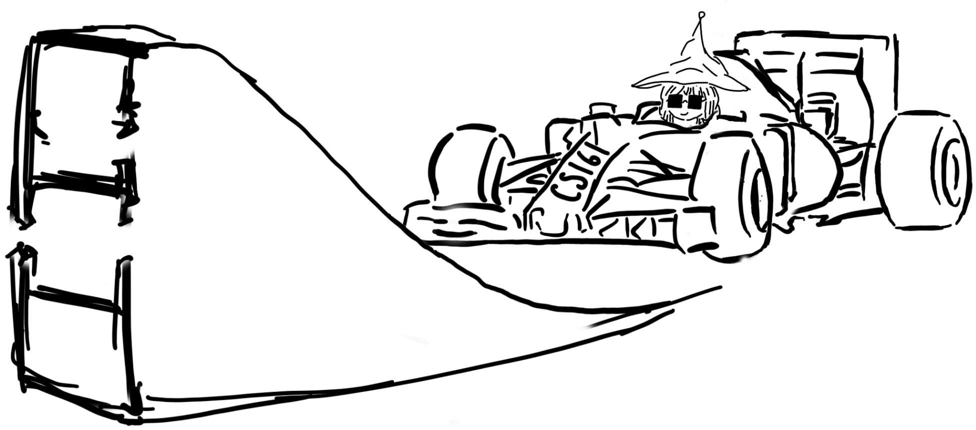
- ◯ Attacker's IP address
- ◯ Name server's IP address
- ◯ Resolver's IP address
- ◯ The source IP field can be left blank.

Q11.8 (1 point) What is the primary reason DNSSEC does not provide confidentiality?

- ◯ The trust anchor model used by DNSSEC is incompatible with encryption protocols.
- ◯ Confidentiality would make the DNS query process too slow.
- ◯ DNS data is considered public information.
- ◯ Implementing encryption would require a new set of DNS record types, which is not feasible.

Post-Exam Activity: Bot's Broken Ramp

Oh no! The ramp is broken! Draw in CS161 Course staff in order by height to hold up the ramp, so that EvanBot can drive over safely.



Comment Box

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