Nicholas & Peyrin
Summer 2021

CS 161
Computer Security

Final

For questions with **circular bubbles**, you may select exactly *one* choice on Examtool.

○ Unselected option

● Only one selected option

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

■ You can select

■ multiple squares

For questions with a **large box**, you need to write your answer in the text box on Examtool.

There is an appendix at the end of this exam, containing descriptions of all C functions used on this exam.

You have 170 minutes, plus a 10-minute buffer for distractions or technical difficulties, for a total of 180 minutes. There are 12 questions of varying credit (200 points total).

The exam is open note. You can use an unlimited number of handwritten cheat sheets, but you must work alone.

Clarifications will be posted on Examtool.

Q1 *MANDATORY – Honor Code* (5 points)

Read the following honor code and type your name on Examtool.

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 and a corresponding notch on Nick's Stanley Fubar demolition tool.

Solution: Everyone gets 5 free points for making it to the end of the semester!

Q2 *True/false* (30 points)

Each true/false is worth 2 points.

Q2.1 TRUE or FALSE: TLS with Diffie-Hellman is still vulnerable to man-in-the-middle attackers by intercepting and performing two separate key exchanges with both sides.

○ TRUE

● FALSE

> **Solution:** False. In TLS, the server's public parameter $g^a \bmod p$ is signed by the server, so an attacker cannot modify this value.

Q2.2 TRUE or FALSE: In TLS, an attacker cannot change the client and server random values R_b and R_s sent at the beginning of the exchange undetected, even though the client and server have not yet agreed on a set of symmetric keys.

● TRUE

● FALSE

> **Solution:** True. MACs over the entire dialogue are sent at the end of the handshake in order to verify that no communication was tampered with. If the attacker tampers with R_b or R_s, the exchanged MACs will not match, and the tampering will be detected.
>
> After the exam, we decided that the wording on this question was ambiguous (particularly the phrase about not yet agreeing on a set of symmetric keys), so this question was dropped, and all students received credit.

Q2.3 TRUE or FALSE: Referrer validation is a valid defense against reflected XSS attacks.

Clarification during exam: This question has been dropped. All students will receive credit on this question.

● TRUE

● FALSE

> **Solution:** The intended solution was false: Referrer validation is a defense against CSRF attacks.
>
> However, referrer validation could stop some reflected XSS attacks. For example, if the victim clicks on a reflected XSS link on the attacker's website, the server could see the request came from the attacker's website and reject the request.
>
> Because the answer was ambiguous, we dropped this question and gave everyone full points.

Q2.4 TRUE or FALSE: Hybrid encryption typically uses symmetric encryption to encrypt the plaintext and asymmetric encryption to encrypt a symmetric key.

● TRUE

○ FALSE

Solution: True. Hybrid encryption uses symmetric encryption to encrypt the message because it is fast and can handle arbitrary-length ciphertexts, and asymmetric encryption is only used to encrypt the symmetric key because it is slow and can only handle ciphertexts up to a certain length.

Q2.5 TRUE or FALSE: If the first and last node in a Tor circuit learn that they are part of the same circuit and collude with one another, they are able to deanonymize the Tor user, even if the middle node is honest.

● TRUE ○ FALSE

Solution: True. The middle node of the Tor circuit exists to defend against the attack where the entry node directly contacts the exit node to collude. If they are able to bypass this, then the anonymity guarantees are lost.

Q2.6 TRUE or FALSE: One advantage of pointer authentication over stack canaries is that pointer authentication is harder to brute-force than stack canaries.

○ TRUE ● FALSE

Solution: False. A brute-force attack is easier with pointer authentication because there are fewer unused bits in a pointer authentication code (PAC) (around 25 bits on a 64-bit system) compared to the randomized bytes of the stack canary (56 bits on a 64-bit system).

Q2.7 TRUE or FALSE: While developing a website, the programmer leaves the server password hidden in the website's HTML. This violates Shannon's Maxim.

● TRUE ○ FALSE

Solution: True. The programmer is relying through security through obscurity, which is a violation of Shannon's Maxim.

Q2.8 TRUE or FALSE: Most CAPTCHAs are set up to distinguish good bots, such as web crawlers, from malicious bots.

○ TRUE ● FALSE

Solution: False. CAPTCHAs can only distinguish between bots and humans.

Q2.9 TRUE or FALSE: Cursorjacking is a UI attack that relies on creating a fake cursor that is more prominent or visible than the real cursor.

● True ○ False

Solution: True. This is the definition of cursorjacking.

Q2.10 True or False: In a program that has stack canaries enabled, the stack canary takes on the same value across multiple executions of the program.

○ True ● False

Solution: False. The stack canary takes on a different value every time the program is run.

Q2.11 True or False: If Alice encrypts a message with AES-CBC, but instead of using completely random IVs, she uses R, $R+1$, $R+2$, and so on, where R is a random value that she chose once, this scheme is IND-CPA secure.

○ True ● False

Solution: False. As seen in discussion, if the attacker can predict future IVs in AES-CBC, then AES-CBC is not IND-CPA secure.

Q2.12 True or False: DNSSEC uses TLS to protect against attacks when transmitting packets.

○ True ● False

Solution: False. DNSSEC uses UDP and offline signatures protect records. (Recall that DNSSEC is supposed to be fast and lightweight, and TLS is slower because of a long handshake.)

Q2.13 True or False: SHA-256 is a good hash function to use when hashing passwords.

○ True ● False

Solution: False. SHA256 is a fast hash function, which would allow offline password guessing attacks to be done efficiently. Instead, a slow hash function such as PBKDF2 or Argon2 would be better since it makes brute-force attacks mostly infeasible.

Q2.14 True or False: A firewall does not defend against a trusted party inside the firewall that becomes malicious and attempts to breach other computers within the network.

● True ○ False

Solution: True. Firewalls assume that users inside the firewall are trusted, so a malicious user inside the network has unlimited power to attack other machines inside the network.

Q2.15 True or False: One weakness of one-time authentication codes, such as those sent to a user's cell phone number, is that they are still vulnerable to a transient phishing attack.

● True ○ False

Solution: True. Transient phishing attacks simply trick the user into entering the one-time code sent by the legitimate service, and then the attackers use the code to access the account themselves.

Q2.16 (0 points) True or False: `EvanBot is a real bot.`

● True ○ False

Solution: True. EvanBot always fails CAPTCHAs.

Q3 *Piazza Policy* (18 points)

Q3.1 (5 points) Which of the following URLs have the same origin as `http://piazza.com/`? Select all that apply.

☐ (A) `https://piazza.com/`

☐ (B) `http://piazza.com:614/`

☐ (C) `http://web.piazza.com/`

■ (D) `http://piazza.com/run`

☐ (E) `http://aplaza.com/`

☐ (F) None of the above

Solution: Recall that the same-origin policy requires that the protocol, domain, and port number must match. The port number is assumed to be 80 for HTTP and 443 for HTTPS.

A: False. The protocols (and default port numbers) don't match.

B: False. The port numbers don't match.

C: False. The domain names don't match.

D: True. The protocols, port numbers, and domain names all match.

E: False. The domain names don't match.

Q3.2 (3 points) If the script `<script src="http://cs161.org/tracking.js"></script>` is included in `http://piazza.com/`, which of these pages can the script modify?

● (G) `http://piazza.com/`

○ (H) `http://bank.com/`

○ (I) `http://cs161.org/`

○ (J) All of the above

○ (K) None of the above

○ (L) —

Solution: JavaScript has the origin of the page that loads it, so the script can only modify pages on the `http://piazza.com` origin. Only the selected answer choice matches the protocol, domain name, and port number.

Q3.3 (5 points) In which of the following contexts would the contents of a cookie with the HttpOnly flag be sent? Select all that apply.

■ (A) A user clicks on a link that sends a request over HTTP to the domain of the cookie

■ (B) A user clicks on a link that sends a request over HTTPS to the domain of the cookie

■ (C) JavaScript is used to send a request over HTTP to the domain of the cookie

□ (D) JavaScript is used to read the value of the cookie and send its contents to a different domain

■ (E) A page includes an <img> tag loading an image using HTTP from the domain of the cookie

□ (F) None of the above

Solution: HttpOnly only prevents JavaScript from directly reading the value of a cookie, but it does not prevent it being sent as a cookie in in any requests to the domain, even if the requests are made in JavaScript. Thus, only option (D) is incorrect because JavaScript attempts to directly read the value of the cookie.

Q3.4 (5 points) `http://evanbot.piazza.com` is setting a cookie. Which of these cookie attributes can `http://evanbot.piazza.com` set (without being rejected by the browser) so that the cookie gets sent on a request to `http://web.piazza.com/pictures`? Select all that apply.

■ (G) `domain=evanbot.piazza.com; path=/pictures`

■ (H) `domain=piazza.com; path=/pictures`

□ (I) `domain=com; path=/pictures`

■ (J) `domain=evanbot.piazza.com`

□ (K) `domain=piazza.evanbot.com; path=/pictures`

□ (L) None of the above

Solution: Recall that cookie policies requires that the domain of the cookie is a domain suffix of the request and that the path of the cookie is a path prefix of the request.

G: False. `evanbot.piazza.com` is not a domain suffix of `web.piazza.com`.

H: True. `piazza.com` is a domain suffix of `web.piazza.com`, and `/pictures` is a path prefix of itself.

I: False. Top-level domains such as `com` are not a valid cookie domain.

J: False. `evanbot.piazza.com` is not a domain suffix of `web.piazza.com`.

K: False. `piazza.evanbot.com` is not a domain suffix of `web.piazza.com`.

Q4 *Coffee-Shop Attacks* (17 points)

Dr. Yang comes to MoonBucks and tries to connect to the network in the coffee shop. Dr. Yang and `http://www.piazza.com` are communicating through TCP. Mallory is an on-path attacker.

Q4.1 (5 points) Which of the following protocols are used when Dr. Yang first connects to the Wi-Fi network and visits `http://www.piazza.com`? Assume any caches are empty. Select all that apply.

- ☐ (A) CSRF
- ■ (B) IP
- ■ (C) DNS (or DNSSEC)
- ■ (D) HTTP
- ■ (E) DHCP
- ☐ (F) None of the above

> **Solution:**
>
> A: False. CSRF is not a protocol, but a web attack.
>
> B: True. IP is used to send messages across the internet and is used by TCP, which is used by TLS, which is used by HTTPS.
>
> C: True. DNS is used to look up the IP address of `www.piazza.com`.
>
> D: True. HTTP is the application protocol being used.
>
> E: True. DHCP is used to receive the initial network configuration for the client.

Q4.2 (3 points) Suppose Mallory spoofs a packet with a valid, upcoming sequence number to inject the malicious message into the connection. Would this affect other messages in the connection?

- ● (G) Yes, because the malicious message replaces some legitimate message
- ○ (H) Yes, because future messages will arrive out of order
- ○ (I) No, because on-path attackers cannot inject packets into a TCP connection
- ○ (J) No, because TCP connections are encrypted
- ○ (K) ——
- ○ (L) ——

> **Solution:** When the server receives the original TCP packet whose sequence number was used by Mallory, the server will ignore it, thinking that it has already received its data and that it was retransmitted.

Q4.3 (3 points) To establish a TCP connection, Dr. Yang first sends a SYN packet with Seq $= 980$ to the server and receives a SYN-ACK packet with Seq $= 603$; Ack $= 981$. What packet should Dr. Yang include in the next packet to complete the TCP handshake?

- ○ (A) SYN-ACK packet with Seq $= 981$; Ack $= 604$

○ (B) SYN-ACK packet with Seq $= 604$; Ack $= 981$

● (C) ACK packet with Seq $= 981$; Ack $= 604$

○ (D) ACK packet with Seq $= 604$; Ack $= 981$

○ (E) Nothing to send, because the TCP handshake is already finished.

○ (F) ——

> **Solution:** This is the third step of the 3-way handshake, when the client sends an ACK packet to acknowledge the server's SYN-ACK packet.

Q4.4 (3 points) Immediately after the TCP handshake, Mallory injects a valid RST packet to the server. Next, Mallory spoofs a SYN packet from Dr. Yang to the server with headers Seq $= X$. The server responds with a SYN-ACK packet with Seq $= Y$; Ack $= X + 1$. What is the destination of this packet?

● (G) Dr. Yang

○ (H) The server

○ (I) Mallory

○ (J) None of the above

○ (K) ——

○ (L) ——

> **Solution:** The server uses the source as the destination for the SYN-ACK packet. Because Mallory spoofed the packet from the client, the response is sent to the client.

Q4.5 (3 points) Which of the following network attackers would be able to perform the same attacks as Mallory?

Clarification during exam: By "perform the same attacks," we mean "reliably perform the same attacks."

● (A) A MITM attacker between Dr. Yang and the server

○ (B) An off-path attacker

○ (C) All of the above

○ (D) None of the above

○ (E) ——

○ (F) ——

> **Solution:** A MITM attacker has all the capabilities of an on-path attacker, so it would be able to perform Mallory's attacks. An off-path attacker would be unable to guess the sequence numbers and would be unable to perform Mallory's attacks.

Q5 *Dual Asymmetry* **(15 points)**

Alice wants to send two messages M_1 and M_2 to Bob, but they do not share a symmetric key.

Clarification during exam: Assume that p is a large prime and that g is a generator $\bmod p$, like in ElGamal. Assume that all computations are done modulo p in Scheme A.

Q5.1 (3 points) Scheme A: Bob publishes his public key $B = g^b$. Alice randomly selects r from 0 to p-2. Alice then sends the ciphertext $(R, S_1, S_2) = (g^r, M_1 \times B^r, M_2 \times B^{r+1})$.

Select the correct decryption scheme for M_1:

- ● (A) $R^{-b} \times S_1$
- ○ (B) $R^b \times S_1$
- ○ (C) $B^{-b} \times S_1$
- ○ (D) $B^b \times S_1$
- ○ (E) —
- ○ (F) —

Solution:

$S_1 = M_1 \times B^r$	Given in the question
$S_1 = M_1 \times g^{br}$	Substitute $B = g^b$
$M_1 = g^{-br} \times S_1$	Multiply both sides by g^{-br}
$M_1 = R^{-b} \times S_1$	Substitute $R = g^r$

Q5.2 (3 points) Select the correct decryption scheme for M_2:

- ● (G) $B^{-1} \times R^{-b} \times S_2$
- ○ (H) $B \times R^{-b} \times S_2$
- ○ (I) $B^{-1} \times R^b \times S_2$
- ○ (J) $B^{-1} \times R \times S_2$
- ○ (K) —
- ○ (L) —

Solution:

$S_2 = M_2 \times B^{r+1}$	Given in the question
$S_2 = M_2 \times g^{b(r+1)}$	Substitute $B = g^b$
$S_2 = M_2 \times g^{br+b}$	Exponentiation properties
$M_2 = g^{-br-b} \times S_2$	Multiply both sides by g^{-br-b}
$M_2 = g^{-br} \times g^{-b} \times S_2$	Exponentiation properties
$M_2 = R^{-b} \times B^{-1} \times S_2$	Substitute $B = g^b$ and $R = g^r$
$M_2 = B^{-1} \times R^{-b} \times S_2$	Rearrange terms

Q5.3 (4 points) Is Scheme A IND-CPA secure? If it is secure, briefly explain why (1 sentence). If it is not secure, briefly describe how you can learn something about the messages.

Clarification during exam: For Scheme A, in the IND-CPA game, assume that a single plaintext is composed of two parts, M_1 and M_2.

○ (A) Secure ○ (C) —— ○ (E) ——

● (B) Not secure ○ (D) —— ○ (F) ——

Solution: This scheme is not IND-CPA secure. Eve can determine if $M_1 = M_2$ by checking if $S_2 = S_1 \times B$.

Q5.4 (5 points) Scheme B: Alice randomly chooses two 128-bit keys K_1 and K_2. Alice encrypts K_1 and K_2 with Bob's public key using RSA (with OAEP padding) then encrypts both messages with AES-CTR using K_1 and K_2. The ciphertext is $\text{RSA}(\mathsf{PK}_{\mathsf{Bob}}, K_1 \| K_2), \text{Enc}(K_1, M_1), \text{Enc}(K_2, M_2)$.

Which of the following is required for Scheme B to be IND-CPA secure? Select all that apply.

☐ (G) K_1 and K_2 must be different

■ (H) A different IV is used each time in AES-CTR

☐ (I) M_1 and M_2 must be different messages

☐ (J) M_1 and M_2 must be a multiple of the AES block size

☐ (K) M_1 and M_2 must be less than 128 bits long

☐ (L) None of the above

Solution:

G: False. Because Enc is an IND-CPA secure encryption algorithm, the key does not need to be changed between two encryptions.

H: True. AES-CTR requires that a unique nonce is used for each encryption, or it loses its confidentiality guarantees.

I: False. A secure encryption algorithm would not leak the fact that two messages are the same.

J: AES-CTR can encrypt any length of plaintext. Padding is not needed in AES-CTR.

K: AES-CTR can encrypt any length of plaintext.

Q6 *Under Pressure* (16 points)

Alice and Bob are communicating large pieces of secret data with each other using an IND-CPA secure encryption scheme, but they are being slowed down by their connection! They decide to use compression to improve the amount of data they can send.

Consider the following properties of compression algorithms:

- Compression algorithms reduce the length of the input.
- High-entropy (random-looking) data may only have its length reduced slightly, or not at all
- Low-entropy (predictable) data can often have its length reduced considerably

Q6.1 (3 points) Alice and Bob consider first encrypting and then compressing their data. They send $\mathsf{compress}(\mathsf{Enc}(K, M))$, where the input to the compression algorithm is the output of the encryption algorithm. Provide **one** reason why this might be a bad idea.

> **Solution:** Encryption produces high-entropy, random-looking outputs. This means that attempting to compress the ciphertext will likely yield little to no compression.

Q6.2 (5 points) Realizing their mistake, Alice and Bob instead decide to compress their data before encrypting it. They send $\mathsf{Enc}(K, \mathsf{compress}(M))$, where the input to the encryption algorithm is the output of the compression algorithm.

TRUE or FALSE: This combination of compress-then-encrypt is IND-CPA secure. If you answer True, briefly justify your answer (no formal proof needed). If you answer False, describe how an adversary could win the IND-CPA game with probability greater than 0.5.

○ (G) True ● (H) False ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

> **Solution:** Notice that the length of the ciphertext is now dependent on properties of the plaintext (other than the length of the plaintext). Our typical assumption is that the length of the ciphertext only leaks the length of the plaintext, but this is not true with compress-then-encrypt constructions.
>
> The adversary can send $M_0 = 0^{128}$ (128-bit message consisting solely of 0's) and M_1 as a pseudorandom, 128-bit message. If the oracle encrypts M_0, the result of $\mathsf{compress}(M_0)$ will be short, and the resulting ciphertext will be short. If the oracle encrypts M_1, the result of $\mathsf{compress}(M_1)$ will be about as long as the original message, and the resulting ciphertext will be long. The adversary can thus win the IND-CPA game with probability of 1.

For the rest of this question, consider a compress-then-encrypt algorithm with the following properties:

- Assume that Alice and Bob's messages consist of byte strings.
- When compressing, any chain of m consecutive, identical runs of n bytes (total length mn) are compressed into a single run of length n. Runs consisting of a single byte ($n = 1$) are included.
- Encryption preserves the exact length of the message (no padding is used).

For example, the following sequence of bytes:

Index	0	1	2	3	4	5	6	7	8	9
Byte	`0x00`	`0x11`	`0x22`	`0x33`	`0x44`	`0x44`	`0x55`	`0x66`	`0x77`	`0x77`

would be compressed to a message of length 8, since there are 2 1-byte runs in positions 4–5 and 8–9.

Similarly, the following sequence of bytes:

Index	0	1	2	3	4	5	6	7	8	9
Byte	`0xaa`	`0xbb`	`0xaa`	`0xbb`	`0xcc`	`0xdd`	`0xcc`	`0xdd`	`0xee`	`0xff`

would be compressed to 6 bytes, since there are two 2-byte runs in positions 0–3 and two 2-byte runs in positions 4–7.

Q6.3 (5 points) Alice sends the same message $M =$ secret repeatedly to Bob, using the compression algorithm from above for a compress-then-encrypt scheme.

Assume that Mallory doesn't have complete control over the messages that Alice sends, but Mallory is able to *append* to the sent messages before they are compressed and encrypted.

For example, if Alice repeatedly sends the message $M =$ secret, Mallory can force the compressed and encrypted message to be $M' =$ secret$\|x$, where x is chosen by Mallory. She can do this repeatedly for any value of x that she chooses.

If Mallory can observe all resulting ciphertexts from the compress-then-encrypt algorithm as they are transmitted to Bob, devise a way to recover the **last** byte of secret.

Clarification during exam: Assume that Alice's messages will never produce ambiguous behavior when being compressed.

Clarification during exam: Assume that the encryption algorithm preserves the exact length of the plaintext.

Solution: The solution here is to rely on the information leakage through the length of the ciphertext. Observe that, if the last byte of message matches the first byte of Mallory's appended portion of the message x, the message would be shorter than if they were different.

Using this, Mallory only needs to try 256 different messages to learn the last byte of the message. When the ciphertext length is 1 less than for all other possible bytes, we know the chosen byte is the last byte of secret.

Q6.4 (3 points) Assume that Mallory has a method for recovering the last byte of the message. If secret is 32 bytes long and there are 256 possible values for a byte, what is the most number of messages Alice has to send (that Mallory can append to) in order to learn the entire secret?

○ (G) 2^{128} ○ (I) 32^{256} ○ (K) 256^{32}

○ (H) 2^{256} ● (J) 256×32 ○ (L) None of the above

Solution: Once Mallory learned the last byte of secret, she can fix that byte of her message and move on to the next byte. Because the compression algorithm is able to detect arbitrary lengths of runs, if we know that the last byte of the message is x, Mallory can append $y\|x$

to the message, trying every byte y. When y is equal to the second-to-last byte of secret, the message length will be 2 less than it would be otherwise. This process can be repeated for all 32 bytes of the message, totalling 256×32 messages at most.

Q7 *TLS Secret Chaining* **(12 points)**

Recall that TLS with RSA does not provide forward secrecy. An attacker who has recorded past connections and then stolen the server's private key can decrypt any past connection.

Consider a modified TLS scheme. RSA is used for the first connection. In future connections, the premaster secret is instead encrypted using the cipher key from the previous connection. Assume this encryption is IND-CPA secure.

The server and client perform n TLS connections. The first connection is numbered C_1, and the most recent connection is numbered C_n. The attacker records some of these connections and then steals the server's private key. For each set of recorded connections, select all connections the attacker can decrypt.

Clarification during exam: Assume that ranges of connections are inclusive. For example, "All connections between C_1 and C_n" would include C_1 and C_n.

Q7.1 (3 points) The attacker records all connections except C_1.

- ○ (A) Connection C_1 only
- ○ (B) Connections C_1 and C_2 only
- ○ (C) All connections except C_1
- ○ (D) All connections between C_1 and C_{n-1}
- ○ (E) All connections between C_1 and C_n
- ● (F) None of the above

Q7.2 (3 points) The attacker records all connections except C_n.

- ○ (G) Connection C_1 only
- ○ (H) Connections C_1 and C_2 only
- ○ (I) All connections except C_1
- ● (J) All connections between C_1 and C_{n-1}
- ○ (K) All connections between C_1 and C_n
- ○ (L) None of the above

Q7.3 (3 points) The attacker records all connections except C_2.

- ● (A) Connection C_1 only
- ○ (B) Connections C_1 and C_2 only
- ○ (C) All connections except C_1
- ○ (D) All connections between C_1 and C_{n-1}
- ○ (E) All connections between C_1 and C_n
- ○ (F) None of the above

Solution: Notice that, in order to learn the premaster secret for C_i, you need to know the PS for C_{i-1}, since C_i's PS was encrypted with information derived from the PS of C_{i-1}. For C_1, the PS is encrypted with the server's public key, so all connections between C_1 and C_i need to be recorded in order to decrypt connection C_i. Missing just one connection removes the ability to learn the PS for every subsequent connection.

Q7.4 (3 points) Suppose we modify the TLS scheme from above. In every connection, the client encrypts their random number R_b with RSA before sending it to the server. (Recall that in regular TLS, R_b is sent with no encryption.)

Does this scheme provide forward secrecy?

○ (G) Yes, because R_b is needed to generate the symmetric keys

○ (H) Yes, because an attacker can perform a replay attack

● (I) No, because the attacker can still learn R_b after stealing the private key

○ (J) No, because only Diffie-Hellman TLS provides forward secrecy

○ (K) ——

○ (L) ——

Solution: No. If an attacker records the connection and later steals the private key of the connection, they can decrypt the encrypted R_b (as well as the premaster secret) in order to learn the master secret and break secrecy.

Q8 *Intrusion Detection Scenarios* (12 points)

For each scenario below, select the best detector or detection method for the attack.

Q8.1 (3 points) The attacker constructs a path traversal attack with URL escaping: `%2e%2e%2f%2e%2e%2f`.

- ○ (A) NIDS, because of interpretation issues
- ○ (B) NIDS, because of cost
- ● (C) HIDS, because of interpretation issues
- ○ (D) HIDS, because of cost
- ○ (E) ——
- ○ (F) ——

Solution: This path traversal attack is masked using percent encoding in URLs. A traditional NIDS might not recognize this since it is specific to HTTP servers, so a HIDS would be the best option here in order ot avoid the interpretation issues of percent encoding.

Q8.2 (3 points) The attacker is attacking a large network with hundreds of computers, and a detector must be installed as quickly as possible.

- ○ (G) NIDS, because of interpretation issues
- ● (H) NIDS, because of cost
- ○ (I) HIDS, because of interpretation issues
- ○ (J) HIDS, because of cost
- ○ (K) ——
- ○ (L) ——

Solution: A major advantage of NIDS is that they can be quickly installed in order to cover an entire network. Because of the time constraints, the NIDS would be the best in order to mitigate the time cost.

Q8.3 (3 points) The attacker constructs an attack that is encrypted with HTTPS.

- ○ (A) NIDS, because of interpretation issues
- ○ (B) NIDS, because of cost
- ● (C) HIDS, because of interpretation issues
- ○ (D) HIDS, because of cost
- ○ (E) ——
- ○ (F) ——

Solution: A NIDS is not able to decrypt data since it doesn't have the keys that are stored on the host. Thus, only the host can decrypt an interpret the requests, and a HIDS would be the best IDS to use here.

Q8.4 (3 points) The attacker constructs a buffer overflow attack using shellcode they found online in a database of common attacks.

● (G) Signature-based

○ (H) Specification-based

○ (I) Anomaly-based

○ (J) Behavioral

○ (K) ——

○ (L) ——

Solution: This shellcode is easily obtainable and has not been modified, so a signature that matches the exact shellcode would be most effective in detecting this attack.

Q9 *To The Moon* (15 points)

ToTheMoon Bank has just created an online banking system. When a user wants to complete a transfer, they follow these steps:

1. The user logs in by making a POST request with their username and password.
2. The server sets a cookie with `name=auth_user` and `value=$token`, where `$token` is a session token specific to the user's login session.
3. The user initiates a transfer by making a GET request to `https://tothemoonbank.com/transfer?amount=$amount&to=$user`, replacing `$amount` and `$user` with the intended amount and recipient. Transfers use a parameterized SQL query.
4. The server runs the SQL query `SELECT username FROM users WHERE session_token = '$token'`, replacing `token` with the value of the cookie. The server does not use parameterized SQL or any input sanitization.

Q9.1 (4 points) Which of the following attacks are possible in this system? Select all that apply.

■ (A) SQL injection

☐ (B) ROP attack

■ (C) CSRF attack

☐ (D) Path traversal attack

☐ (E) None of the above

☐ (F) ——

Solution: Of the answer options available, the only attacks that are indicated to exist are CSRF in step 3 (because the request does not include a CSRF token). and SQL injection in step 4 (because the server does not use paramaterized SQL or input sanitization).

ROP is a memory safety attack, and path traversal attacks involve filesystems. These are not mentioned in the system, so they aren't indicated to exist.

Q9.2 (4 points) Mallory is a malicious user with an account on ToTheMoon Bank. Mallory creates a malicious link `https://tothemoonbank.com/transfer?amount=100&to=Mallory`.

Which of the following scenarios would cause Alice to send $100 to Mallory? Select all that apply.

☐ (G) Alice clicks on the malicious link when Alice is not logged into the bank

■ (H) Alice clicks on the malicious link when Alice is logged into the bank

☐ (I) Alice visits Mallory's website, which has an `img` tag loading the malicious link, when Alice is not logged into the bank

■ (J) Alice visits Mallory's website, which has an `img` tag loading the malicious link, when Alice is logged into the bank

☐ (K) None of the above

☐ (L) ——

Solution: A CSRF attack would require the user to be logged in on the bank website. After this, any option that makes a GET request would transfer the money, so both an `img` tag and a link would cause this behavior.

Q9.3 (4 points) Suppose Step 4 is modified. To initiate a transfer, instead of making a GET request, the user makes a POST request to `https://tothemoonbank.com/transfer` with the amount and recipient in the POST body.

Which of the following scenarios would cause Alice to send $100 to Mallory? Select all that apply.

Clarification during exam: The malicious link in the answer choices should be `https://tothemoonbank.com/transfer?amount=100&to=Mallory`.

☐ (A) Alice clicks on the malicious link when Alice is not logged into the bank

☐ (B) Alice clicks on the malicious link when Alice is logged into the bank

☐ (C) Alice visits Mallory's website, which has an `img` tag loading the malicious link, when Alice is not logged into the bank

☐ (D) Alice visits Mallory's website, which has an `img` tag loading the malicious link, when Alice is logged into the bank

■ (E) None of the above

☐ (F) ——

Solution: Neither the `img` tag nor the malicious link would cause a POST request to be sent, so none of these options would cause a transfer to take place.

Q9.4 (3 points) Which user inputs might be vulnerable to SQL injection? Select all that apply.

☐ (G) `amount` parameter

☐ (H) `to` parameter

■ (I) Value of the `auth_user` cookie

☐ (J) None of the above

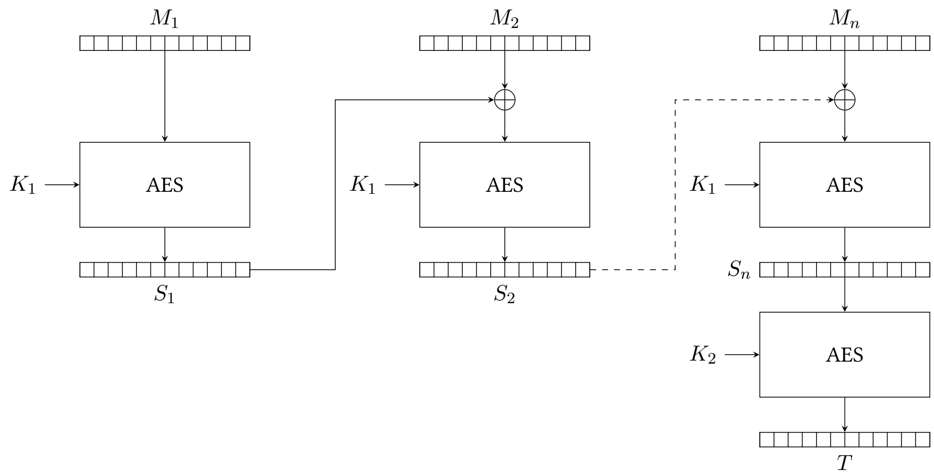
☐ (K) ——

☐ (L) ——

Solution: The question indicates that only the token query in step 4 is vulnerable to SQL injection, so only the value of the `auth_user` cookie would be vulnerable.

Q10 ***AES-EMAC*** **(19 points)**

Consider AES-EMAC, which is another scheme for generating secure MACs.



Q10.1 (2 points) TRUE or FALSE: Given only T, an attacker can generate a valid MAC for $M||M'$, for an M' of the attacker's choosing.

○ TRUE ● FALSE

Solution: False. Without the keys, the attacker cannot compute further intermediate states S_i.

Q10.2 (2 points) TRUE or FALSE: Given only T and K_1, an attacker can generate a valid MAC for $M||M'$, for an M' of the attacker's choosing.

○ TRUE ● FALSE

Solution: False. Without K_2, the attacker cannot output a valid final state T, and the attacker cannot reverse the final AES encryption to learn the intermediate state S_n.

Q10.3 (2 points) TRUE or FALSE: Given only T, K_1, and K_2, an attacker can generate a valid MAC for $M||M'$, for an M' of the attacker's choosing.

● TRUE ○ FALSE

Solution: True. First, the attacker can compute $D(T, K_2)$ to retrieve S_n. (In other words, the attacker computes AES decryption on T with key K_2).

Then, the attacker adds more intermediate state $S_{n+1}, S_{n+2}, \ldots$ by performing more AES encryptions using K_1.

Finally, the attacker encrypts the last S_i block with K_2 to produce the final MAC.

Q10.4 (2 points) TRUE or FALSE: Given the output T and the secret keys K_1 and K_2, the entire original message M can be reconstructed.

◯ TRUE ● FALSE

> **Solution:** False. One way to see this is to note that the output is only one block long, while the message is n blocks long. There are not enough bits to reconstruct one unique message given the MAC output.

For the rest of the question, regardless of your answer to the previous parts, assume that the output is $S_1||S_2 \dots ||S_n||T$.

Q10.5 (4 points) Which values are needed to recover the entire original message? Select all that apply.

- ■ (A) S_i for all $1 \leq i \leq n$
- □ (B) T
- ■ (C) K_1
- □ (D) K_2
- □ (E) None of the above
- □ (F) ——

> **Solution:** Notice that this mode is very similar to CBC mode with a constant 0 IV. In order to recover the message, the output of each encryption block is needed, which corresponds to the states S_1 to S_n. In CBC, decryption requires the original key, and K_1 corresponds to this key in this mode. Neither K_2 nor T are necessary since they would only be used to recover S_n, which is already given to us.

Q10.6 (3 points) Which of these equations is correct for calculating a plaintext block M_i given the output and both keys K_1 and K_2?

- ● (G) $M_i = \text{Dec}(K_1, S_i) \oplus S_{i-1}$
- ◯ (H) $M_i = \text{Dec}(K_1, S_i \oplus S_{i-1})$
- ◯ (I) $M_i = \text{Dec}(K_2, S_i) \oplus S_{i-1}$
- ◯ (J) $M_i = \text{Dec}(K_2, S_i \oplus S_{i-1})$
- ◯ (K) $M_i = \text{Enc}(K_1, S_i) \oplus S_{i-1}$
- ◯ (L) $M_i = \text{Enc}(K_1, S_i \oplus S_{i-1})$

> **Solution:** By inspecting the diagram, we can write out the equation for encryption: $S_i = E(K_1, M_i \oplus S_{i-1})$. Note that it is very similar to AES-CBC.
>
> Now we can solve for M_i to produce the decryption algorithm:
>
> $$\begin{aligned} S_i &= E(K_1, M_i \oplus S_{i-1}) && \text{Encryption equation} \\ D(K_1, S_i) &= M_i \oplus S_{i-1} && \text{Decrypt both sides} \\ D(K_1, S_i) \oplus S_{i-1} &= M_i && \text{XOR both sides with } S_{i-1} \end{aligned}$$

Q10.7 (4 points) Select all true statements about this scheme.

■ (A) To compute AES-EMAC on a message, the message must first be padded to a multiple of the block size

☐ (B) Encryption can be parallelized

■ (C) Decryption can be parallelized

☐ (D) AES-EMAC is IND-CPA secure

☐ (E) None of the above

☐ (F) ——

Solution:

A: True. The message is split into blocks and each block is encrypted with AES, so the message must be padded to a multiple of the block size.

B: False. Encrypting block i requires the ciphertext from block $i - 1$, so before we can encrypt block i, we must first wait for block $i - 1$ to be encrypted.

C: True. The decryption algorithm only depends on ciphertext blocks, not plaintext blocks, and all ciphertext blocks are known at the beginning of decryption.

D: False. There is no randomness involved, so the scheme is deterministic. Deterministic schemes cannot be IND-CPA secure.

Q11 *DNS Lookups* (15 points)

EvanBot performs a lookup for the IP address of `toon.cs161.org`. For each DNS or DNSSEC record, determine which name server sent the record.

Q11.1 (3 points) `A` type record with the IP address of `toon.cs161.org`

- ○ (A) root name server
- ○ (B) `.org` name server
- ● (C) `cs161.org` name server
- ○ (D) None of the above
- ○ (E) —
- ○ (F) —

> **Solution:** This is the final answer record, which is sent by the `cs161.org` name server.

Q11.2 (3 points) `A` type record with the IP address of the `cs161.org` name server

- ○ (G) root name server
- ● (H) `.org` name server
- ○ (I) `cs161.org` name server
- ○ (J) None of the above
- ○ (K) —
- ○ (L) —

> **Solution:** In this record, the `cs161.org` name server's parent name server is redirecting the resolver to the `cs161.org` name server. The parent of the `cs161.org` name server is the `.org` name server.

Q11.3 (3 points) `A` type record with the IP address of `cs161.org` (not the name server)

- ○ (A) root name server
- ○ (B) `.org` name server
- ○ (C) `cs161.org` name server
- ● (D) None of the above
- ○ (E) —
- ○ (F) —

> **Solution:** The IP address of `cs161.org` was not queried for, and `cs161.org` is not a name server, so this record will not be sent in this DNS lookup.

Q11.4 (3 points) `DNSKEY` type record with the public key of the `cs161.org` name server

- ○ (G) root name server
- ○ (H) `.org` name server
- ● (I) `cs161.org` name server
- ○ (J) None of the above
- ○ (K) —
- ○ (L) —

Solution: In DNSSEC, each name server sends its own public keys, so this record comes from the `cs161.org` name server.

Q11.5 (3 points) `DS` type record with the hash of the `.org` name server's public key

- ● (A) root name server
- ○ (B) `.org` name server
- ○ (C) `cs161.org` name server
- ○ (D) None of the above
- ○ (E) ——
- ○ (F) ——

Solution: In DNSSEC, the `DS` type record is used by the parent to endorse the child's public key. In this record, the child is the `.org` name server, so the parent must be the root name server.

Q12 *Wallet Management* (26 points)

Consider the following vulnerable C code:

```c
#include <stdio.h>
#include <stdlib.h>

struct wallet {
    char owner[4];   /* 4 bytes. */
    int amt;         /* 4 bytes. */
};

int main(void) {
    int wallet_idx = 0;
    struct wallet wals[8];
    char buf[16];

    while (1) {
        /* Get wallet index. */
        printf("Enter wallet index:\n");
        fgets(buf, 16, stdin);
        int wallet_idx = atoi(buf);
        if (wallet_idx < 0) {
            /* Exit loop if invalid index. */
            break;
        }

        /* Update dollar amount. */
        printf("Enter dollar amount:\n");
        fgets(buf, 16, stdin);
        wals[wallet_idx].amt = atoi(buf);

        /* Read owner. */
        printf("Enter owner name:\n");
        gets(wals[wallet_idx].owner);
    }

    return 0;
}
```

Assume you are on a little-endian 32-bit x86 system. Assume that there is no compiler padding or additional saved registers in all subparts.

Clarification during exam: The `atoi` function converts a string to an integer. For example, `atoi("3")` would return the integer value 3.

Q12.1 (4 points) For the first subpart, assume that **no memory safety defenses** are enabled.

Let `SHELLCODE` be a 24-byte malicious shellcode. If the address of `wals` is 0x9fffcad0, provide an input as a series of Python `print` statements that would cause your program to execute malicious shellcode.

For example, a sequence of non-malicious inputs that updates wallet 0 without exiting the loop would be:

```
print('0')
print('1000')
print('NN')
```

Write your answer in Python 2 syntax (just like in Project 1).

Solution: This is a standard buffer overflow. First, we need to input a `'0'` to start writing at the beginning of the buffer. Next, we need to input any number to get past the call to update the number of dollars in the wallet. Then, we input our exploit that overflows the buffer in the call to `gets`. The RIP is located 36 bytes after the start of `wals`, so we can input the shellcode, followed by 12 dummy bytes, followed by the address of our shellcode. Finally, we need to exit the loop by inputting an invalid index to actually return to the RIP and execute our shellcode:

```
print('0')
print('1234')
print(SHELLCODE + 'A' * 44 + '\xd0\xca\xff\x9f')
print('-1');
```

For the remaining parts of this question, assume that **stack canaries are enabled**.

Q12.2 (3 points) Complete the stack diagram with stack canaries enabled. Each row represents 4 bytes.

Parts (2a), (2b), and (2c):

RIP of `main`
(2a)
(2b)
(2c)
(3a)
(3b)
(3c)
(3d)

○ (G) (2a) - SFP of `main`; (2b) - `wallet_idx`; (2c) - canary

● (H) (2a) - SFP of `main`; (2b) - canary; (2c) - `wallet_idx`

○ (I) (2a) - canary; (2b) - SFP of `main`; (2c) - `wallet_idx`

○ (J) (2a) - canary; (2b) - `wallet_idx`; (2c) - SFP of `main`

○ (K) (2a) - `wallet_idx`; (2b) - SFP of `main`; (2c) - canary

○ (L) (2a) - `wallet_idx`; (2b) - canary; (2c) - SFP of `main`

Q12.3 (3 points) Parts (3a), (3b), (3c), and (3d):

◯ (A) (3a) - `wals[0].amt`; (3b) - `wals[0].owner`; (3c) - `wals[1].amt`; (3d) - `wals[1].owner`

◯ (B) (3a) - `wals[0].owner`; (3b) - `wals[0].amt`; (3c) - `wals[1].owner`; (3d) - `wals[1].amt`

● (C) (3a) - `wals[7].amt`; (3b) - `wals[7].owner`; (3c) - `wals[6].amt`; (3d) - `wals[6].owner`

◯ (D) (3a) - `wals[7].owner`; (3b) - `wals[7].amt`; (3c) - `wals[6].owner`; (3d) - `wals[6].amt`

◯ (E) ——

◯ (F) ——

Solution: The full stack diagram is as follows:

RIP of `main`
SFP of `main`
canary
`wallet_idx`
`wals[7].amt`
`wals[7].owner`
`wals[6].amt`
`wals[6].owner`

Q12.4 (3 points) Describe, briefly, a vulnerability on line 19. Additionally, describe a one-line fix to line 19 that would make this code memory-safe.

Solution: Line 19 fails to check for a valid index. While it ensures that the index will never be negative, it should also check that the user does not index past the end of the `wals` array. Thus, a one-line fix would be to replace the check with

```
if (wallet_idx < 0 || wallet_idx >= 16)
```

Q12.5 (5 points) Let `SHELLCODE` be a 24-byte malicious shellcode. If the address of the RIP of `main` is 0xbfeffc80, provide an input as a series of Python `print` statements that would cause your program to execute malicious shellcode.

Solution: Now that we have a stack canary in the way, we need be a bit smarter about how we overwrite the RIP. Notice: Because of the indexing vulnerability, we have the ability to write to any address in memory above the start of the `wals` array by choosing an invalid index.

Each `struct wallet` is 8 bytes long, so the `wallet_idx` variable and the canary would appear to be `wals[8]`, and the SFP and RIP of main would appear to be `wals[9]`.

First, we input `'9'` to maliciously index into the SFP and RIP. Next, we input any number to update the number of dollars. Then, we input our exploit: 4 dummy bytes to overwrite the SFP, then input the address of SHELLCODE, which will be 4 bytes after the RIP itself (thus using the address of RIP + 4), then the shellcode. Finally, we input an invalid index to exit the loop:

```
print('9')
print('1234')
print('A' * 4 + '\x84\xfc\xef\xbf' + SHELLCODE)
print('-1')
```

Q12.6 (4 points) Assume that the call to `gets` on line 31 is replaced with `fread(wals[wallet_idx].owner, 1, 4, stdin)`. Recall that `fread` does stop reading at a newline and does not add a NULL terminator.

EvanBot thinks that this code is no longer exploitable because the `gets` function is no longer used. **Assume that your 24-byte shellcode must be written in a contiguous block of memory.** Is EvanBot correct? If you answer Yes, describe why you can no longer exploit this code. If you answer No, describe how you would write your shellcode to a contiguous block of memory.

○ (G) Yes ● (H) No ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Solution: In fact, even though `fread` only directly lets you write every other block of 4 bytes (since it only controls the 4-byte `owner` field of an 8-byte struct), you can still control a contiguous block of memory through the `amt` field. By treating 4 bytes of shellcode as a 32-bit integer and outputting its decimal representation, the `atoi` function will convert the decimal representation back into the raw bits of the shellcode, which will then be written to memory.

Thus, the full algorithm is as follows: For parts of memory written using the `owner` field, simply output the raw bytes to be read by the `fread` call. For parts of memory written using the `amt` field, treat the 4 bytes as a 32-bit integer and output its decimal representation. This allows the attacker to control a contiguous block of memory to place the shellcode.

An alternative solution is to pass the shellcode as an argument or an environment variable to the program, causing it to be placed in memory.

Q12.7 (4 points) This part is independent of the previous subpart, but assume that stack canaries are still enabled. Which of the following actions would individually prevent the attacker from executing malicious shellcode (not necessarily using the exploit from above)?

Clarification during exam: The answer choice "Enabling pointer authentication" has been dropped. All students will receive credit for that answer choice only.

☐ (A) Enabling non-executable pages in addition to stack canaries

☐ (B) Enabling pointer authentication

☐ (C) Replacing the call to `gets` on line 31 with `fgets(wals[wallet_idx].owner, 4, stdin)`

☐ (D) Swapping the positions of the `owner` and `amt` fields in the `wallet` struct

■ (E) None of the above

☐ (F) ——

Solution: A: False. Non-executable pages is trivial to bypass in most circumstances using return-to-libc (which is clearly used in the program) or a ROP attack.

B: Because pointer authentication is only defined in 64-bit systems, and this question uses a 32-bit system, this answer choice is ambiguous, so we gave points to everyone on this answer choice.

C: False. The RIP of `main` can still be overwritten by indexing into `wals[9]`, and shellcode could be placed elsewhere under the control of the attacker, such as in the environment variables or in the arguments to the program (just like in Project 1 Question 4).

D: False. The call to `gets` is still vulnerable to a buffer overflow. The attacker just needs to account for the fact that they are now starting to write 4 bytes higher than before.

C Function Definitions

```
int printf(const char *format, ...);

       printf() produces output according to the format string format.

char *gets(char *s);

       gets()  reads  a  line from stdin into the buffer pointed to by s until
       either a terminating newline or EOF, which it replaces with a null byte
       ('\0').

char *fgets(char *s, int size, FILE *stream);

       fgets()  reads in at most one less than size characters from stream and
       stores them into the buffer pointed to by s.  Reading  stops  after  an
       EOF  or a newline.  If a newline is read, it is stored into the buffer.
       A terminating null byte ('\0') is stored after the  last  character  in
       the buffer.

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

       The function fread() reads nmemb items of data, each size bytes long,
       from the stream pointed to by stream, storing them at the location
       given by ptr.

       Note that fread() does not add a null byte after input.
```